HP 64793

H8/338/329 Emulator Softkey Interface

User's Guide

HP Part No. 64793-97002
Printed in U.S.A.
October 1992

Edition 1

Notice

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

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

UNIX is a registered trademark of AT&T.

Torx is a registered trademark of Camcar Division of Textron, Inc.

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

Printing History

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

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

Edition 1 **64793-97002, October 1992**

Using This Manual

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

- HP 64793A H8/338 emulator
- HP 64793B H8/329 emulator

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

Model	Supported Microprocesorts	Reffered to as
HP 64793A (H8/338 emulator)	HD6473388CP HD6433388CP HD6413388CP HD6473378CP HD6433378CP HD6413378CP HD6433368CP	H8/338 H8/338 H8/338 H8/337 H8/337 H8/337 H8/336
HP 64793B (H8/329 emulator)	HD6473298P HD6473298C HD6433298P HD6413298P HD6433288P HD6473278P HD6473278C HD6433278P HD6413278P HD6433268P	H8/329 H8/329 H8/329 H8/329 H8/328 H8/327 H8/327 H8/327 H8/327 H8/326

For the most part, the H8/329 and H8/338 emulators all operate the same way. Differences between the emulators are described where they exist. Both the H8/329 and H8/338 emulators will be referred to as the "H8/338 emulator". In the specific instances where H8/329 emulator

differs from H8/338 emulator, it will be described as the "H8/329 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 does not:

- Show you how to use every Softkey Interface command and option; the Softkey Interface is described in the *Softkey Interface Reference.*

Organization

Chapter 1 **Introduction.** This chapter lists the H8/338 emulator features and describes how they can help you in developing new hardware and software.

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 plug the emulator into a target system, and how to use the "in-circuit" emulation features.

Chapter 4 **Configuring the Emulator.** You can configure the emulator to adapt it to your specific development needs. This chapter describes the options available when configuring the emulator and how to save and restore particular configurations.

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

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

Illustrations

1

Introduction to the H8/338 Emulator

Introduction

The topics in this chapter include:

- Purpose of the H8/338 emulator.
- Features of the H8/338 emulator.

Purpose of the H8/338 Emulator

The H8/338 emulator is designed to replace the H8/338 microprocessor in your target system so you can control operation of the microprocessor in your application hardware (usually referred to as the *target system*). The H8/338 emulator performs just like the H8/338 microprocessor, but is a device that allows you to control the H8/338 directly. These features allow you to easily debug software before any hardware is available, and ease the task of integrating hardware and software.

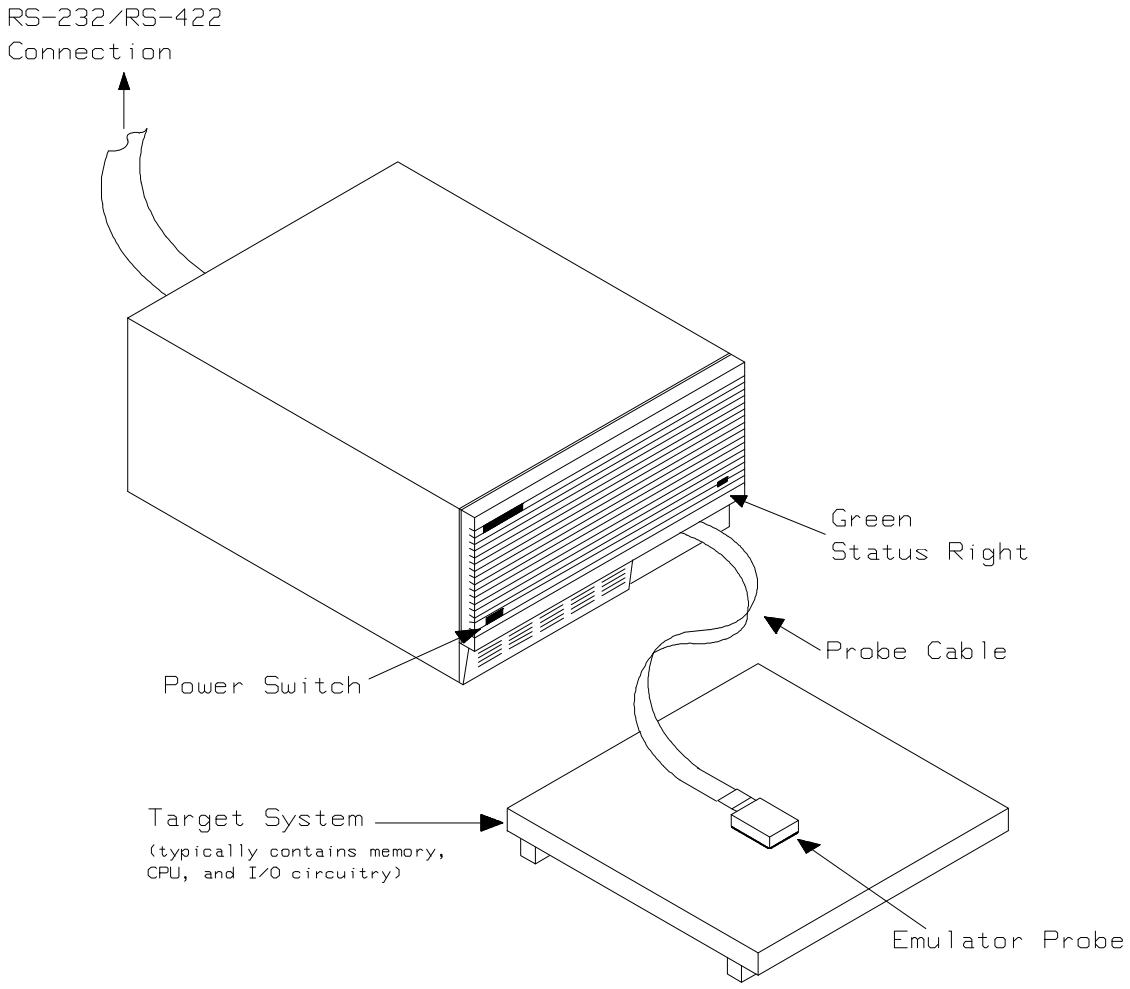


Figure 1-1. HP 64793 Emulator for the H8/338 Processor

1-2 Introduction to the H8/338 Emulator

Features of the H8/338 Emulator

Supported Microprocessors

The HP 64793A H8/338 emulator and HP 64793B H8/329 emulators support the microprocesors listed in the following table.

Model	Supported Microprocessor
HP 64793A (H8/338 emulator)	HD6473388CP (H8/338) HD6433388CP (H8/338) HD6413388CP (H8/338) HD6473378CP (H8/337) HD6433378CP (H8/337) HD6413378CP (H8/337) HD6433368CP (H8/336)
HP 64793B (H8/329 emulator)	HD6473298P (H8/329) HD6473298C (H8/329) HD6433298P (H8/329) HD6413298P (H8/329) HD6433288P (H8/328) HD6473278P (H8/327) HD6473278C (H8/327) HD6433278P (H8/327) HD6413278P (H8/327) HD6433268P (H8/326)

Each model provides with an emulation probe designed for its support microprocessors. By replacing the emulation probe, the HP64793 can support processors other than its original support processors. Contact Hewlett-Packard to replace the emulation probe.

Clock Speeds

Maximum clock speed is 10 MHz (system clock).

Emulation Memory

The HP64793 H8/338 emulator is used with the following Emulation Memory Card.

- HP 64725A 128K byte Emulation Memory Card

The emulation memory can be configured into 128 byte blocks. A maximum of 16 ranges can be configured as emulation RAM(eram), emulation ROM(erom), target system RAM(tram), target system ROM(trom), or guarded memory(grd). The H8/338 emulator will attempt to break to the emulation monitor upon accessing guarded memory, additionally, you cam configure the emulator to break to the emulation monitor upon performing a write to ROM(which will stop a runaway program).

Analysis

The HP64793 H8/338 emulator is used with one of the following analyzers which allows you to trace code execution and processor activity.

- HP 64703 64-channel Emulation Bus Analyzer and 16-channel State/Timing Analyzer
- HP 64704 80-channel Emulation Bus Analyzer
- HP 64706 48-channel Emulation Bus Analyzer

The Emulation Bus Analyzer monitors the emulation processor using an internal analysis bus. The HP64703 64-channel Emulation Bus Analyzer and 16-channel State/Timing Analyzer allows you to probe up to 16 different lines in your target system.

Register Display and Modification

You can display or modify the H8/338 internal register contents. This includes the ability to modify the program counter (PC) value 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.

Breakpoints

You can set the emulator/analyzer interaction so the emulator will break to the monitor program when the analyzer finds a specific state, allowing you to perform post-mortem analysis of the program execution.

You can also define software breakpoints in your program. The emulator uses an H8/338 special code to provide software breakpoints; This special code (5770 hexadecimal) is H8/338 undefined instruction. When you define a software breakpoint, the emulator places the special code at the specified address; after the special code causes emulator execution to break out of the user program (into the monitor), the emulator replaces the original opcode. See the "Using Software Breakpoints" section of the "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.

Real-Time Execution

Real-time signifies continuous execution of your program at full rated processor speed 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 target system memory; load/dump target system memory; and display or modify registers,and single step.

Limitations, Restrictions

Foreground Monitor

Foreground Monitor is not supported for the H8/338 emulator.

Monitor Break at Sleep/Standby Mode

When the emulator breaks into the emulation monitor, sleep or software standby mode is released.

Store Condition and Trace

Disassembling of program execution in the trace list may not be accurate when the emulation analyzer is used with store condition. Refer to chapter 2 of this manual for more information.

Step Command and Interrupts

Step execution cannot be performed in the following cases.

- When the emulator is in the monitor and a suspended interrupt is existed.
- When the emulator is in the monitor and a level sensed interrupt is existed (including interrupts from internal I/O device).

Refer to Chapter 5 of this manual.

RAM Enable Bit

The internal RAM of H8/338 processor can be enabled/disabled by RAME (RAM enable bit). However, once you map the internal RAM area to emulation RAM, the emulator still accesses emulation RAM even if the internal RAM is disabled by RAME.

Software Performance Measurement

Program Activity Measurement using the Software Performance Measurement Tool (SPMT) is valid only for H8/338 internal ROM area. (that is, 0 hex through 3fff hex.) Outside this area, the result of Program Activity Measurement is not reliable

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 H8/338 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 Installation/Service* manual show you how to do this.

2. Installed the Softkey Interface software on your computer. Refer to the *HP 64700 Series Installation/Service* manual for instructions on installing software.

3. In addition, you should read and understand the concepts of emulation presented in the *Concepts of Emulation and Analysis* manual. The *Installation/Service* manual also covers HP64700 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 H8/338 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/hp64793/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          "THIS IS MESSAGE A"
Msg_B           .SDATA          "THIS IS MESSAGE B"
Msg_I           .SDATA          "INVALID COMMAND"
End_Msgs

                .SECTION        Prog,CODE
;*****************************************************
;*  Set up the Stack Pointer
;*****************************************************
Init            MOV.W           #Stack,R7
;*****************************************************
;* Clear previous command
;*****************************************************
Clear           MOV.B           #H'00,R0L
                MOV.B           R0L,@Cmd_Input
;*****************************************************
;* Read command input byte.  If no command has been
;* entered, continue to scan for it.
;*****************************************************
Scan            MOV.B           @Cmd_Input,R2L
                CMP.B           #H'00,R2L
                BEQ             Scan
;*****************************************************
;* A command has been entered.  Check if it is
;* command A, command B, or invalid command.
;*****************************************************
Exe_Cmd         CMP.B           #H'41,R2L
                BEQ             Cmd_A
                CMP.B           #H'42,R2L
                BEQ             Cmd_B
                BRA             Cmd_I
;*****************************************************
;* Command A is entered.  R3L= the number of bytes
;* in message A.  R4 = location of the message.
;* Jump to the routine which writes the message.
;*****************************************************
Cmd_A           MOV.B           #Msg_B-Msg_A,R3L
                MOV.W           #Msg_A,R4
                BRA             Write_Msg
;*****************************************************
;* Command B is entered.
;*****************************************************
Cmd_B           MOV.B           #Msg_I-Msg_B,R3L
                MOV.W           #Msg_B,R4
                BRA             Write_Msg
;*****************************************************
;* An invalid command is entered.
;*****************************************************
Cmd_I           MOV.B           #End_Msgs-Msg_I,R3L
                MOV.W           #Msg_I,R4
```

Figure 2-1. Sample Program Listing

```
;*************************************************
;* The destination area is cleared.
;*************************************************
;
Write_Msg       MOV.W           #Msg_Dest,R5
Clear_Old       MOV.B           #h'20,R6L
Clear_Loop      MOV.B           R0L,@R5
                ADDS.W          #1,R5
                DEC.B           R6L
                BNE             Clear_Loop
;*************************************************
;* Message is written to the destination.
;*************************************************
;
                MOV.W           #Msg_Dest,R5
Write_Loop      MOV.B           @R4+,R6L
                MOV.B           R6L,@R5
                ADDS.W          #1,R5
                DEC.B           R3L
                BNE             Write_Loop
;*************************************************
;
;* Go back and scan for next command.
;*************************************************
;
                BRA             Clear

                .SECTION        Data,COMMON
;*************************************************
;
;* Command input byte.
;*************************************************
;
Cmd_Input       .RES.B          1
                .RES.B          1
;*************************************************
;
;* Destination of the command messages.
;*************************************************
;
Msg_Dest        .RES.W          H'7f
Stack
                .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 **Clear** 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 R3L 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**.

Prior to writing the message, **Clear_Old** clears the destination area. After the message is written, 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.

Sample Program Assembly

The sample program is written for and assembled with the HP 64876 H8/300 Assembler/Linkage Editor. The sample program was assembled with the following command (which assumes that **/usr/hp64000/bin** is defined in the PATH environment variable).

```
$ h83asm -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(0fc00)
print cmd_rds
output cmd_rds
exit
```

Figure 2-2. Linkage Editor Subcommoand File

Generate HP Absolute file

To generate HP Absolute file for the Softkey Interface, you need to use **"h83cnvhp"** absolute file format converter program. The h8cnvhp converter is provided with HP 64876 H8/300 Assembler/Linkage Editor. To generate HP Absolute file, enter following command:

```
$ h83cnvhp 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. Otherwise, you will see the warning message when file format converter **h83cnvhp** is executed. And no local symbol file will be generated. 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 H8/338 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 H8/338 emulator, enter:

```
make_sys emh8338 <RETURN>
```

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

```
add <module_number> naming_it h8338
<RETURN>
```

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

```
end <RETURN>
```

If the measurement system and emulation module are named "emh8338" and "h8338" as shown above, you can enter the emulation system with the following command:

```
emh8338 default h8338 <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).

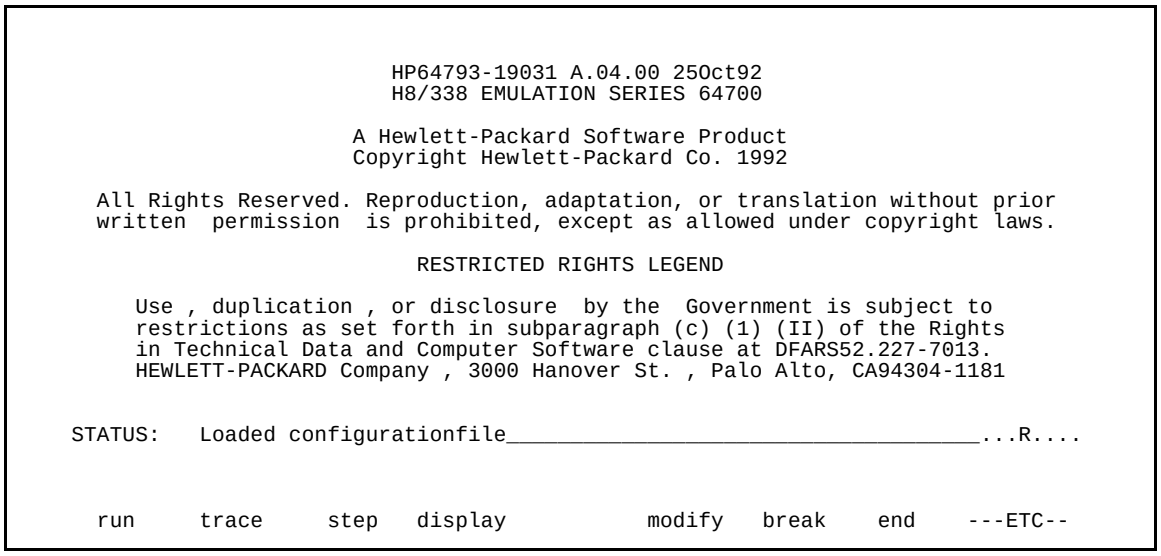


Figure 2-3. Softkey Interface Display

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. In the case of H8/338 emulator, the address range 0 hex through 3fff hex is mapped as emulation ROM and fd80 hex through ff7f hex as emulation RAM. In the case of H8/325 emulator, the address range 0 hex through 7fff hex is mapped as emulation ROM and fb80 hex through ff7f hex as emulation RAM.

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)
 --More--(20%)
```

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:   H8/338--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.

```
Global symbols in cmd_rds
Static symbols
Symbol name ____________________ Address range ___ Segment ____________ Offset
Cmd_Input                        FC00                                      0000
Init                             1000                                      0000
Msg_Dest                         FC02                                      0002
Msgs                             2000                                      0000

Filename symbols
Filename ____________________________________________________________________
cmd_rds.src









STATUS:   H8/338--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>

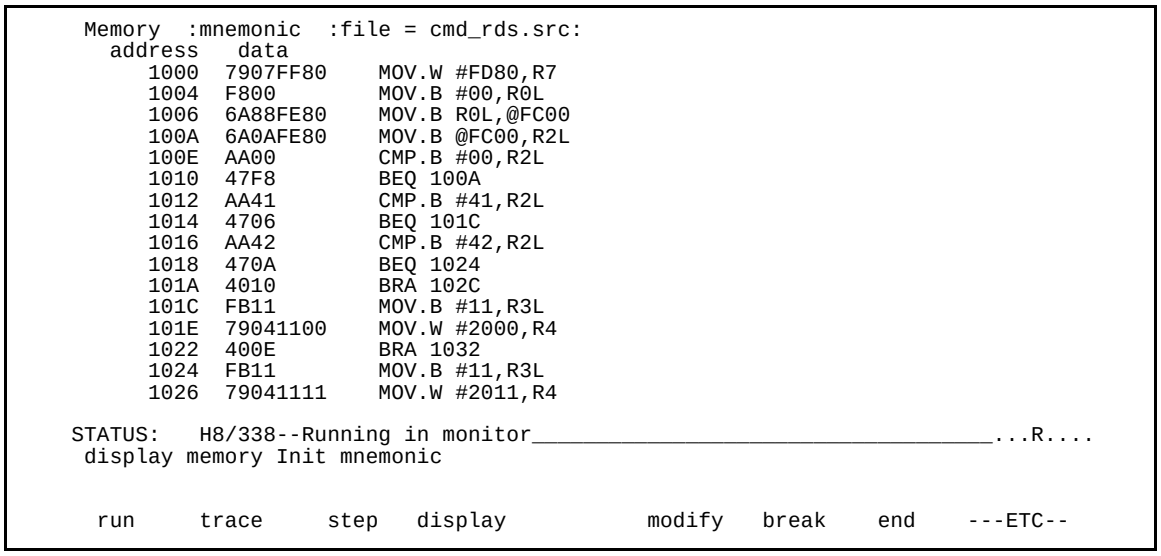
```
Symbols in cmd_rds.src:
Static symbols
Symbol name ____________________ Address range ___ Segment _____________ Offset
Clear                             1004                                       0004
Clear_Loop                        1038                                       0038
Clear_Old                         1036                                       0036
Cmd_A                             101C                                       001C
Cmd_B                             1024                                       0024
Cmd_I                             102C                                       002C
Cmd_Input                         FC00                                       0000
Data                              FC00                                       0000
END_Msgs                       00002031
Exe_Cmd                           1012                                       0012
Init                              1000                                       0000
Msg_A                             2000                                       0000
Msg_B                             2011                                       0011
Msg_Dest                          FC02                                       0002
Msg_I                             2022                                       0022

STATUS:  cws: cmd_rds.src:_____________________________________________...R....
 display local_symbols_in cmd_rds.src:


  run     trace     step   display           modify   break    end    ---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.



Displaying Memory with Symbols

You can include symbol information in memory display.

set symbols on <RETURN>

Note

The "**set**" command is effective only to the window which the command is invoked. When you access the emulator from multiple windows, you need to use the command at each window.

```
 Memory  :mnemonic  :file = cmd_rds.src:
   address  label          data
      1000     :Init      7907FF80    MOV.W #FD00,R7
      1004  cmd_rd:Clear  F800        MOV.B #00,R0L
      1006                6A88FE80    MOV.B R0L,@:Cmd_Input
      100A  cmd_rds:Scan  6A0AFE80    MOV.B @:Cmd_Input,R2L
      100E                AA00        CMP.B #00,R2L
      1010                47F8        BEQ cmd_rds.src:Scan
      1012  cmd_:Exe_Cmd  AA41        CMP.B #41,R2L
      1014                4706        BEQ cmd_rds.sr:Cmd_A
      1016                AA42        CMP.B #42,R2L
      1018                470A        BEQ cmd_rds.sr:Cmd_B
      101A                4010        BRA cmd_rds.sr:Cmd_I
      101C  cmd_rd:Cmd_A  FB11        MOV.B #11,R3L
      101E                79041100    MOV.W #2000,R4
      1022                400E        BRA cmd_rd:Write_Msg
      1024  cmd_rd:Cmd_B  FB11        MOV.B #11,R3L
      1026                79041111    MOV.W #2011,R4

STATUS:   H8/338--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:

```
 Memory  :bytes :blocked :repetitively
      address    data       :hex                                  :ascii
      FC02-09    54   48    49   53   20   49   53   20      T H I S    I S
      FC0A-11    4D   45    53   53   41   47   45   20      M E S S  A G E
      FC12-19    41   00    00   00   00   00   00   00      A . . .  . . . .
      FC1A-21    00   00    00   00   00   00   00   00      . . . .  . . . .
      FC22-29    00   00    00   00   00   00   00   00      . . . .  . . . .
      FC2A-31    00   00    00   00   00   00   00   00      . . . .  . . . .
      FC32-39    00   00    00   00   00   00   00   00      . . . .  . . . .
      FC3A-41    00   00    00   00   00   00   00   00      . . . .  . . . .
      FC42-49    00   00    00   00   00   00   00   00      . . . .  . . . .
      FC4A-51    00   00    00   00   00   00   00   00      . . . .  . . . .
      FC52-59    00   00    00   00   00   00   00   00      . . . .  . . . .
      FC5A-61    00   00    00   00   00   00   00   00      . . . .  . . . .
      FC62-69    00   00    00   00   00   00   00   00      . . . .  . . . .
      FC6A-71    00   00    00   00   00   00   00   00      . . . .  . . . .
      FC72-79    00   00    00   00   00   00   00   00      . . . .  . . . .
      FC7A-81    00   00    00   00   00   00   00   00      . . . .  . . . .

STATUS:   H8/338--Running user program_____________________________...R....
 modify memory Cmd_Input bytes to 41h


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

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

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

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 an H8/338 special code; This special code (5770 hexadecimal) is H8/338 undefined instruction.

When you define or enable a software breakpoint, the emulator will replace the opcode at the software breakpoint address with the special code.

Note

You must set software breakpoints only 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 special code, you cannot define software breakpoints in target ROM.

When software breakpoints are enabled and emulator detects a fetching the special code (5770 hexadecimal), 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 special code is software breakpoints or opcode in your target program.

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

If the special code is 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 special code with the original opcode.

Up to 32 software breakpoints may be defined.

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 H8/338 special code (5770 hexadecimal) 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_A** label, enter the following command.

modify software_breakpoints set Cmd_A <RETURN>

Notice that when using local symbols in expressions, the source file in which the local symbol is defined must be included.

After the software breakpoint has been set, enter the following command to display memory and see if the software breakpoint was correctly inserted.

display memory Init ***memonic*** <RETURN>

```
 Memory  :mnemonic  :file = cmd_rds.src:
   address  label          data
      1000     :Init      7907FF80    MOV.W #FF80,R7
      1004  cmd_rd:Clear  F800        MOV.B #00,R0L
      1006                6A88FE80    MOV.B R0L,@:Cmd_Input
      100A  cmd_rds:Scan  6A0AFE80    MOV.B @:Cmd_Input,R2L
      100E                AA00        CMP.B #00,R2L
      1010                47F8        BEQ cmd_rds.src:Scan
      1012  cmd_:Exe_Cmd  AA41        CMP.B #41,R2L
      1014                4706        BEQ cmd_rds.sr:Cmd_A
      1016                AA42        CMP.B #42,R2L
      1018                470A        BEQ cmd_rds.sr:Cmd_B
      101A                4010        BRA cmd_rds.sr:Cmd_I
*     101C  cmd_rd:Cmd_A  5770        Illegal Opcode
      101E                79041100    MOV.W #1100,R4
      1022                400E        BRA cmd_rd:Write_Msg
      1024  cmd_rd:Cmd_B  FB11        MOV.B #11,R3L
      1026                79041111    MOV.W #1111,R4

STATUS:   H8/338--Running in monitor_____________________________________...R....
 display memory Init mnemonic


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

As you can see, the software breakpoint is shown in the memory display with an asterisk.

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*** 41h <RETURN>

You will see the line of the software breakpoint is displayed in

inverse-video. The inverse-video shows that the Program Counter is now at the address.

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.

When software breakpoints are hit, they become inactivated. To reactive the breakpoint so that is "pending", you must enter the "modify software_breakpoint set" command again.

Clearing a Software Breakpoint

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

```
modify software_breakpoints clear
Cmd_A <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>
```

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>, ...
```

You will see the inverse-video moves according to the step execution. 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>.

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 1022
 PC 1022  SP FF80  CCR 80 <i       >          MDCR E7
 R0 0000  R1 0000  R2 0041  R3 0011  R4 1100  R5 FE82  R6 0020  R7 FF80









STATUS:   H8/338--Stepping complete_____________________________________...R....
 display registers

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

You can use "register class" and "register name" to display registers. Refer to the "Register Class and Name" section in Chapter 5.

When you enter the "**step**" command with registers displayed, the register display is updated every time you enter the command.

step <RETURN>, <RETURN>, <RETURN>

```
Registers

Next_PC 1022
 PC 1022  SP FD00  CCR 80 <i       >          MDCR E7
 R0 0000  R1 0000  R2 0041  R3 FE11  R4 2000  R5 FC02  R6 0020  R7 FE80

Step_PC 1022  BRA cmd_rds.src:Write_Msg
Next_PC 1032
 PC 1032  SP FD00  CCR 80 <i       >          MDCR E7
 R0 0000  R1 0000  R2 0041  R3 FE11  R4 2000  R5 FC02  R6 0020  R7 FF80

Step_PC 1032  MOV.W #FE82,R5
Next_PC 1036
 PC 1036  SP FF80  CCR 80 <i       >          MDCR E7
 R0 0000  R1 0000  R2 0041  R3 FE11  R4 2000  R5 FC02  R6 0020  R7 FF80



STATUS:   H8/338--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".

Displaying the Trace

The trace listings which follow are of program execution on the H8/338 emulator. To display the trace, enter:

display trace <RETURN>

```
Trace List                  Offset=0
Label:      Address       Data            Opcode or Status          time count
Base:       symbols       hex            mnemonic w/symbols          relative
after     :Cmd_Input        42FF    42    read  mem byte           ------------
+001   :cmd_rds.s:+0010     47F8  BEQ cmd_rds.src:Scan                200     nS
+002   cmd_rds.:Exe_Cmd     AA41  CMP.B #41,R2L                       200     nS
+003   cmd_rds.src:Scan     6A0A    6A0A  fetch mem                   200     nS
+004   :cmd_rds.s:+0014     4706  BEQ cmd_rds.sr:Cmd_A                200     nS
+005   :cmd_rds.s:+0016     AA42  CMP.B #42,R2L                       200     nS
+006   cmd_rds.sr:Cmd_A     FB11    FB11  fetch mem                   200     nS
+007   :cmd_rds.s:+0018     470A  BEQ cmd_rds.sr:Cmd_B                200     nS
+008   :cmd_rds.s:+001A     4010    4010  fetch mem                   200     nS
+009   cmd_rds.sr:Cmd_B     FB11  MOV.B #11,R3L                       200     nS
+010   :cmd_rds.s:+0026     7904  MOV.W #2011,R4                      200     nS
+011   :cmd_rds.s:+0028     2011    2011  fetch mem                   200     nS
+012   :cmd_rds.s:+002A     4006  BRA cmd_rd:Write_Msg                200     nS
+013   cmd_rds.sr:Cmd_I     FB0F    FB0F  fetch mem                   200     nS
+014   cmd_rd:Write_Msg     7905  MOV.W #FC02,R5                      200     nS

STATUS:   H8/338--Running user program    Emulation trace complete______........
 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.

```
Trace List                  Offset=0
Label:      Address       Data            Opcode or Status          time count
Base:       symbols       hex            mnemonic w/symbols          relative
+015   :cmd_rds.s:+0034     FE82    FE82  fetch mem                    200     nS
+016   cmd_rd:Clear_Old      FE20  MOV.B #20,R6L                      200     nS
+017   cmd_r:Clear_Loop      68D8  MOV.B R0L,@R5                      200     nS
+018   :cmd_rds.s:+003A      0B05  ADDS #1,R5                         200     nS
+019      :Msg_Dest          0000    00    write mem byte             200     nS
+020   :cmd_rds.s:+003C      1A0E  DEC R6L                            200     nS
+021   :cmd_rds.s:+003E      46F8  BNE cmd_r:Clear_Loop               200     nS
+022   :cmd_rds.s:+0040      7905    7905  fetch mem                  200     nS
+023   cmd_r:Clear_Loop      68D8  MOV.B R0L,@R5                      200     nS
+024   :cmd_rds.s:+003A      0B05  ADDS #1,R5                         200     nS
+025   :cmd_rds.s:+0003      0000    00    write mem byte             200     nS
+026   :cmd_rds.s:+003C      1A0E  DEC R6L                            200     nS
+027   :cmd_rds.s:+003E      46F8  BNE cmd_r:Clear_Loop               200     nS
+028   :cmd_rds.s:+0040      7905    7905  fetch mem                  200     nS
+029   cmd_r:Clear_Loop      68D8  MOV.B R0L,@R5                      200     nS

STATUS:   H8/338--Running user program    Emulation trace complete______........
 display trace


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

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.

Displaying Trace with Time Count Absolute

Enter the following command to display count information absolute from the trigger state.

display trace count absolute
<RETURN>

```
Trace List                  Offset=0
Label:      Address       Data            Opcode or Status          time count
Base:       symbols       hex           mnemonic w/symbols           absolute
after  cmd_rd:Cmd_Input     42FF    42    read  mem byte           ------------
+001           abs 2010     47F8  BEQ cmd_rds.src:Scan             + 200     nS
+002   cmd_rds.:Exe_Cmd     AA41  CMP.B #41,R2L                    + 400     nS
+003   cmd_rds.src:Scan     6A0A    6A0A  fetch mem                + 600     nS
+004           abs 2014     4706  BEQ cmd_rds.sr:Cmd_A             + 800     nS
+005           abs 2016     AA42  CMP.B #42,R2L                    +   1.0   nS
+006   cmd_rds.sr:Cmd_A     FB11    FB11  fetch mem                +   1.2   uS
+007           abs 2018     470A  BEQ cmd_rds.sr:Cmd_B             +   1.4   uS
+008           abs 201A     4010    4010  fetch mem                +   1.6   uS
+009   cmd_rds.sr:Cmd_B     FB11  MOV.B #11,R3L                    +   1.8   uS
+010           abs 2026     7904  MOV.W #1011,R4                   +   2.0   uS
+011           abs 2028     1011    1011  fetch mem                +   2.2   uS
+012           abs 202A     4006  BRA cmd_rd:Write_Msg             +   2.4   uS
+013   cmd_rds.sr:Cmd_I     FB0F    FB0F  fetch mem                +   2.6   uS
+014   cmd_rd:Write_Msg     7905  MOV.W #FE02,R5                   +   2.8   uS

STATUS:   H8/338--Running user program    Emulation trace complete______........


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

Changing the Trace Depth

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

display trace depth 512 <RETURN>

This command increases the number of states in the trace list to 512. If you use the <NEXT> key to page down through the trace, you can see where the program returns to the **Clear** instruction at state 349.

```
Trace List                  Offset=0
Label:      Address       Data            Opcode or Status          time count
Base:       symbols       hex           mnemonic w/symbols          absolute
+345   :cmd_rds.s:+004C     46F6  BNE cmd_r:Write_Loop             +  72.40  uS
+346   :cmd_rds.s:+004E     40B4  BRA cmd_rds.sr:Clear             +  72.60  uS
+347   cmd_r:Write_Loop     6C4E    6C4E  fetch mem                +  72.80  uS
+348            1050       F3FF    F3FF  fetch mem                +  73.00  uS
+349   cmd_rds.sr:Clear     F800  MOV.B #00,R0L                    +  73.20  uS
+350   :cmd_rds.s:+0006     6A88  MOV.B R0L,@:Cmd_Input            +  73.40  uS
+351   :cmd_rds.s:+0008     FC00    FC00  fetch mem                +  73.60  uS
+352   cmd_rds.src:Scan     6A0A  MOV.B @:Cmd_Input,R2L            +  73.80  uS
+353      :Cmd_Input        0000    00    write mem byte           +  74.00  uS
+354   :cmd_rds.s:+000C     FC00    FC00  fetch mem                +  74.20  uS
+355   :cmd_rds.s:+000E     AA00  CMP.B #00,R2L                    +  74.40  uS
+356      :Cmd_Input        00FF    00    read  mem byte           +  74.60  uS
+357   :cmd_rds.s:+0010     47F8  BEQ cmd_rds.src:Scan             +  74.80  uS
+358   cmd_rds.:Exe_Cmd     AA41    AA41  fetch mem                +  75.00  uS
+359   cmd_rds.src:Scan     6A0A  MOV.B @:Cmd_Input,R2L            +  75.20  uS

STATUS:   H8/338--Running user program    Emulation trace complete______........
 display trace


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

H8/338 Analysis Status Qualifiers

The status qualifier "read" was used in the example trace command used above. The following analysis status qualifiers may also be used with the H8/338 emulator.

Qualifier	Status Bits (32..44)	Description
backgrnd	0 xxxx xxxx xxxxB	Background cycle
byte	x 1xxx xxxx xx1xB	Byte access
foregrnd	1 xxxx xxxx xxxxB	Foreground cycle
grd	x 10xx xxxx xxxxB	Guarded memory access
ifetch	x 1xxx xxx0 1101B	Fetch from internal ROM
intack	x xxx0 xxxx xxxxB	Interrupt acknowledge cycle
io	x 1xxx xxxx 0xxxB	Internal I/O access
memory	x 1xxx xxxx 1xxxB	Memory access
read	x 1xxx xxxx xxx1B	Read cycle
word	x 1xxx xxxx xx0xB	Word access
write	x 1xxx xxxx xxx0B	Write cycle
wrrom	x 1x0x xxxx xxx0B	Write to ROM cycle

Trace Analysis Considerations

There are some points to be noticed when you use the emulation analyzer.

How to Specify Trigger Condition

You need to be careful to specify the condition on which the emulation analyzer should start the trace. Suppose that you would like to start the trace when the program begins executing **Exe_Cmd** routine:

```
trace after cmd_rds.src:Exe_Cmd
<RETURN>
modify memory Cmd_Input bytes to 41h
<RETURN>
```

You will see:

```
Trace List                Offset=0
Label:      Address      Data          Opcode or Status          time count
Base:       symbols      hex          mnemonic w/symbols          absolute
after  cmd_rds.:Exe_Cmd    AA41    AA41  fetch mem                ------------
+001   cmd_rds.src:Scan    6A0A  MOV.B @:Cmd_Input,R2L          + 200     nS
+002   :cmd_rds.s:+000C    FE80    FE80  fetch mem              + 400     nS
+003   :cmd_rds.s:+000E    AA00  CMP.B #00,R2L                  + 600     nS
+004      :Cmd_Input       00FF    00    read  mem byte         + 800     nS
+005   :cmd_rds.s:+0010    47F8  BEQ cmd_rds.src:Scan           +   1.0   uS
+006   cmd_rds.:Exe_Cmd    AA41    AA41  fetch mem              +   1.2   uS
+007   cmd_rds.src:Scan    6A0A  MOV.B @:Cmd_Input,R2L          +   1.4   uS
+008   :cmd_rds.s:+000C    FE80    FE80  fetch mem              +   1.6   uS
+009   :cmd_rds.s:+000E    AA00  CMP.B #00,R2L                  +   1.8   uS
+010      :Cmd_Input       00FF    00    read  mem byte         +   2.0   uS
+011   :cmd_rds.s:+0010    47F8  BEQ cmd_rds.src:Scan           +   2.2   uS
+012   cmd_rds.:Exe_Cmd    AA41    AA41  fetch mem              +   2.4   uS
+013   cmd_rds.src:Scan    6A0A  MOV.B @:Cmd_Input,R2L          +   2.6   uS
+014   :cmd_rds.s:+000C    FE80    FE80  fetch mem              +   2.8   uS

STATUS:   H8/338--Running user program    Emulation trace complete______........
 trace after Exe_Cmd


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

This is not what we were expecting to see. As you can see at the first line of the trace list, the address of **Exe_Cmd** routine appears on the address bus during the program executing **Scan** loop. This made the emulation analyzer start trace. To avoid mis-trigger by this cause, set the trigger condition to the second instruction of the routine you want to trace:

trace after cmd_rds.src:Exe_Cmd+2 <RETURN>

(Since the instruction at Exe_Cmd label is two bytes instruction, the next instruction starts from **Exe_Cmd+2**.)

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

```
Trace List                Offset=0
Label:      Address      Data          Opcode or Status          time count
Base:       symbols      hex          mnemonic w/symbols          absolute
after  :cmd_rds.s:+0014    4706  BEQ cmd_rds.sr:Cmd_A             ------------
+001   :cmd_rds.s:+0016    AA42    AA42  fetch mem                + 200     nS
+002   cmd_rds.sr:Cmd_A    FB11  MOV.B #11,R3L                    + 400     nS
+003   :cmd_rds.s:+001E    7904  MOV.W #2000,R4                   + 600     nS
+004   :cmd_rds.s:+0020    2000    2000  fetch mem                + 800     nS
+005   :cmd_rds.s:+0022    400E  BRA cmd_rd:Write_Msg             +   1.0   uS
+006   cmd_rds.sr:Cmd_B    FB11    FB11  fetch mem                +   1.2   uS
+007   cmd_rd:Write_Msg    7905  MOV.W #FE82,R5                   +   1.4   uS
+008   :cmd_rds.s:+0034    FC02    FC02  fetch mem                +   1.6   uS
+009   cmd_rd:Clear_Old    FE20  MOV.B #20,R6L                    +   1.8   uS
+010   cmd_r:Clear_Loop    68D8  MOV.B R0L,@R5                    +   2.0   uS
+011   :cmd_rds.s:+003A    0B05  ADDS #1,R5                       +   2.2   uS
+012      :Msg_Dest        0000    00    write mem byte           +   2.4   uS
+013   :cmd_rds.s:+003C    1A0E  DEC R6L                          +   2.6   uS
+014   :cmd_rds.s:+003E    46F8  BNE cmd_r:Clear_Loop             +   2.8   uS

STATUS:   H8/338--Running user program    Emulation trace complete______........
 modify memory Cmd_Input byte to 41h


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

If you need to see the execution of the instruction at **Exe_Cmd** label, use **trace about** command instead of **trace after** command. When you use the **trace about** command, the state which triggered the analyzer will appear in the center of the trace list.

Store Condition and Trace

When you specify store condition with **trace only** command, disassembling of program execution is unreliable.

trace <RETURN>

```
Trace List                  Offset=0
Label:      Address       Data            Opcode or Status          time count
Base:       symbols       hex           mnemonic w/symbols          absolute
after  cmd_rds.:Exe_Cmd     AA41    AA41  fetch mem                 ------------
+001   cmd_rds.src:Scan     6A0A  MOV.B @:Cmd_Input,R2L             + 200     nS
+002   :cmd_rds.s:+000C     FC00    FC00  fetch mem                 + 400     nS
+003   :cmd_rds.s:+000E     AA00  CMP.B #00,R2L                     + 600     nS
+004      :Cmd_Input        00FF    00    read  mem byte            + 800     nS
+005   :cmd_rds.s:+0010     47F8  BEQ cmd_rds.src:Scan              +   1.0   uS
+006   cmd_rds.:Exe_Cmd     AA41    AA41  fetch mem                 +   1.2   uS
+007   cmd_rds.src:Scan     6A0A  MOV.B @:Cmd_Input,R2L             +   1.4   uS
+008   :cmd_rds.s:+000C     FC00    FC00  fetch mem                 +   1.6   uS
+009   :cmd_rds.s:+000E     AA00  CMP.B #00,R2L                     +   1.8   uS
+010      :Cmd_Input        00FF    00    read  mem byte            +   2.0   uS
+011   :cmd_rds.s:+0010     47F8  BEQ cmd_rds.src:Scan              +   2.2   uS
+012   cmd_rds.:Exe_Cmd     AA41    AA41  fetch mem                 +   2.4   uS
+013   cmd_rds.src:Scan     6A0A  MOV.B @:Cmd_Input,R2L             +   2.6   uS
+014   :cmd_rds.s:+000C     FC00    FC00  fetch mem                 +   2.8   uS

STATUS:   H8/338--Running user program    Emulation trace complete______........
 trace after Exe_Cmd


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

The program is executing the **Scan** loop.

Now, trace only accesses to the address range **Init** through **Init+0ffh**.

trace only range Init ***thru*** Init+0ffh <RETURN>

```
Trace List                  Offset=0
Label:      Address       Data            Opcode or Status          time count
Base:       symbols       hex           mnemonic w/symbols          absolute
after  cmd_rds.:Exe_Cmd     AA41    AA41  fetch mem                 ------------
+001   cmd_rds.src:Scan     6A0A  MOV.B @:Cmd_Input,R2L             + 240     nS
+002   :cmd_rds.s:+000C     FE80    FE80  fetch mem                 + 400     nS
+003   :cmd_rds.s:+000E     AA00    AA00  fetch mem                 + 600     nS
+004   :cmd_rds.s:+0010     47F8  BEQ cmd_rds.src:Scan              +   1.0   uS
+005   cmd_rds.:Exe_Cmd     AA41    AA41  fetch mem                 +   1.2   uS
+006   cmd_rds.src:Scan     6A0A  MOV.B @:Cmd_Input,R2L             +   1.4   uS
+007   :cmd_rds.s:+000C     FE80    FE80  fetch mem                 +   1.6   uS
+008   :cmd_rds.s:+000E     AA00    AA00  fetch mem                 +   1.8   uS
+009   :cmd_rds.s:+0010     47F8  BEQ cmd_rds.src:Scan              +   2.2   uS
+010   cmd_rds.:Exe_Cmd     AA41    AA41  fetch mem                 +   2.4   uS
+011   cmd_rds.src:Scan     6A0A  MOV.B @:Cmd_Input,R2L             +   2.6   uS
+012   :cmd_rds.s:+000C     FE80    FE80  fetch mem                 +   2.8   uS
+013   :cmd_rds.s:+000E     AA00    AA00  fetch mem                 +   3.0   uS
+014   :cmd_rds.s:+0010     47F8  BEQ cmd_rds.src:Scan              +   3.4   uS

STATUS:   H8/338--Running user program    Emulation trace complete______........
 trace only range Init thru Init+0ffh


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

As you can see the execution of CMP.B instructions are not disassembled. This occurs when the analyzer cannot get necessary information for disassembling because of the store condition. Be careful when you use the **trace only** command.

Triggering the Analyzer by Data

You may want to trigger the emulation analyzer when specific data appears on the data bus. You can accomplish this with the following command.

trace after data **<data>** <RETURN>

There are some points to be noticed when you trigger the analyzer in this way. You always need to specify the <data> with 16 bits value even when access to the data is performed by byte access. This is because the analyzer is designed so that it can capture data on internal data bus (which has 16 bits width). The following table shows the way to specify the trigger condition by data.

Location of data	Access	Available <data> Specification
Internal ROM,RAM	word access	hhll *1
		hhxx *2
		xxll *2
	byte access	ddxx *2
Others	byte access *3	ddxx

*1 hhll means 16 bits data
*2 dd,hh,ll mean 8 bits data
*3 H8/338 processor performs word access (MOV.W etc..) to external memory and internal I/O by two byte accesses.

For example, to trigger the analyzer when the processor performs word access to data 1234 hex in internal ROM, you can do any of the following:

trace after data 1234h <RETURN>
trace after data 12xxh <RETURN>
trace after data 0xx34h <RETURN>

To trigger the analyzer when the processor accesses data 12 hex in external ROM:

```
trace after data 12xxh <RETURN>
```

Notice that you always need to specify "xx" as the lower 8 bits value to capture byte access of the processor. Be careful to trigger the analyzer by data.

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

Using the H8/338 Emulator In-Circuit

When you are ready to use the H8/338 Emulator in conjunction with actual target system hardware, there are some special considerations you should keep in mind.

- installing the emulator probe
- properly configure the emulator

We will cover the first topic in this chapter. For complete details on in-circuit emulation configuration, refer to Chapter 4.

Installing the Target System Probe

Caution

The following precautions should be taken while using the H8/338 Emulator. Damage to the emulator circuitry may result if these precautions are not observed.

Power Down Target System. Turn off power to the user target system and to the H8/338 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 H8/338 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 H8/338 Emulator; when powering down, turn off the emulator first, then turn off power to the target system.

Pin Guard

The HP 64793 emulator is shipped with a pin guard to prevent impact damage to the target system probe pins. The guard should be left in place while you are not using the emulator.

H8/338 Emulator

HP 64793A H8/338 emulator is shipped with a non-conductive pin guard over the target system probe.

H8/329 Emulator

HP 64793B H8/329 emulator is shipped with a conductive plastic pin guard over the target system probe pins. When you **do** use the emulator, either for normal emulation tasks, or to run performance verification on the emulator, you must remove this conductive pin guard to avoid intermittent failures due to the target system probe lines being shorted together.

Pin Protector (H8/329 Only)

The target system probe of the H8/329 emulator has a pin protector that prevents damage to the probe when inserting and removing the probe from the target system microprocessor socket. **Do not** use the probe without a pin protector installed. If the target system probe is installed on a densely populated circuit board, there may not be enough room for the plastic shoulders of the probe socket. If this occurs, another pin protector may be stacked onto the existing pin protector.

Installing the Target System Probe

1. Remove the H8/338 microprocessor from the target system 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.
4.

Note

When you are using the H8/338 emulator, we recommend that you use **ITT CANNON** "**LCS-84**" series 84 pin PLCC socket to make sure the contact between emulator probe and target system microprocessor socket.

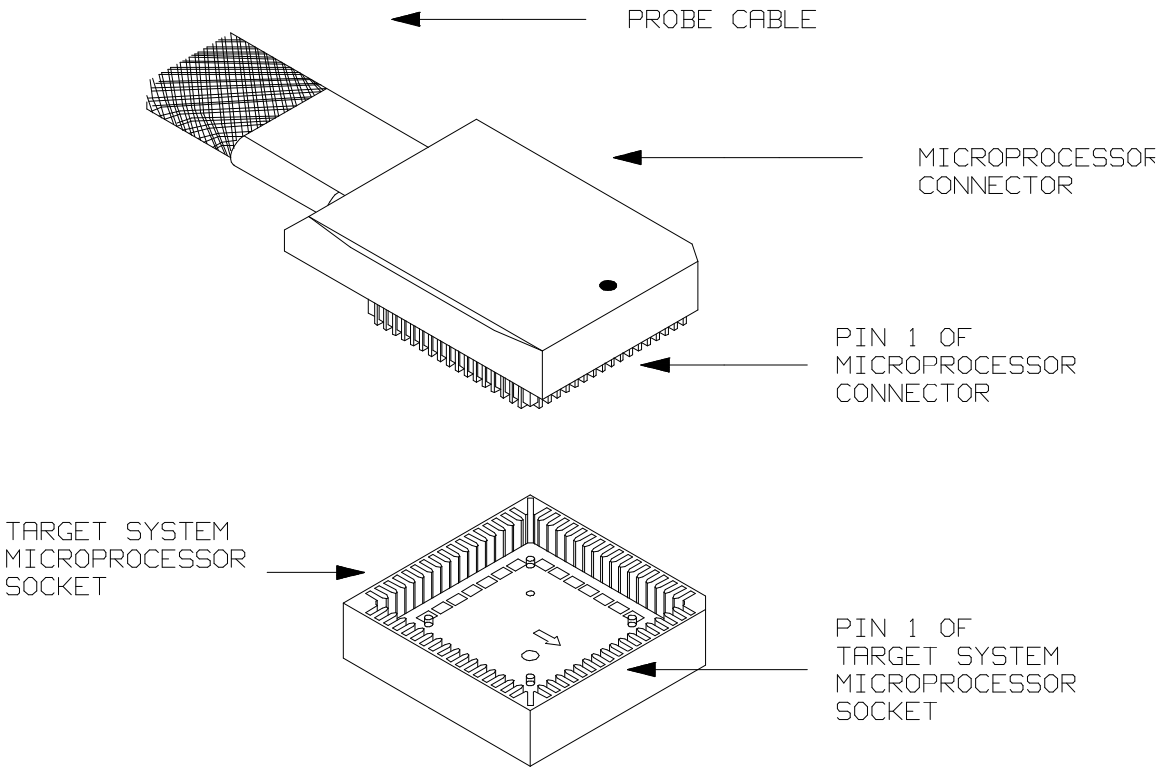


Figure 3-1. Installing the Probe (H8/338 emulator)

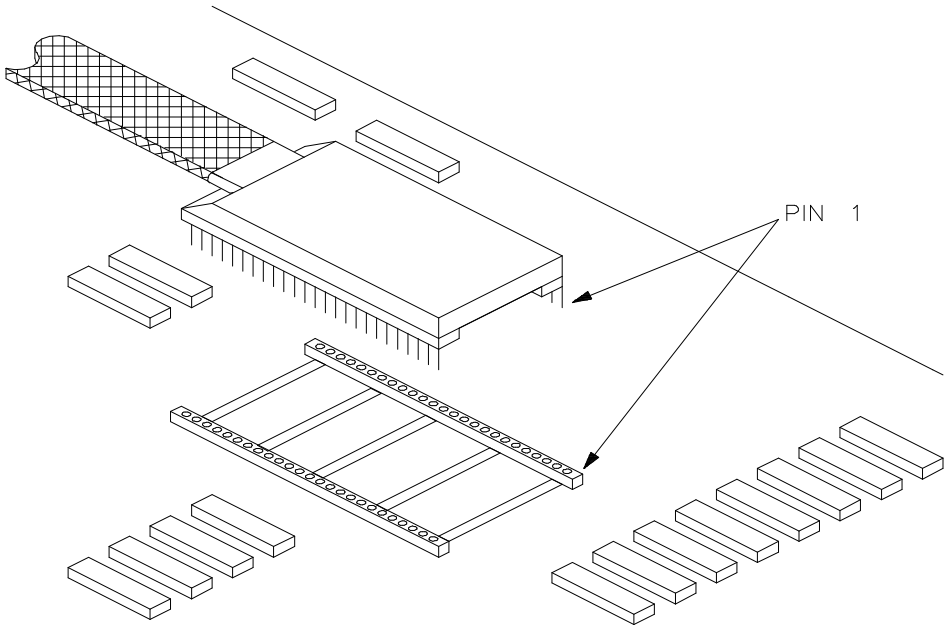


Figure 3-2. Installing the Probe (H8/329 emulator)

Pin State in Background

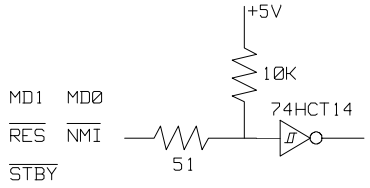
While the emulator is running the background monitor, probe pins are in the following state.

Address Bus	Same as foreground
Data Bus	Always high impedance otherwise you direct the emulator to modify target memory.
$\overline{\text{AS}}$	Same as foreground
$\overline{\text{RD}}$	Same as foreground
$\overline{\text{WR}}$	Always high otherwise you direct the emulator to modify target memory.
Others	Same as foreground

Target System Interface (H8/338)

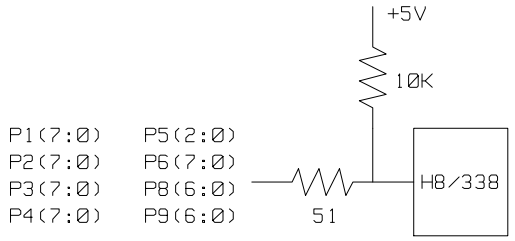
MD1 MD0
RES NMI
STBY

These signals are connected to 74HCT14 through 51 ohm series resistor and 10K ohm pull-up resistor.



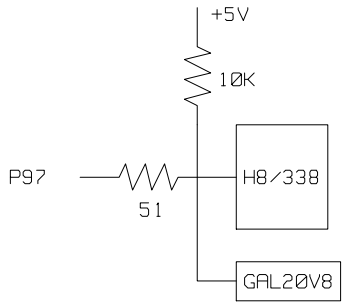
P1(7:0) P5(2:0)
P2(7:0) P6(7:0)
P3(7:0) P8(6:0)
P4(7:0) P9(6:0)

These signals are connected to H8/338 emulation processor through 51 ohm series resistor and 10K ohm pull-up resistor.



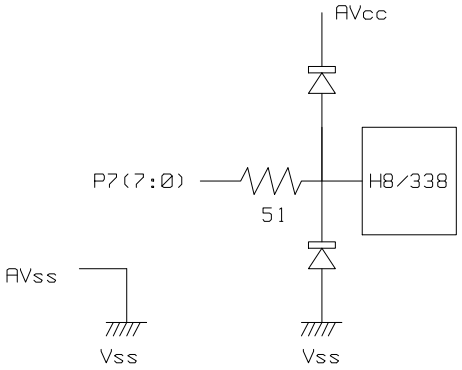
P97

This signal are connected to H8/338 emulation processor and GAL20V8 through 51 ohm series resistor.



P7(7:0)

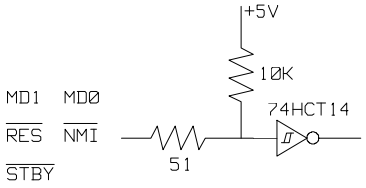
These signals are connected to H8/338 emulation processor through 51 ohm series resister



Target System Interface (H8/329)

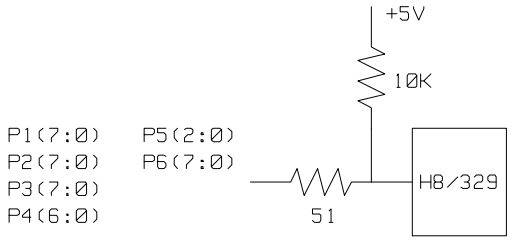
MD1 MD0
RES NMI
STBY

These signals are connected to 74HCT14 through 51 ohm series resistor and 10K ohm pull-up resistor.



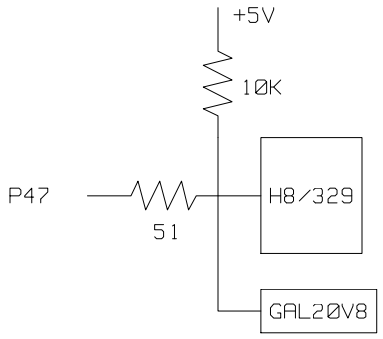
P1(7:0) P4(6:0)
P2(7:0) P5(2:0)
P3(7:0) P6(7:0)

These signals are connected to H8/329 emulation processor through 51 ohm series resistor and 10K ohm pull-up resistor.



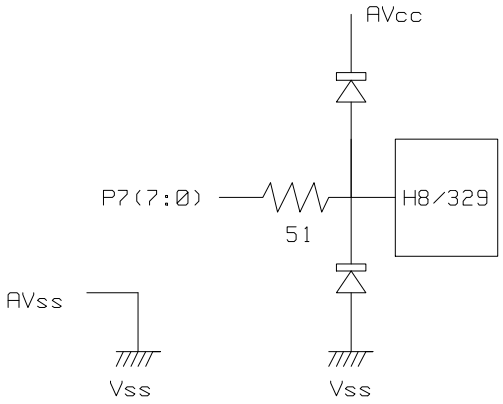
P47

This signal are connected to H8/329 emulation processor and GAL20V8 through 51 ohm series resistor.



P7(7:0)

These signals are connected to H8/329 emulation processor through 51 ohm series resister



In-Circuit Configuration Options

The H8/338 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.

Enabling/Disabling /NMI input from the target system

You can configure the emulator to accept the /NMI input from the target system.

Enabling/Disabling /RES input from the target system

You can configure the emulator to accept the /RES input from the target system.

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. 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.

4

Configuring the Emulator

Introduction

Your H8/338 emulator can be used in all stages of target system development. For instance, you can run the emulator out-of-circuit when developing your target system software, or you can use the emulator in-circuit when integrating software with target system hardware. You can use the emulator's internal clock or the target system clock. Emulation memory can be used in place of, or along with, target system memory. 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 may be configured to suit your needs at any stage of the development process. This chapter describes the options available when configuring the HP 64793 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:

- Mapping memory.

Emulator Pod Configuration:

- Selecting the microprocessor to be emulated.
- Selecting the processor operation mode.
- Enabling /NMI input from the target system.
- Enabling /RES input from the target system.
- Setting up 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 H8/338 microprocessor. The maximum clock speed is 10 MHz (system clock).

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?

The "restrict to real-time" question lets you configure the emulator so that commands which cause the emulator to break to monitor and return to the user program are refused.

no All commands, regardless of whether or not they require a break to the emulation monitor, are accepted by the emulator.

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

- Display/modify registers.
- Display/modify internal I/O registers.
- Display/modify target system memory.
- Load/store target system memory.

Caution

If your target system circuitry is dependent on constant execution of program code, you should restrict the emulator to real-time runs. This will help insure that target system damage does not occur. However, remember that you can still execute the "reset", "break", and "step" commands; you should use caution in executing these commands.

Memory Configuration

The memory configuration questions allow 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?

Mapping memory

The emulation memory consists of 64K bytes, mappable in 128 byte blocks. You can define up to 16 different map terms. The emulation memory system does not introduce wait states.

The memory mapper allows you to characterize memory locations. It allows you to specify whether a certain range of memory is present in the target system or whether you will be using the 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.

The default memory mapping is shown below.

	emulation ROM (erom)	**emulation RAM (eram)**
H8/338 emulator	0 - BFFF	F780 - FF7F
H8/329 emulator	0 - 7FFF	FB80 - FF7F

Caution

The default emulator configuration maps location 0 hex through 3fff hex as emulation ROM and location fd80 hex through ff7f hex as emulation RAM. This must be needed when you use the H8/338 internal ROM and RAM.

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).

For example, you might be developing a system with the following characteristics:

- input port at 0f00 hex
- output port at 0f100 hex
- program and data from 1000 through 2fff hex

Suppose that the only thing that exists in your target system at this time are input and output ports and some control logic; no memory is available. you can reflect this by mapping the I/O ports to target system memory space and the rest of memory to emulation memory space:

```
delete all <RETURN>
1000h thru 2fffh emulation rom
<RETURN>
0f000h thru 0f1ffh emulation ram
<RETURN>
end <RETURN>
```

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 microprocessor to be emulated.

H8/338 Emulator

336 When you are going to emulate H8/336 microprocessor, select this item.

337 When you are going to emulate H8/337 microprocessor, select this item.

338 When you are going to emulate H8/338 microprocessor, select this item.

H8/329 Emulator

326 When you are going to emulate H8/326 microprocessor, select this item.

327 When you are going to emulate H8/327 microprocessor, select this item.

328 When you are going to emulate H8/328 microprocessor, select this item.

329 When you are going to emulate H8/329 microprocessor, select this item.

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 and MD1. If you are using the emulator out of circuit when "external" is selected, the emulator will operate in mode 3.

When mode_1 through mode_3 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. (Internal ROM is disabled)
mode_2	The emulator will operate in mode 2. (Internal ROM is enabled)
mode_3	The emulator will operate in mode 3. (Single Chip Mode)

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.

The emulator does not accept any interrupt while in background monitor. Edge sensed interrupts are suspended while running the background monitor, and such interrupts will occur when context is changed to foreground. Level sensed interrupts and internal interrupts are ignored during the background operation.

Enable /RES input from the target system?

This configuration allows you to specify whether or not the emulator responds to /RES signal from the target system during foreground operation.

While running the background monitor, the emulator ignores /RES signal, otherwise the emulator 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 input during foreground operation.

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

Reset value for stack pointer?

This question allows you to specify the value to which the stack pointer (SP) 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 16-bit hexadecimal even address outside internal I/O register area.

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.

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 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>
```

Limitations, Restrictions

Foreground Monitor

Foreground monitor is not supported for the H8/338 emulator.

Sleep and Software Stand-by Mode

When the emulator breaks into the monitor, the H8/338 sleep or software stand-by mode is released and comes to normal processor mode. If you use the "display register" command (see the "Displaying Registers" section of the "Getting started" chapter) at the sleep or software stand-by mode, the emulation processor mode will come to normal mode and the emulator will fetch next program.

Store Condition and Trace

Disassembling of program execution is unreliable when the emulation analyzer is used with store condition (that is**, trace only** command). Refer to chapter 2 of this manual.

Step Command and Interrupts

Step execution cannot be performed in the following cases.

- When the emulator is in the monitor and a suspended interrupt is existed.
- When the emulator is in the monitor and a level sensed interrupt is existed (including interrupts from internal I/O device).

In the above cases, you will see an error message when you attempt to execute the **step** command.

The contents of registers will be the same as those before the issue of the **step** command.

RAM Enable Bit

The internal RAM of H8/338 processor can be enabled/disabled by RAME (RAM enable bit). However, once you map the internal RAM area to emulation RAM, the emulator still accesses emulation RAM even if the internal RAM is disabled by RAME.

Software Performance Analysis

Program activity measurement using the Software Performance Measurement Tool (SPMT) is valid only for H8/338 internal ROM area. Outside this area, the result of program activity measurement is not reliable.

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 Classes and Names (H8/338 Emulator)

The following register classes and names are used with the display/modify registers commands in H8/338 emulator.

BASIC (*) class

Register name	Description
PC	Program counter
CCR	Condition code register
R0	Register 0
R1	Register 1
R2	Register 2
R3	Register 3
R4	Register 4
R5	Register 5
R6	Register 6
R7	Register 7
SP	Stack pointer
MDCR	Mode control register

SYS class (System control registers)

Register name	Description
STCR	Serial timer control register
SYSCR	System control register
MDCR	Mode control register
ISCR	IRQ sense control register
IER	IRQ enable register

PORT class (I/O port)

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
P8DDR	Port 8 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
P1PCT	Port 1 input pull up MOS control register
P2PCT	Port 2 input pull up MOS control register
P3PCT	Port 3 input pull up MOS control register

FRT class (16 bit free running timer)

Register name	Description
TIER	Timer interrupt enable register
FRTCSR	Timer control/status register
FRC	Free running counter
OCRA	Output compare register A
OCRB	Output compare register B
FRTCR	Timer control register
TOCR	Timer output compare control register
ICRA	Input capture register A
ICRB	Input capture register B
ICRC	Input capture register C
ICRD	Input capture register D

TMR0 class (8 bit timer 0)

Register name	Description
TCR0	Timer control register
TCSR0	Timer control/status register
TCORA0	Timer constant register A
TCORB0	Timer constant register B
TCNT0	Timer counter

TMR1 class (8 bit timer 1)

Regsiter name	Description
TCR1	Timer control register
TCSR1	Timer control/status register
TCORA1	Timer constant register A
TCORB1	Timer constant register B
TCNT1	Timer counter

PWM0 class (PWM timer 0)

Register name	Description
PWMTCR0	Timer control register
DTR0	Duty register
PWMTCNT0	Timer counter

PWM1 class (PWM timer 1)

Regsiter name	Description
PWMTCR1	Timer control register
DTR1	Duty register
PWMTCNT1	Timer counter

SCI0 class (Serial communication interface 0)

Register name	Description
SMR0	Serial mode register
BRR0	Bit rate register
SCR0	Serial control register
TDR0	Transmit data register
SSR0	Serial status register
RDR0	Receive data register

SCI1 class (Serial communication interface 1)

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

ADC class (A/D converter)

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

DAC class (D/A converter)

Register name	Description
DADR0	D/A data register 0
DADR1	D/A data register 1
DACR	D/A control register

No class The following register names are not included in any register class.

Register name	Description
R0H	Register 0 H
R0L	Register 0 L
R1H	Register 1 H
R1L	Register 1 L
R2H	Register 2 H
R2L	Register 2 L
R3H	Register 3 H
R3L	Register 3 L
R4H	Register 4 H
R4L	Register 4 L
R5H	Register 5 H
R5L	Register 5 L
R6H	Register 6 H
R6L	Register 6 L
R7H	Register 7 H
R7L	Register 7 L

Register Classes and Names (H8/329 Emulator)

The following register classes and names are used with the display/modify registers commands in H8/329 emulator.

BASIC (*) class

Register name	Description
PC	Program counter
CCR	Condition code register
R0	Register 0
R1	Register 1
R2	Register 2
R3	Register 3
R4	Register 4
R5	Register 5
R6	Register 6
R7	Register 7
SP	Stack pointer
MDCR	Mode control register

SYS class (System control)

Register name	Description
STCR	Serial timer control register
SYSCR	System control register
MDCR	Mode control register
ISCR	IRQ sense control register
IER	IRQ enable register

Port class (I/O port)

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
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
P1PCR	Port 1 input pull up MOS control register
P2PCR	Port 2 input pull up MOS control register
P3PCR	Port 3 input pull up MOS control register

FRT class (16 bit free running timer)

Register name	Description
FRTCR	Timer control register
FRTCSR	Timer control/status register
FRC	Free running counter
OCRA	Output compare register A
OCRB	Output compare register B
ICR	Input capture register
FNCR	FRT noise-canceler control register

TMR0 class (8 bit timer 0)

Register name	Description
TCR0	Timer control register
TCSR0	Timer control/status register
TCORA0	Timer constant register A
TCORB0	Timer constant register B
TCNT0	Timer counter

TMR1 class (8 bit timer 1)

Register name	Description
TCR1	Timer control register
TCSR1	Timer control/status register
TCORA1	Timer constant register A
TCORB1	Timer constant register B
TCNT1	Timer counter

SCI class (Serial communication interface)

Register name	Description
SMR	Serial mode register
BRR	Bit rate register
SCR	Serial control register
TDR	Transmit data register
SSR	Serial status register
RDR	Receive data register

ADC class (A/D converter)

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

No Class The following register names are not included in any register class.

Register name	Description
R0H	Register 0 H
R0L	Register 0 L
R1H	Register 1 H
R1L	Register 1 L
R2H	Register 2 H
R2L	Register 2 L
R3H	Register 3 H
R3L	Register 3 L
R4H	Register 4 H
R4L	Register 4 L
R5H	Register 5 H
R5L	Register 5 L
R6H	Register 6 H
R6L	Register 6 L
R7H	Register 7 H
R7L	Register 7 L

Index

Notes