HP 64793

H8/338/329 Emulator Terminal Interface

User's Guide

HP Part No. 64793-97000
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.

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 64793-97000, October 1992

Using This Manual

This manual will show you how to use the following emulators with the firmware resident Terminal 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 Microprocessors	Referred to as
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	H8/337
	HD6433278P	H8/327
	HD6413278P	H8/327
	HD6433268P	H8/326

For the most part, the H8/338 and H8/329 emulators all operate the same way. Differences between the emulators are described where they exist. Both the H8/338 and H8/329 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 will:

- give you an introduction to using the emulator
- explore various ways of applying the emulator to accomplish your tasks
- show you emulator commands which are specific to the H8/338 Emulator

This manual will not:

- tell you how to use every emulator command; this is done in the *Terminal Interface Reference.*

Organization

Chapter 1 An introduction to the H8/338 emulator features and how they can help you in developing new hardware and software.

Chapter 2 A brief introduction to using the H8/338 Emulator. You will load and execute a short program, and make some measurements using the emulation analyzer.

Chapter 3 How to plug the emulator probe into a target system.

Chapter 4 Configuring the emulator to adapt it to your specific measurement needs.

Appendix A H8/338 Emulator Specific Command Syntax

Contents

Illustrations

Notes

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.

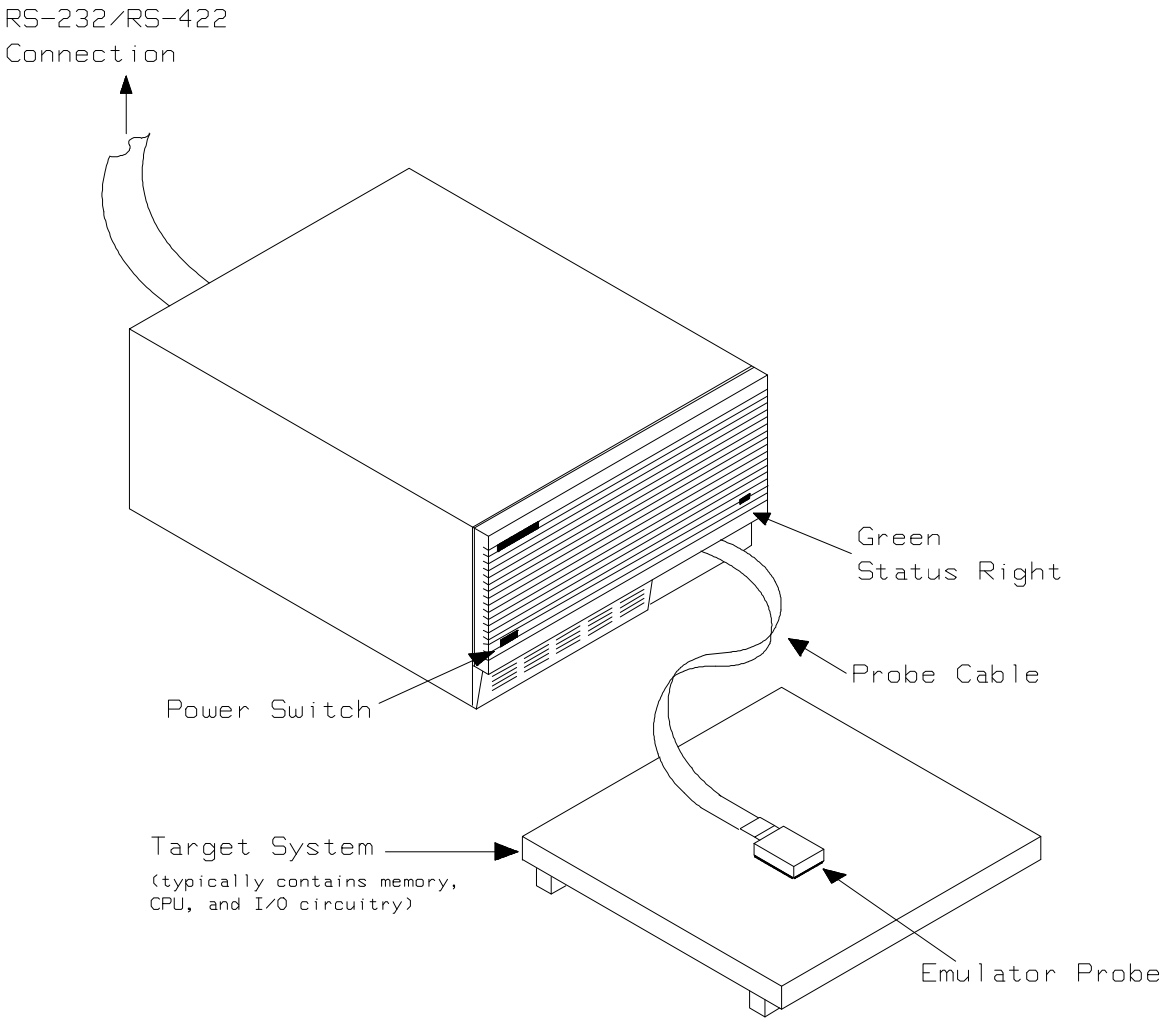


Figure 1-1. HP 64793 Emulator for the H8/338 Processor

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 microprocessors 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 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 can configure the emulator to break to the emulation monitor upon performing a write to ROM (which will stop a runaway program).

Analysis

The 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 HP 64703 64-channel Emulation Bus Analyzer and 16-channel State/Timing Analyzer allows you to probe up to 16 different lines in your target system.

Registers

You can display or modify the H8/338 internal register contents. This includes the ability to modify the program counter (PC) value so you can control where the emulator starts a program run.

Breakpoints

You can set the emulator/analyzer interaction so the emulator will break to the monitor program when the analyzer finds a specific state or states, allowing you to perform post-mortem analysis of the program execution. You can also set software breakpoints in your program using the **bp** command. This feature is realized by inserting a special instruction into user program. One of undefined opcodes (5770 hex) is used as software breakpoint instruction. Refer to the "Using Software Breakpoints" section of "Getting Started" chapter for more information.

Reset Support

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

Real Time Operation

Real-time signifies continuous execution of your program at full rated processor speed without interference from the emulator. (Such interference occurs when the emulator needs to break to the monitor to perform an action you requested, such as displaying target system memory.) Emulator features performed in real time include: running and analyzer tracing. Emulator features not performed in real time include: display or modify of target system memory, load/dump of target memory, display or modification of 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 analyzer is used with store condition (specified by **tsto** command). Refer to the "Trace Analysis Considerations" section in Chapter 2.

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 the "Restrictions and Considerations" section in Chapter 4.

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.

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. When you have completed this chapter, you will be able to perform these tasks:

- Set up an emulation configuration for out of circuit emulation use
- Map memory
- Transfer a small program into emulation memory
- Use run/stop controls to control operation of your program
- Use memory manipulation features to alter the program's operation
- Use analyzer commands to view the real time execution of your program
- Use software breakpoint feature to stop program execution at specific address
- Search memory for strings or numeric expressions
- Make program coverage measurements

Before You Begin

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

1. Completed hardware installation of the HP 64700 emulator in the configuration you intend to use for your work:
   - Standalone configuration
   - Transparent configuration
   - Remote configuration
   - Local Area Network configuration

   References: *HP 64700 Series Installation/Service* manual

2. If you are using the Remote Configuration, you must have completed installation and configuration of a terminal emulator program which will allow your host to act as a terminal connected to the emulator. In addition, you must start the terminal emulator program before you can work the examples in this chapter.

3. If you have properly completed steps 1 and 2 above, you should be able to hit <RETURN> (or <ENTER> on some keyboards) and get one of the following command prompts on your terminal screen:

```
U>
R>
M>
```

   If you do not see one of these command prompts, retrace your steps through the hardware and software installation procedures outlined in the manuals above, verifying all connections and procedural steps.

   In any case, you **must** have a command prompt on your terminal screen before proceeding with the tutorial.

A Look at the Sample Program

The sample program "COMMAND_READER" used in this chapter is shown in figure 2-1. The program emulates a primitive command interpreter.

Data Declarations

MESSAGE_A, MESSAGE_B and INVALID_INPUT define the messages used by the program to respond to various command inputs.

Initialization

The locations of the input area is moved into R1 for use by the program. Next, the CLEAR routine clears the command byte (the first byte location pointed to by the input area address - fe80 hex). R0L contains 00 hex for later use.

READ_INPUT

This routine continuously reads the byte at location fe80 hex until it is something other than a null character (00 hex); when this occurs, the PROCESS_COMM routine is executed.

```
1100                                    .SECTION        SAMPDATA,DATA,LOCATE=H'1100
1100 54484953204953204D45 MESSAGE_A     .SDATA          "THIS IS MESSAGE A"
110A 53534147452041
1111 54484953204953204D45 MESSAGE_B     .SDATA          "THIS IS MESSAGE B"
111B 53534147452042
1122 494E56414C494420434F INVALID_INPUT         .SDATA          "INVALID COMMAND"
112C 4D4D414E44

1000                                    .SECTION        SAMPPROG,CODE,LOCATE=H'1000
     0000FE80             INPUT_POINTER         .EQU            H'FE80
     0000FF00             OUTPUT_POINTER        .EQU            H'FF00

1000 7907FF80             INIT          MOV.W           #STACK,R7
1004 7901FE80                           MOV.W           #INPUT_POINTER,R1

1008 F800                 CLEAR                 MOV.B           #H'00,R0L
100A 6898                               MOV.B           R0L,@R1

100C 681A                 READ_INPUT    MOV.B           @R1,R2L
100E AA00                               CMP.B           #H'00,R2L
1010 47FA                               BEQ             READ_INPUT

1012 AA41                 PROCESS_COMM  CMP.B           #H'41,R2L
1014 4706                               BEQ             COMMAND_A
1016 AA42                               CMP.B           #H'42,R2L
1018 470A                               BEQ             COMMAND_B
101A 4010                               BRA             UNRECOGNIZED

101C FB11                 COMMAND_A     MOV.B           #H'11,R3L
101E 79041100                           MOV.W           #MESSAGE_A,R4
1022 400E                               BRA             OUTPUT

1024 FB11                 COMMAND_B     MOV.B           #H'11,R3L
1026 79041111                           MOV.W           #MESSAGE_B,R4
102A 4006                               BRA             OUTPUT

102C FB0F                 UNRECOGNIZED  MOV.B           #H'0F,R3L
102E 79041122                           MOV.W           #INVALID_INPUT,R4

1032 7905FF00             OUTPUT                MOV.W           #OUTPUT_POINTER,R5
1036 FE20                 CLEAR_OLD     MOV.B           #H'20,R6L
1038 68D8                 CLEAR_LOOP    MOV.B           R0L,@R5
103A 0B05                               ADDS.W          #1,R5
103C 1A0E                               DEC.B           R6L
103E 46F8                               BNE             CLEAR_LOOP

1040 7905FF00                           MOV.W           #OUTPUT_POINTER,R5
1044 6C4E                 OUTPUT_LOOP   MOV.B           @R4+,R6L
1046 68DE                               MOV.B           R6L,@R5
1048 0B05                               ADDS.W          #1,R5
104A 1A0B                               DEC.B           R3L
104C 46F6                               BNE             OUTPUT_LOOP
104E 40B8                               BRA             CLEAR

FF40                                    .SECTION        STACKAREA,STACK,LOCATE=H'FF40
FF40 0040                               .RES.B          64
FF80                      STACK

                                        .END
```

Figure 2-1. Sample Program Listing

2-4 Getting Started

PROCESS_COMM

Compares the input byte (now something other than a null) to the possible command bytes of "A" (ASCII 41 hex) and "B" (ASCII 42 hex), then jumps to the appropriate set up routine for the command message. If the input byte does not match either of these values, a branch to a set up routine for an error message is executed.

COMMAND_A, COMMAND_B, UNRECOGNIZED

These routines set up the proper parameters for writing the output message: the number of bytes in the message is moved to the R3L and the base address of the message in the data area is moved to R4.

OUTPUT

First the base address of the output area is moved to R5. Then the CLEAR_OLD routine writes nulls to 32 bytes of the output area (this serves both to initialize the area and to clear old messages written during previous program passes).

Finally, the proper message is written to the output area by the OUTPUT_LOOP routine. When done, OUTPUT_LOOP jumps back to CLEAR and the command monitoring process begins again.

Using the various features of the emulator, we will show you how to load this program into emulation memory, execute it, monitor the program's operation with the analyzer, and simulate entry of different commands utilizing the memory access commands provided by the HP 64700 command set.

Using the Help Facility

If you need a quick reference to the Terminal Interface syntax, you can use the built-in **help** facilities. For example, to display the top level **help** menu, type:

```
R> help
```

```
help  - display help information

  help <group>          - print help for desired group
  help -s <group>       - print short help for desired group
  help <command>        - print help for desired command
  help                  - print this help screen

--- VALID group NAMES ---
  gram     - system grammar
  proc     - processor specific grammar

  sys      - system commands
  emul     - emulation commands
  trc      - analyzer trace commands
  *        - all command groups
```

You can type the **?** symbol instead of typing **help**. For example, if you want a list of commands in the **emul** command group, type:

```
R> ? emul
```

```
emul - emulation commands
------------------------------------------------------------------------------
b......break to monitor  cp.....copy memory       mo.....modes
bc.....break condition   dump...dump memory       r......run user code
bp.....breakpoints       es.....emulation status  reg....registers
cf.....configuration     io.....input/output      rst....reset
cim....copy target image load...load memory       rx.....run at CMB execute
cmb....CMB interaction   m......memory            s......step
cov....coverage          map....memory mapper     ser....search memory
```

To display help information for any command, just type **help** (or **?**) and the command name. For example:

```
R> help load
```

```
load - download absolute file into processor memory space

  load -i         - download intel hex format
  load -m         - download motorola S-record format
  load -t         - download extended tek hex format
  load -S         - download symbol file
  load -h         - download hp format (requires transfer protocol)
  load -a         - reserved for internal hp use
  load -e         - write only to emulation memory
  load -u         - write only to target memory
  load -o         - data received from the non-command source port
  load -s         - send a character string out the other port
  load -b         - data sent in binary (valid with -h option)
  load -x         - data sent in hex ascii (valid with -h option)
  load -q         - quiet mode
  load -p         - record ACK/NAK protocol (valid with -imt options)
  load -c <file>  - data is received from the 64000. file name format is:
                    <filename>:<userid>:absolute
```

Initialize the Emulator to a Known State

To initialize the emulator to a known state for this tutorial:

Note

It is especially important that you perform the following step if the emulator is being operated in a standalone mode controlled by only a data terminal. The only program entry available in this mode is through memory modification; consequently, if the emulator is reinitialized, emulation memory will be cleared and a great deal of tedious work could be lost.

1. Verify that no one else is using the emulator or will have need of configuration items programmed into the emulator.

2. Initialize the emulator by typing the command:

```
R> init -p
```

Set Up the Proper Emulation Configuration

Set Up Emulation Conditions

To set the emulator's configuration values to the proper state for this tutorial, do this:

1. Type:

```
R> cf
```

You should see the following configuration items displayed:

```
cf chip=338
cf clk=int
cf mode=ext
cf nmi=en
cf rrt=dis
cf rsp=9
cf trst=en
```

Note

The individual configuration items won't be explained in this example; refer to Chapter 3 of this manual and the *Reference* manual for details.

1. If the configuration items displayed on your screen don't match the ones listed above, here is how to make them agree:

   For each configuration item that does not match, type:

```
R> cf <config_item>=<value>
```

For example, if you have the following configuration items displayed (those in **bold** indicate items different from the list above):

```
cf clk=ext
cf mode=ext
cf nmi=en
cf rrt=en
cf rsp=9
cf trst=en
```

To make these configuration values agree with the desired values, type:

```
R> cf clk=int
R> cf rrt=dis
```

2. Now, you need to set up the stack pointer.
   Type:

```
R> cf rsp=0ff80
```

3. Let's go ahead and set up the proper break conditions.
   Type:

```
R> bc
```

   You will see:

```
bc -d bp #disable
bc -e rom #enable
bc -d bnct #disable
bc -d cmbt #disable
bc -d trig1 #disable
bc -d trig2 #disable
```

For each break condition that does not match the one listed, use one of the following commands:

To enable break conditions that are currently disabled, type:

```
R> bc -e <breakpoint type>
```

To disable break conditions that are currently enabled, type:

```
R> bc -d <breakpoint type>
```

For example, if typing **bc** gives the following list of break conditions:

```
bc -d bp #disable
bc -d rom #disable
bc -d bnct #disable
bc -d cmbt #disable
bc -e trig1 #enable
bc -e trig2 #enable
```

(items in **bold** indicate improper values for this example)

Type the following commands to set the break conditions correctly for this example:

R> **bc -e rom**

(this enables the write to ROM break)

R> **bc -d trig1 trig2**

(this disables break on triggers from the analyzer)

Map Memory

The emulation memory can be configured as you desire. You can define memory area as emulation RAM, emulation ROM, target RAM, target ROM or guarded memory. For this tutorial, map the address 0 hex through 1fff hex as emulation ROM, and fe80 hex through ff7f hex as emulation RAM.

Type:

R> **map 0..1fff erom**
R> **map 0fe80..0ff7f eram**

To verify that memory blocks are mapped properly, type:

R> **map**

You will see:

```
# remaining number of terms   : 14
# remaining emulation memory  : df00h bytes
map  00000..01fff     erom    # term 1
map  0fe80..0ff7f     eram    # term 2
map  other tram
```

Transfer Code into Emulation Memory

Transferring Code from a Terminal In Standalone Configuration

To transfer code into emulation memory from a data terminal running in standalone mode, you must use the modify memory commands. This is necessary because you have no host computer transfer facilities to automatically download the code for you (as if you would if you were using the transparent configuration or the remote configuration.) To minimize the effects of typing errors, you will modify only one row of memory at a time in this example. Do the following:

1. Enter the data information for the program by typing the following commands:

```
R> m 1100..110f=54,48,49,53,20,49,53,20,4d,45,53,53,41,47,45,20
R> m 1110..111f=41,54,48,49,53,20,49,53,20,4d,45,53,53,41,47,45
R> m 1120..112f=20,42,49,4e,56,41,4c,49,44,20,43,4f,4d,4d,41,4e
R> m 1130=44
```

You could also type the following line instead:

```
R> m 1100="THIS IS MESSAGE ATHIS IS MESSAGE BINVALID COMMAND"
```

2. You should now verify that the data area of the program is correct by typing:

```
R> m 1100..1130
```

You should see:

```
01100..0110f     54 48 49 53 20 49 53 20 4d 45 53 53 41 47 45 20
01110..0111f     41 54 48 49 53 20 49 53 20 4d 45 53 53 41 47 45
01120..0112f     20 42 49 4e 56 41 4c 49 44 20 43 4f 4d 4d 41 4e
01130..01130     44
```

If this is not correct, you can correct the errors by re-entering only the modify memory commands for the particular rows of memory that are wrong.

For example, if row 1100..110f shows these values:

```
01100..0110f     54 48 49 53 20 20 49 53 20 4d 45 53 53 41 47 45
```

you can correct this row of memory by typing:

```
R> m 1100..110f=54,48,49,53,20,49,53,20,4d,45,53,53,41,47,45,20
```

Or, you might need to modify only one location, as in the instance where address 110f hex equals 21 hex rather than 20 hex. Type:

```
R> m 110f=20
```

3. Enter the program information by typing the following commands:
   (Note the hex letters must be preceded by a digit)

```
R> m 1000..100f=79,07,0ff,80,79,01,0fe,80,0f8,00,68,98,68,01a,0aa,00
R> m 1010..101f=47,0fa,0aa,41,47,06,0aa,42,47,0a,40,10,0fb,11,79,04
R> m 1020..101f=11,00,40,0e,0fb,11,79,04,11,11,40,06,0fb,0f,79,04
R> m 1030..103f=11,22,79,05,0ff,00,0fe,20,68,0d8,0b,05,1a,0e,46,0f8
R> m 1040..104f=79,05,0ff,00,6c,4e,68,0de,0b,05,1a,0b,46,0f6,40,0b8
```

4. You should now verify that the program area is correct by typing:

```
R> m 1000..104f
```

You should see:

```
01000..0100f     79 07 ff 80 79 01 fe 80 f8 00 68 98 68 1a aa 00
01010..0101f     47 fa aa 41 47 06 aa 42 47 0a 40 10 fb 11 79 04
01020..0102f     11 00 40 0e fb 11 79 04 11 11 40 06 fb 0f 79 04
01030..0103f     11 22 79 05 ff 00 fe 20 68 d8 0b 05 1a 0e 46 f8
01040..0104f     79 05 ff 00 6c 4e 68 de 0b 05 1a 0b 46 f6 40 b8
```

If this is not correct, you can correct the errors by re-entering only the modify memory commands for the particular rows of memory that are wrong.

Transferring Code From A Host, HP 64700 In Transparent Configuration

The method provided in this example assumes that you are running an HP 64876 H8/300 Assembler/Linkage Editor on an HP 9000/300 computer running the HP-UX operating system. In addition, you must have the HP 64000 **transfer** software running on your host.

If you are not using an HP 64876 H8/300 Assembler/Linkage Editor, you may be able to adapt the methods below to load your code into the emulator (refer to the *HP 64700 Reference* manual for help).

If you are not able to transfer code from your host to the emulator using one of these methods, use the method described previously under "Transferring Code From A Terminal In Standalone Mode", as it will work in all cases. However, transferring code using host transfer facilities is easier and faster than modifying memory locations, especially for large programs.

1. First, you must establish communications with your host computer through the transparent mode link provided in the HP 64700. Type:

```
R> xp -s 02a
```

   This sets the second escape character to "*".(The first escape character remains at the HP 64700 powerup default of hex 01b, which is the ASCII <ESC>character.) The sequence "<ESC>*" toggles the transparent mode software within the HP 64700 for the duration of one command (that is, any valid line of HP 64700 commands (not exceed 254 characters) concatenated by semicolons and terminated by a <carriage return>). Refer to the *Reference* manual for more information on the **xp** command.

   Enable the transparent mode link by typing:

```
R> xp -e
```

   If you then press <RETURN> a few times, you should see:

```
login:
login:
login:
```

This is the login prompt for an HP-UX host system. (Your prompt may differ depending on how your system manager has configured your system.)

2. Log in to your host system and start up an editor such as "vi". You should now enter the source code for the sample program shown at the beginning of the chapter. When finished, save the program to filename "sampprog.src".

Note

If you need help learning how to log in to your HP-UX host system or use other features of the system, such as editors, refer to the HP-UX Concepts and Tutorials guides and your HP-UX system administrator.

3. Assemble your code with the following command.

```
$ h83asm sampprog
```

If any assembly errors were reported, re-edit your file and verify that the code was entered correctly.

4. Link the program to the correct addresses and generate absolute file with the following command.

```
$ h8lnk sampprog
```

5. Convert the Hitachi SYSROF absolute file generated above into HP format with the following command. This is needed to load the file into the emulator. Refer to the *HP 64876 H8/300 Assembler/Linkage Editor* manual for more details.

```
$ h83cnvhp -x sampprog
```

An HP format absolute file sampprog.X will be generated.

Now it's time to transfer your code into the emulator. Do the following:

1. Disable the transparent mode so that your terminal will talk directly to the emulator. Type:

```
$ <ESC>* xp -d
```

The "<ESC>*" sequence temporarily toggles the transparent mode so that the emulator will accept commands; "xp -d" then fully disables the transparent mode.

2. Load code into the emulator by typing:

```
R> load -hbo
transfer -rtb sampprog.X<ESC>*
(NOTE: DO NOT TYPE CARRIAGE RETURN!)
```

The system will respond:

```
##
R>
```

load -hbo tells the emulator to load code expected in HP binary file format and to expect the data from the other port (the one connected to the host). It then puts you in communication with the host; you then enter the transfer command to start the HP 64000 transfer utility. Typing "<ESC>*" tells the system to return to the emulator after transferring the code. The "##" marks returned by the system indicates that the emulator loaded two records from the host.

3. At this point you should examine a portion of memory to verify that your code was loaded correctly.

Type:

```
R> m 1100..1130
```

You should see:

```
01100..0110f     54 48 49 53 20 49 53 20 4d 45 53 53 41 47 45 20
01110..0111f     41 54 48 49 53 20 49 53 20 4d 45 53 53 41 47 45
01120..0112f     20 42 49 4e 56 41 4c 49 44 20 43 4f 4d 4d 41 4e
01130..01130     44
```

If your system does not match, verify 1) that you entered the source code correctly; 2) that you entered the linker parameters correctly.

Looking at Your Code

Now that you have loaded your code into emulation memory, you can display it in mnemonic format. Type:

```
R> m -dm 1000..104f
```

You will see:

```
01000  -              MOV.W #ff80,R7
01004  -              MOV.W #fe80,R1
01008  -              MOV.B #00,R0L
0100a  -              MOV.B R0L,@R1
0100c  -              MOV.B @R1,R2L
0100e  -              CMP.B #00,R2L
01010  -              BEQ 100c
01012  -              CMP.B #41,R2L
01014  -              BEQ 101c
01016  -              CMP.B #42,R2L
01018  -              BEQ 1024
0101a  -              BRA 102c
0101c  -              MOV.B #11,R3L
0101e  -              MOV.W #1100,R4
01022  -              BRA 1032
01024  -              MOV.B #11,R3L
01026  -              MOV.W #1111,R4
0102a  -              BRA 1032
0102c  -              MOV.B #0f,R3L
0102e  -              MOV.W #1122,R4
01032  -              MOV.W #ff00,R5
01036  -              MOV.B #20,R6L
01038  -              MOV.B R0L,@R5
0103a  -              ADDS #1,R5
0103c  -              DEC R6L
0103e  -              BNE 1038
01040  -              MOV.W #ff00,R5
01044  -              MOV.B @R4+,R6L
01046  -              MOV.B R6L,@R5
01048  -              ADDS #1,R5
0104a  -              DEC R3L
0104c  -              BNE 1044
0104e  -              BRA 1008
```

Familiarize Yourself with the System Prompts

Note

The following steps are not intended to be complete explanations of each command; the information is only provided to give you some idea of the meanings of the various command prompts you may see and reasons why the prompt changes as you execute various commands.

You should gain some familiarity with the HP 64700 emulator command prompts by doing the following:

1. Ignore the current command prompt. Type:

```
*> rst
```

   You will see:

```
R>
```

   The **rst** command resets the emulation processor and holds it in the reset state. The "R>" prompt indicates that the processor is reset.

2. Type:

```
R> r 1000
```

   You will see:

```
U>
```

   The **r** command runs the processor from address 1000 hex.

3. Type:

```
U> b
```

   You will see:

```
M>
```

The **b** command causes the emulation processor to "break" execution of whatever it was doing and begin executing within the emulation monitor. The "M>" prompt indicates that the emulator is running in the monitor.

Running the Sample Program

4. Type:

```
M> r 1000
```

The emulator changes state from background to foreground and begins running the sample program from location 1000 hex.

Note

The default number base for address and data values within HP 64700 is hexadecimal. Other number bases may be specified. Refer to the *HP 64700 Reference* manual for further details.

5. Let's look at the registers to verify that the address registers were properly initialized with the pointers to the input and output areas. Type:

```
U> reg
```

You will see:

```
reg pc=100c ccr=84 r0=0000 r1=0fe80 r2=0000 r3=0000 r4=0000 r5=0000 r6=0000
reg r7=0ff80 sp=0ff80 mdcr=e7
```

Notice that R1 contains fe80 hex.

6. Verify that the input area command byte was cleared during initialization.

   Type:

```
U> m -db 0fe80
```

   You will see:

```
0fe80..0fe80     00
```

   The input byte location was successfully cleared.

7. Now we will use the emulator features to make the program work. Remember that the program writes specific messages to the output area depending on what the input byte location contains. Type:

```
U> m 0fe80=41
```

   This modifies the input byte location to the hex value for an ASCII "A". Now let's check the output area for a message.

```
U> m 0ff00..0ff1f
```

   You will see:

```
0ff00..0ff0f     54 48 49 53 20 49 53 20 4d 45 53 53 41 47 45 20
0ff10..0ff1f     41 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

   These are the ASCII values for MESSAGE_A.

   Repeat the last two commands twice. The first time, use 42 instead of 41 at location fe00 hex and note that MESSAGE_B overwrites MESSAGE_A. Then try these again, using any number except 00, 41, or 42 and note that the INVALID_INPUT message is written to this area.

Stepping Through the Program

8. You can also direct the emulator processor to execute one instruction or number of instructions. Type:

```
M> s 1 1000;reg
```

This command steps 1 instruction from address 1000 hex, and displays registers. You will see:

```
01000  -               MOV.W #ff80,R7
PC = 01004
reg pc=1004 ccr=88 r0=0000 r1=fe80 r2=0000 r3=0000 r4=1111 r5=ff11 r6=0041
reg r7=ff80 sp=ff80 mdcr=e7
```

Notice that PC contains 1004 hex.

9. To step one instruction from present PC, you only need to type **s** at prompt. Type:

```
M> s;reg
```

You will see:

```
01004  -               MOV.W #0fe80,R1
PC = 01008
reg pc=1008 ccr=88 r0=0000 r1=fe80 r2=0000 r3=0000 r4=1111 r5=ff11 r6=0041
reg r7=ff80 sp=ff80 mdcr=e7
```

Tracing Program Execution

Now let's use the emulation analyzer to trace execution of the program. Suppose that you would like to start the trace when the program begins writing data to the message output area.You can do this by specifying analyzer trigger upon encountering the address fe00 hex. Furthermore, you might want to store only the data written to the output area. This can be accomplished by modifying what is known as the "analyzer storage specification".

Note

For this example, you will be using the analyzer in the easy configuration, which simplifies the process of analyzer measurement setup. The complex configuration allows more powerful measurements, but requires more interaction from you to set up those measurements. For more information on easy and complex analyzer configurations and the analyzer, refer to the *HP 64700 Analyzer User's Guide* and the *Reference.*

Now, let's set the trigger specification. Type:

```
M> tg addr=0ff00
```

To store only the accesses to the address range ff00 through ff11 hex, type:

```
M> tsto addr=0ff00..0ff11
```

Let's change the data format of the trace display so that you will see the output message writes displayed in ASCII format:

```
M> tf addr,h data,A count,R seq
```

Start the trace by typing:

```
M> t
```

You will see:

```
Emulation trace started
```

To start the emulation run, type:

```
M> r 1000
```

Now, you need to have a "command" input to the program so that the program will jump to the output routines (otherwise the trigger will not be found, since the program will never access address fe00 hex). Type:

```
U> m 0fe80=41
```

To display the trace list, type:

```
U> tl 0..34
```

You will see:

```
 Line  addr,H  data,A    count,R  seq
-----  ------  ------  ---------  ---
    0  ff00    ..            ---   +
    1  ff01    ..       1.200 uS   .
    2  ff02    ..       1.200 uS   .
    3  ff03    ..       1.200 uS   .
    4  ff04    ..       1.200 uS   .
    5  ff05    ..       1.200 uS   .
    6  ff06    ..       1.200 uS   .
    7  ff07    ..       1.200 uS   .
    8  ff08    ..       1.200 uS   .
    9  ff09    ..       1.200 uS   .
   10  ff0a    ..       1.200 uS   .
   11  ff0b    ..       1.200 uS   .
   12  ff0c    ..       1.200 uS   .
   13  ff0d    ..       1.200 uS   .
   14  ff0e    ..       1.200 uS   .
   15  ff0f    ..       1.200 uS   .
   16  ff10    ..       1.200 uS   .
   17  ff11    ..       1.200 uS   .
   18  ff00    TT       19.00 uS   .
   19  ff01    HH       1.800 uS   .
   20  ff02    II       1.800 uS   .
   21  ff03    SS       1.800 uS   .
   22  ff04    ..       1.800 uS   .
   23  ff05    II       1.800 uS   .
   24  ff06    SS       1.800 uS   .
   25  ff07    ..       1.800 uS   .
   26  ff08    MM       1.800 uS   .
   27  ff09    EE       1.800 uS   .
   28  ff0a    SS       1.800 uS   .
   29  ff0b    SS       1.800 uS   .
   30  ff0c    AA       1.800 uS   .
   31  ff0d    GG       1.800 uS   .
   32  ff0e    EE       1.800 uS   .
   33  ff0f    ..       1.800 uS   .
   34
```

If you look at the last lines of the trace listing, you will notice that the analyzer seems to have stored only part of the output message, even though you specified more than the full range needed to store all of the message. The reason for this is that the analyzer has a storage pipeline, which holds states that have been acquired but not yet written to trace memory. To see all of the states, halt the analyzer by typing:

```
U> th
```

You will see:

```
Emulation trace halted
```

Now display the trace list:

```
U> tl 0..34
```

You will see:

```
 Line  addr,H  data,A    count,R  seq
-----  ------  ------  ---------  ---
    0  ff00    ..            ---   +
    1  ff01    ..       1.200 uS   .
    2  ff02    ..       1.200 uS   .
    3  ff03    ..       1.200 uS   .
    4  ff04    ..       1.200 uS   .
    5  ff05    ..       1.200 uS   .
    6  ff06    ..       1.200 uS   .
    7  ff07    ..       1.200 uS   .
    8  ff08    ..       1.200 uS   .
    9  ff09    ..       1.200 uS   .
   10  ff0a    ..       1.200 uS   .
   11  ff0b    ..       1.200 uS   .
   12  ff0c    ..       1.200 uS   .
   13  ff0d    ..       1.200 uS   .
   14  ff0e    ..       1.200 uS   .
   15  ff0f    ..       1.200 uS   .
   16  ff10    ..       1.200 uS   .
   17  ff11    ..       1.200 uS   .
   18  ff00    TT       19.00 uS   .
   19  ff01    HH       1.800 uS   .
   20  ff02    II       1.800 uS   .
   21  ff03    SS       1.800 uS   .
   22  ff04    ..       1.800 uS   .
   23  ff05    II       1.800 uS   .
   24  ff06    SS       1.800 uS   .
   25  ff07    ..       1.800 uS   .
   26  ff08    MM       1.800 uS   .
   27  ff09    EE       1.800 uS   .
   28  ff0a    SS       1.800 uS   .
   29  ff0b    SS       1.800 uS   .
   30  ff0c    AA       1.800 uS   .
   31  ff0d    GG       1.800 uS   .
   32  ff0e    EE       1.800 uS   .
   33  ff0f    ..       1.800 uS   .
   34  ff10    AA       1.800 uS   .
```

As you can see, all of the requested states have been captured by the analyzer.

Using Software Breakpoints

You can stop program execution at specific address by using **bp** (software breakpoint) command. When you define a software breakpoint to a certain address, the emulator will replace the opcode with one of undefined opcode (5770 hex) as software breakpoint instruction. When the emulator detects the special instruction, user program breaks to the monitor, and the original opcode will be placed at the breakpoint address. A subsequent run or step command will execute from this address.

If the special instruction was not inserted as the result of **bp** command (in other words, it is part of the user program), the "Undefined software breakpoint" message is displayed.

Note

You can 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 software breakpoint instruction, you cannot define software breakpoints in target ROM. You can, however, copy target ROM into emulation memory by **cim** command when you are using the background monitor. (Refer to *HP 64700 Terminal Interface Reference* manual.)

Displaying and Modifying the Break Conditions

Before you can define software breakpoints, you must enable software breakpoints with the **bc** (break conditions) command. To view the default break conditions and change the software breakpoint condition, enter the following commands.

```
M> bc
```

```
bc -d bp #disable
bc -e rom #enable
bc -d bnct #disable
bc -d cmbt #disable
bc -d trig1 #disable
bc -d trig2 #disable
```

```
M> bc -e bp
```

Defining a Software Breakpoint

Now that the software breakpoint is enabled, you can define software breakpoints. Enter the following command to break on the address of the OUTPUT label.

```
M> bp 1032
```

Run the program and verify that execution broke at the appropriate address.

```
M> r 1000
U> m 0fe80=41
!ASYNC_STAT  615! Software break point: 01032
```

```
M> reg
reg pc=1032 ccr=80 r0=0000 r1=fe80 r2=0041 r3=0011 r4=1100 r5=ff11 r6=0041
reg r7=ff80 sp=ff80 mdcr=e7
```

Notice that PC contains 1032.

When a breakpoint is hit, it becomes disabled. You can use the **-e** option to the **bp** command to reenable the software breakpoint.

```
M> bp
###BREAKPOINT FEATURE IS ENABLED###
bp 01032 #disabled
```

```
M> bp -e 1032
M> bp
###BREAKPOINT FEATURE IS ENABLED###
bp 01032 #enabled
```

```
M> r 1000
U> m 0fe80=41
!ASYNC_STAT  615! Software breakpoint: 01032
```

```
M> bp
###BREAKPOINT FEATURE IS ENABLED###
bp 01032 #disabled
```

Searching Memory for Strings or Numeric Expressions

The HP 64700 Emulator provides you with tools that allow you to search memory for data strings or numeric expressions. For example, you might want to know exactly where a string is loaded. To locate the position of the string "THIS IS MESSAGE A" in the sample program. Type:

```
M> ser 0..1fff="THIS IS MESSAGE A"

pattern match at address:  01100
```

You can also find numeric expressions. For example, you might want to find all of the **BEQ** instructions in the sample program. Since a **BEQ** instruction begins with 47 hex, you can search for that value by typing:

```
M> ser -db 1000..1049=47

pattern match at address: 01010
pattern match at address: 01014
pattern match at address: 01018
```

Making Program Coverage Measurements

In testing your program, you will often want to verify that all possible code segments are executed. With the sample program, we might want to verify that all of the code is executed if a command "A", command "B", and an unrecognized command are input to the program.

To make this measurement, we must first reset the coverage status.

```
M> cov -r
```

Note

You should **always** reset the coverage status before making a coverage measurement. Any emulator system command which accesses emulation memory will affect the coverage status bit, resulting in measurement errors if the coverage status is not reset.

Now, run the program and input the three commands:

```
M> r 1000
U> m 0fe80=41
U> m 0fe80=42
U> m 0fe80=43
```

Make the coverage measurement:

```
U> cov 1000..104f
percentage of memory accessed:  % 100.0
```

Trace Analysis Considerations

There are some points you need to attend to in using the emulation analyzer. The following section describes such points.

How to Specify the Trigger Condition

Suppose that you would like to start the trace when the program begins executing PROCESS_COMM routine.

To initialize the emulation analyzer, type:

```
U> tinit
```

To set the trigger condition, type:

```
U> tg addr=1012
```

Start the trace and modify memory so that the program will jump to the PROCESS_COMM routine:

```
U> t
```

```
U> m 0fe80=41
```

To display the trace list, type:

```
 Line   addr,H  H8/338 mnemonic,H                      count,R  seq
-----   ------  ---------------------------------  ---------  ---
    0   1012      aa41  fetch mem                   0.200 uS   +
    1   100c    MOV.B @R1,R2L                       0.200 uS   .
    2   100e    CMP.B #00,R2L                       0.200 uS   .
    3   fe80      00    read  mem byte              0.200 uS   .
    4   1010    BEQ 100c                            0.200 uS   .
    5   1012      aa41  fetch mem                   0.200 uS   .
    6   100c    MOV.B @R1,R2L                       0.200 uS   .
    7   100e    CMP.B #00,R2L                       0.200 uS   .
    8   fe80      00    read  mem byte              0.200 uS   .
    9   1010    BEQ 100c                            0.200 uS   .
   10   1012      aa41  fetch mem                   0.200 uS   .
   11   100c    MOV.B @R1,R2L                       0.200 uS   .
   12   100e    CMP.B #00,R2L                       0.200 uS   .
   13   fe80      00    read  mem byte              0.200 uS   .
   14   1010    BEQ 100c                            0.200 uS   .
   15   1012      aa41  fetch mem                   0.200 uS   .
   16   100c    MOV.B @R1,R2L                       0.200 uS   .
   17   100e    CMP.B #00,R2L                       0.200 uS   .
   18   fe80      00    read  mem byte              0.200 uS   .
   19   1010    BEQ 100c                            0.200 uS   .
   20   1012      aa41  fetch mem                   0.200 uS   .
```

```
U> tl 0..20
```

This is not what we were expecting to see. (We expected to see the program executed PROCESS_COMM routine which starts from 1012 hex.) As you can see at the fist line of the trace list, address 1012 hex appears on the address bus during the program executing READ_INPUT loop. This triggered the emulation analyzer before PROCESS_COMM routine was executed. To avoid mis-trigger by this cause, set the trigger condition to the second instruction of the routine you want to trace. Type:

```
U> tg addr=1014
```

To change the trigger position so that 10 states appear before the trigger in the trace list, type:

```
U> tp -b 10
```

Start the trace again and modify memory:

```
U> t
U> m 0fe80=41
```

Now display the trace list:

```
U> tl -10..10
```

```
 Line   addr,H  H8/338 mnemonic,H                      count,R  seq
-----   ------  ---------------------------------  ---------  ---
  -10   100e    CMP.B #00,R2L                            ---   .
   -9   fe00      00    read  mem byte               0.200 uS   .
   -8   1010    BEQ 100c                             0.200 uS   .
   -7   1012      aa41  fetch mem                     0.200 uS   .
   -6   100c    MOV.B @R1,R2L                        0.200 uS   .
   -5   100e    CMP.B #00,R2L                        0.200 uS   .
   -4   fe00      41    read  mem byte               0.200 uS   .
   -3   1010    BEQ 100c                             0.200 uS   .
   -2   1012    CMP.B #41,R2L                        0.200 uS   .
   -1   100c      681a  fetch mem                     0.200 uS   .
    0   1014    BEQ 101c                             0.200 uS   +
    1   1016      aa42  fetch mem                     0.200 uS   .
    2   101c    MOV.B #11,R3L                        0.200 uS   .
    3   101e    MOV.W #1100,R4                       0.200 uS   .
    4   1020      1100  fetch mem                     0.200 uS   .
    5   1022    BRA 1032                             0.200 uS   .
    6   1024      fb11  fetch mem                     0.200 uS   .
    7   1032    MOV.W #ff00,R5                       0.200 uS   .
    8   1034      ff00  fetch mem                     0.200 uS   .
    9   1036    MOV.B #20,R6L                        0.200 uS   .
   10   1038    MOV.B R0L,@R5                        0.200 uS   .
```

As you can see, the analyzer captured the execution of PROCESS_COMM routine which starts from line -2 of the trace list.

Store Condition and Disassembling

When you specify store condition with **tsto** command, disassembling of program execution may not be accurate.

Type:

```
U> tinit
U> t
U> tl 0..20
```

```
 Line   addr,H  H8/338 mnemonic,H                    count,R  seq
-----   ------  ---------------------------------  ---------  ---
    0   fe80      00    read  mem byte                   ---   +
    1   1010    BEQ 100c                           0.200 uS   .
    2   1012      aa41  fetch mem                   0.200 uS   .
    3   100c    MOV.B @R1,R2L                      0.200 uS   .
    4   100e    CMP.B #00,R2L                      0.200 uS   .
    5   fe80      00    read  mem byte               0.200 uS   .
    6   1010    BEQ 100c                           0.200 uS   .
    7   1012      aa41  fetch mem                   0.200 uS   .
    8   100c    MOV.B @R1,R2L                      0.200 uS   .
    9   100e    CMP.B #00,R2L                      0.200 uS   .
   10   fe80      00    read  mem byte               0.200 uS   .
   11   1010    BEQ 100c                           0.200 uS   .
   12   1012      aa41  fetch mem                   0.200 uS   .
   13   100c    MOV.B @R1,R2L                      0.200 uS   .
   14   100e    CMP.B #00,R2L                      0.200 uS   .
   15   fe80      00    read  mem byte               0.200 uS   .
   16   1010    BEQ 100c                           0.200 uS   .
   17   1012      aa41  fetch mem                   0.200 uS   .
   18   100c    MOV.B @R1,R2L                      0.200 uS   .
   19   100e    CMP.B #00,R2L                      0.200 uS   .
   20   fe80      00    read  mem byte               0.200 uS   .
```

The program is executing READ_INPUT loop.

Now, specify the store condition so that only accesses to the address range 2000 hex through 20ff hex will be stored:

```
U> tsto addr=1000..10ff
```

Start the trace and display the trace list:

```
U> t
```

```
 Line   addr,H  H8/338 mnemonic,H                    count,R  seq
-----   ------  ---------------------------------  ---------  ---
    0   100e      aa00  fetch mem                      ---   +
    1   1010    BEQ 100c                           0.400 uS   .
    2   1012      aa41  fetch mem                   0.200 uS   .
    3   100c    MOV.B @R1,R2L                      0.200 uS   .
    4   100e      aa00  fetch mem                   0.200 uS   .
    5   1010    BEQ 100c                           0.400 uS   .
    6   1012      aa41  fetch mem                   0.200 uS   .
    7   100c    MOV.B @R1,R2L                      0.200 uS   .
    8   100e      aa00  fetch mem                   0.200 uS   .
    9   1010    BEQ 100c                           0.400 uS   .
   10   1012      aa41  fetch mem                   0.200 uS   .
   11   100c    MOV.B @R1,R2L                      0.200 uS   .
   12   100e      aa00  fetch mem                   0.200 uS   .
   13   1010    BEQ 100c                           0.400 uS   .
   14   1012      aa41  fetch mem                   0.200 uS   .
   15   100c    MOV.B @R1,R2L                      0.200 uS   .
   16   100e      aa00  fetch mem                   0.200 uS   .
   17   1010    BEQ 100c                           0.400 uS   .
   18   1012      aa41  fetch mem                   0.200 uS   .
   19   100c    MOV.B @R1,R2L                      0.200 uS   .
   20   100e      aa00  fetch mem                   0.200 uS   .
```

```
U> tl 0..20
```

As you can see, the executions of CMP.B instruction 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 store condition.

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.

```
M> tg data=<data>
```

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:

```
M> tg data=1234
M> tg data=12xx
M> tg data=0xx34
```

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

```
M> tg data=12xx
```

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.

You're now finished with the "Getting Started" example. You can proceed on with using the emulator and use this manual and the *Terminal Interface Reference* manual as needed to answer your questions.

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 perfromance 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.

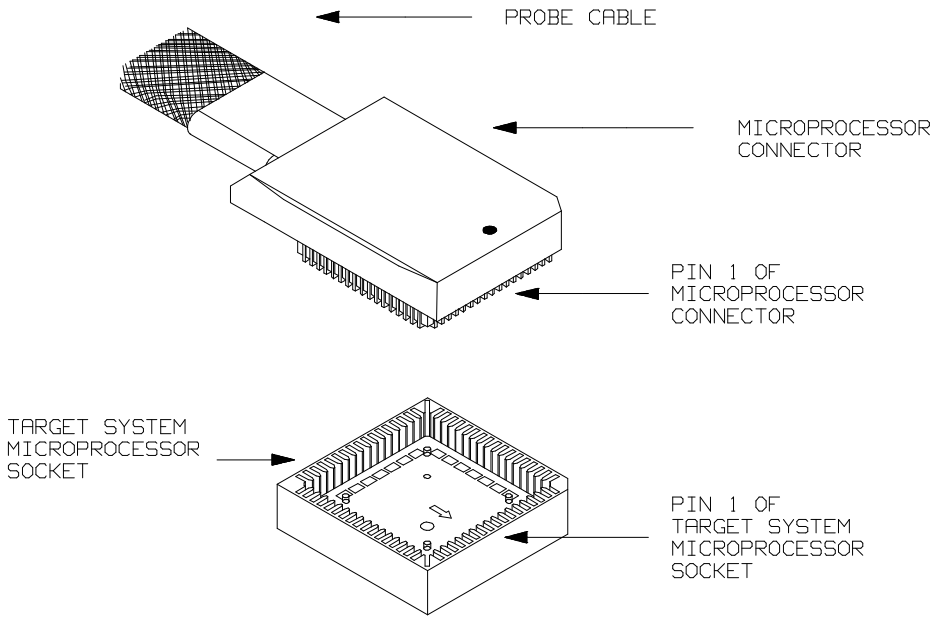


Figure 3-1. Installing the Probe (H8/338 emulator)

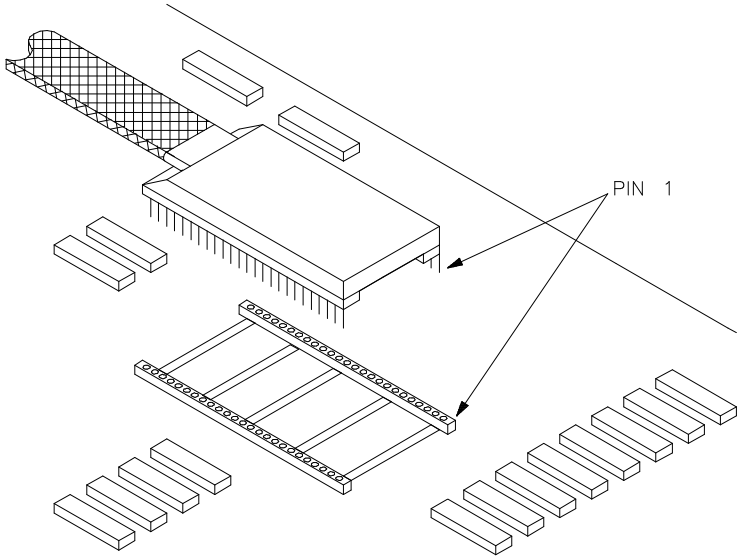


Figure 3-2. Installing the Probe (H8/329 emulator)

Pin State in Background

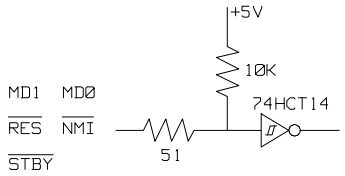
While the emulator is running the background monitor, probe pins are in the following state.

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

Target System Interface (H8/338)

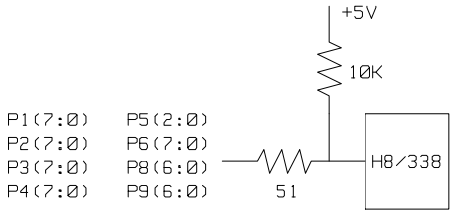
MD1 MD0
RES NMI
STBY

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



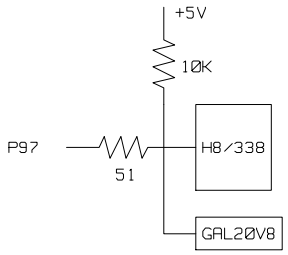
P1(7:0) P5(2:0)
P2(7:0) P6(7:0)
P3(7:0) P8(6:0)
P4(7:0) P9(6:0)

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



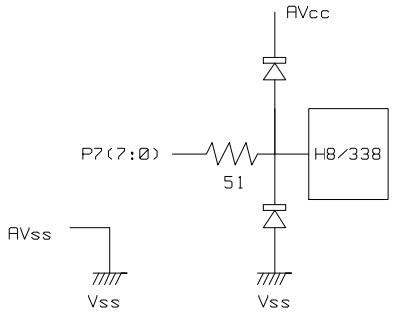
P97

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



P7(7:0)

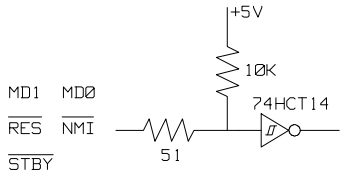
These signals are connected to H8/338 emulation processor through 51 ohm series resister.



Target System Interface (H8/329)

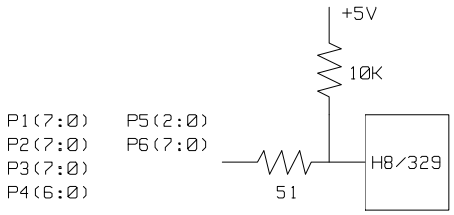
MD1 MD0
RES NMI
STBY

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



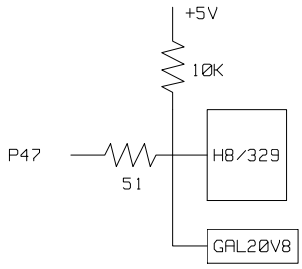
P1(7:0) P5(2:0)
P2(7:0) P6(7:0)
P3(7:0)
P4(6:0)

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



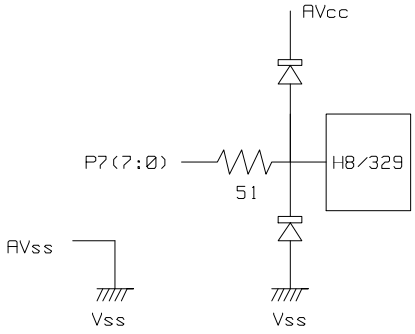
P47

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



P7(7:0)

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



Notes

4

Configuring the H8/338 Emulator

In this chapter, we will discuss:

- how to configure the HP 64700 emulator for H8/338 microprocessor to fit your particular measurement needs.
- some restrictions of HP 64700 emulator for H8/338 microprocessor.

Types of Emulator Configuration

The HP 64700 Emulator is different from other HP emulators (such as those in the HP 64000-UX system) in that there are several different classes of configuration commands.

Emulation Processor to Emulator/Target System

These are the commands which are generally thought of as "configuration" items in the context of other HP 64000 emulator systems. The commands in this group set up the relationships between the emulation processor and the target system, such as determining how the emulator responds to requests for the processor bus. Also, these commands determine how the emulation processor interacts with the emulator itself; memory mapping and the emulator's response to certain processor actions are some of the items which can be configured.

These commands are the ones which are covered in this chapter.

Commands Which Perform an Action or Measurement

Several of the emulator commands do not configure the emulator; they simply start an emulator program run or other measurement, begin or halt an analyzer measurement, or allow you to display the results of such measurements.

These commands are covered in the examples presented in earlier manual chapters; they are also covered in the *HP 64700 Terminal Interface Reference* manual.

Coordinated Measurements

These commands determine how the emulator interacts with other measurement instruments, such as external analyzers, or other HP 64700 emulators connected via the CMB (Coordinated Measurement Bus).

These commands are covered in the *HP 64700 CMB User's Guide* and in the *HP 64700 Terminal Interface Reference* Manual.

Analyzer

The analyzer configuration commands are those commands which actually specify what type of measurement the analyzer is to make.

Some of the analyzer commands are covered earlier in this manual. You can also refer to the *HP 64700 Terminal Interface: Analyzer User's Guide* and the *HP 64700 Terminal Interface Reference* manual.

System

This last group of commands is used by you to set the emulator's data communications protocol, load or dump contents of emulation memory, set up command macros, and so on.

These commands are covered earlier in this manual and in the manual titled *HP 64700 Terminal Interface: User's Reference.*

Emulation Processor to Emulator/Target System

As noted before, these commands determine how the emulation processor will interact with the emulator's memory and the target system during an emulation measurement.

cf

The **cf** command defines how the emulation processor will respond to certain target system signals. It also defines the type of emulation monitor to be used and optionally defines the location of that monitor in emulation memory.

To see the default configuration settings defined by the **cf** command, type:

```
M> cf
```

You will see:

```
cf chip=338
cf clk=int
cf mode=ext
cf nmi=en
cf rrt=dis
cf rsp=9
cf trst=en
```

Let's examine each of these emulator configuration options, with a view towards how they affect the processor's interaction with the emulator.

cf chip (H8/338) The **chip** configuration item defines the microprocessor you emulate.

```
M> cf chip=<chip_name>
```

Valid <chip_name> are the following:

<chip_name>	Description
336	The H8/338 emulator will emulate H8/336 microprocessor.
337	The H8/338 emulator will emulate H8/337 microprocessor.
338	The H8/338 emulator will emulate H8/338 microprocessor.

Note 

Executing this command will drive the emulator into the reset state.

cf chip (H8/329) The **chip** configuration item defines the microprocessor you emulate.

```
M> cf chip=<chip_name>
```

Valid <chip_name> are the following:

<chip_name>	Description
326	The H8/329 emulator will emulate H8/326 microprocessor.
327	The H8/329 emulator will emulate H8/327 microprocessor.
328	The H8/329 emulator will emulate H8/328 microprocessor.
329	The H8/329 emulator will emulate H8/329 microprocessr.

Note 

Executing this command will drive the emulator into the reset state.

cf clk The **clk** (clock) option allows you to select whether the emulation processor's clock will be sourced by your target system or by the emulator.

```
M> cf clk=int
```

You can select the emulator's internal 10 MHz system clock using the above command.

```
M> cf clk=ext
```

You can specify that the emulator should use the clock input to the emulator probe from the target system as the system clock. You must use a clock input conforming to the specifications for the H8/338 microprocessor.

Note 

The H8/338 emulator can operate up to **10 MHz system clock**.

Note 

Executing this command will drive the emulator into the reset state.

cf mode The **mode** (cpu operation mode) configuration item defines operation mode in which the emulator works.

```
M> cf mode=ext
```

The emulator will work using the mode setting by the target system. The target system must supply appropriate inputs to MD0 and MD1. If you are using the emulator out of circuit when **ext** is selected, the emulator will operate in mode 3.

```
M> cf mode=<mode_num>
```

When <mode_num> is selected, the emulator will operate in selected mode regardless of the mode setting by the target system.

Valid <mode_num> are following:

<mode_num>	Description
1	The emulator will operate in mode 1. (expanded mode without internal ROM)
2	The emulator will operate in mode 2. (expanded mode with internal ROM)
3	The emulator will operate in mode 3. (single chip mode)

Note 

Executing this command will drive the emulator into the reset state.

cf nmi The **nmi** (non maskable interrupt) configuration item determines whether or not the emulator responds to /NMI signal from the target system during foreground operation.

```
M> cf nmi=en
```

Using the above command, you can specify that the emulator will respond to /NMI from the target system.

```
M> cf nmi=dis
```

The emulator won't respond to /NMI 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.

Note

When an edge sensed interrupt is suspended, the emulator cannot perform step execution. Refer to the "Restrictions and Considerations" section in this chapter.

Note

Executing this command will drive the emulator into the reset state.

cf rrt The **rrt** (restrict to real time) option lets you configure the emulator so that commands which cause the emulator to break to monitor and return to the user program will be rejected by the emulator command interpreter.

```
M> cf rrt=en
```

You can restrict the emulator to accepting only commands which don't cause temporary breaks to the monitor by entering the above command. Only the following emulator run/stop commands will be accepted:

rst (resets emulation processor)

b (breaks processor to background monitor until you enter another command)

r (runs the emulation processor from a given location)

s (steps the processor through a piece of code -- returns to monitor after each step)

Commands which cause the emulator to break to the monitor and return, such as **reg**, **m** (for target memory display), and others will be rejected by the emulator.

Caution

If your target system circuitry is dependent on constant execution of program code, you should set this option to **cf rrt=en**. This will help insure that target system damage doesn't occur. However, remember that you can still execute the **rst**, **b** and **s** commands; you should use caution in executing these commands.

```
M> cf rrt=dis
```

When you use this command, all commands, regardless of whether or not they require a break to the emulation monitor, are accepted by the emulator.

cf rsp The **rsp** (reset stack pointer) configuration item allows you to specify a value to which the stack pointer will be set upon the transition from emulation reset into the emulation monitor.

```
R> cf rsp=XXXX
```

where **XXXX** is a 16-bit even address, will set the stack pointer to that value upon entry to the emulation monitor after an emulation reset. You **cannot** set **rsp** at the following location.

- Odd address
- Internal I/O register area

For example, to set the stack pointer to 0ff00 hex, type:

```
R> cf rsp=0ff00
```

Now, if you break the emulator to monitor using the **b** command, the stack pointer will be modified to the value 0ff00 hex.

Note

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

cf trst

The **trst** (target reset) configuration item 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 "waiting for the target system reset" (prompt is T>). (You can see the emulator status with **es** command.)

```
M> cf trst=en
```

When you enable target system reset with the above command, the emulator will respond to /RES input during foreground operation.

```
M> cf trst=dis
```

When disabled, the emulator won't respond to /RES input form the target system.

Note

Executing this command will drive the emulator into the reset state.

Memory Mapping

Before you begin an emulator session, you must specify the location and type of various memory regions used by your programs and your target system (whether or not it exists). You do this for several reasons:

- the emulator must know whether a given memory location resides in emulation memory or in target system memory. This allows the emulator to properly orient buffers for the given data transfer.
- the emulator needs to know the size of any emulation memory blocks so it can properly reserve emulation memory space for those blocks.
- the emulator must know if a given space is RAM (read/write), ROM (read only), or doesn't exist. This allows the emulator to determine if certain actions taken by the emulation processor are proper for the memory type being accessed. For example, if the processor tries to write to a emulation memory location mapped as ROM, the emulator will not permit the write (even if the memory at the given location is actually RAM). (You can optionally configure the emulator to break to the monitor upon such occurrence with the **bc -e rom** command.) Also, if

the emulation processor attempts to access a non existent location (known as "guarded"), the emulator will break to the monitor.

You use the **map** command to define memory ranges and types for the emulator. The H8/338 emulator memory mapper allows you to define up to 16 different map terms; each map term has a minimum size of 128 bytes. If you specify a value less than 128 byte, the emulator will automatically allocate an entire block. You can specify one of five different memory types (**erom, eram, trom, tram, grd**).

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

- input port at 0f000 hex
- output port at 0f100 hex
- program and data from 1000 through 3fff 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. Type the following commands:

```
R> map 0f000..0f100 tram
R> map 1000..3fff eram
R> map
```

```
# remaining number of terms   : 14
# remaining emulation memory  : d000h bytes
map  001000..003fff     eram    # term 1
map  00f000..00f17f     tram    # term 2
map  other tram
```

As you can see, the mapper rounded up the second term to 128 bytes block, since those are minimum size blocks supported by the H8/338 emulator.

Note

When you use the internal memory, you **must** map that area to emulation memory. When you power on the emulator, all memory space is mapped to target RAM. Therefore, if you don't map internal memory properly, you cannot access that area.

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.

For further information on mapping, refer to the examples in earlier chapters of this manual and to the *HP 64700 Terminal Interface User's Reference* manual.

Break Conditions

The **bc** command lets you configure the emulator's response to various emulation system and external events.

Write to ROM

If you want the emulator to break into the emulation monitor whenever the user program attempts to write to a memory region mapped as ROM, enter:

```
M> bc -e rom
```

You can disable this function by entering:

```
M> bc -d rom
```

When disabled, the emulator will not break to the monitor upon a write to ROM; however, it will not modify the memory location if the memory at that location is actually RAM.

Software Breakpoints

The **bp** command allows you to insert software traps in your code which will cause a break to the emulation monitor when encountered

during program execution. If you want to enable the insertion and use of software breakpoints by the **bp** command, enter:

```
M> bc -e bp
```

To disable use of software breakpoints, type:

```
M> bc -d bp
```

Any breakpoints which previously existed in memory are disabled, but are not removed from the breakpoint table.

Trigger Signals

The HP 64700 emulator provides four different trigger signals which allow you to selectively start or stop measurements depending on the signal state. These are the **bnct** (rear panel BNC input), **cmbt** (CMB trigger input), **trig1** and **trig2** signals (provided by the analyzer).

You can configure the emulator to break to the monitor upon receipt of any of these signals. Simply type:

```
M> bc -e <signal>
```

For example, to have the emulator break to monitor upon receipt of the **trig1** signal from the analyzer, type:

```
M> bc -e trig1
```

(Note: in this situation, you must also configure the analyzer to drive the **trig1** signal upon finding its trigger by entering **tgout trig1**).

Restrictions and Considerations

Monitor Break at Sleep/Standby Mode

When the emulator breaks into the monitor, sleep or software standby mode is released. For example, if you use the **reg** command at sleep mode, the emulator processor will go into normal state and start execution.

Store Condition and Trace

Disassembling in the trace list may not be accurate when the analyzer is used with store condition (specified by **tsto** command).

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, the following messages will be given when you attempt to execute the **s** command.

```
!ERROR  680! Stepping failed
!STATUS  686! Stepping aborted; number of steps completed: 0
```

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

Where to Find More Information

Due to the architecture of the HP 64700 emulators, there are a wide variety of items that affect how the emulator interacts with your system, controller, and other measuring instruments. If you need more configuration information, we suggest the following strategy:

If you need tutorial information --

- Emulator: look at this manual.
- Analyzer: look at the *Analyzer User's Guide* and this manual.
- CMB: look at the *CMB User's Guide.*

If you need reference information --

- Look at the *Terminal Interface User's Reference* manual (also contains some examples).

A

H8/338 Emulator Specific Command Syntax

The following pages contain descriptions of command syntax specific to the H8/338 emulator. The following syntax items are included (several items are part of other command syntax):

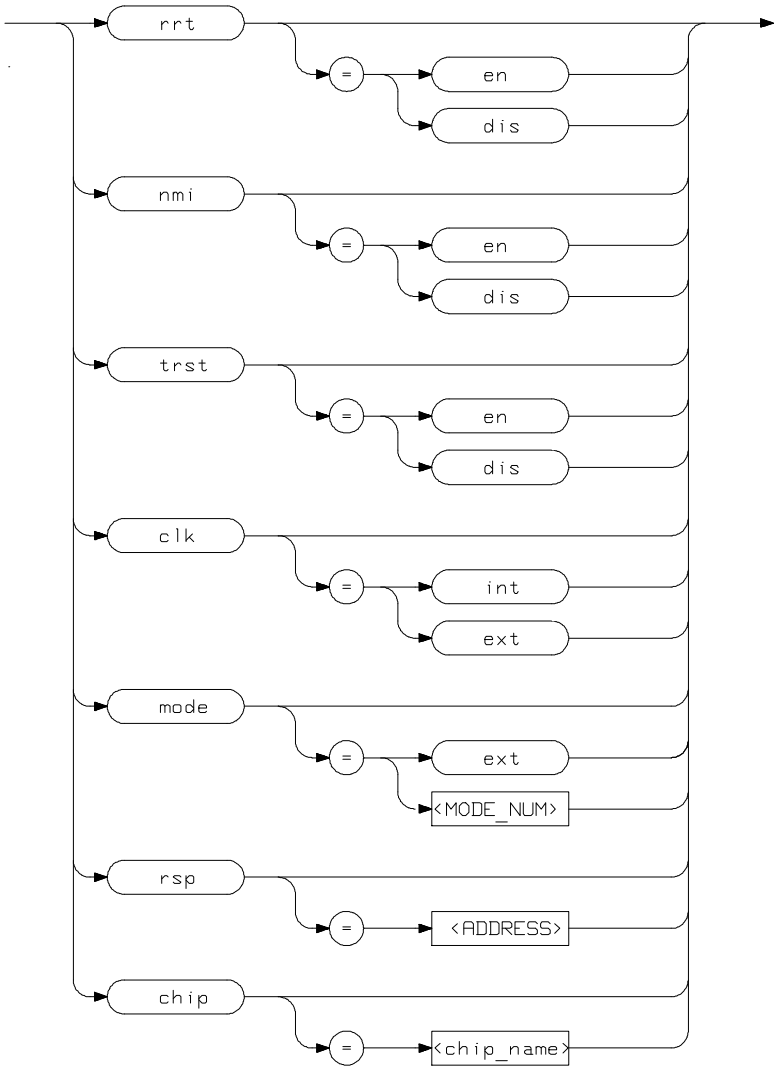
- <CONFIG_ITEMS>. May be specified in the **cf** (emulator configuration) and **help cf** commands.
- <DISPLAY_MODE>. May be specified in the **mo** (display and access mode), **m** (memory), and **ser** (search memory for data) commands. The display mode is used when memory locations are displayed or modified.
- <ADDRESS>. May be specified in emulation commands which allow addresses to be entered.
- <REG_NAME>. May be specified in the **reg** (register) command.

Command and error messages which are specific to the H8/338 emulator are also described in this chapter.

CONFIG_ITEMS

Summary H8/338 emulator configuration items.

Syntax



Description

The H8/338 emulator has several dedicated configuration items which allow you to specify the emulator's interaction with the target system and the rest of the emulation system. These items are:

clk	Select internal/external clock source.
mode	Determine emulator processor operation mode.
nmi	Enable/disable NMI (non maskable interrupt) from target system.
rrt	Restrict emulator to real time runs.
rsp	Specify system stack pointer value to load upon each transition from emulation reset to the monitor.
trst	Enable/disable target system reset.
chip	Define microprocessor to be emulated

Complete explanations of all configuration items are given in chapter 4 of this manual.

Examples

To select an external clock, type:

```
M> cf clk=ext
```

You can obtain the status of configuration items by typing the item name without a value. You can also specify multiple configuration items on the same line. Type:

```
M> cf nmi=dis rrt=dis clk
```

```
cf clk=ext
```

Related information

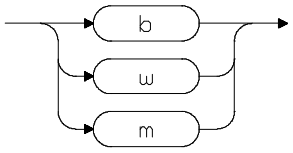
Refer to the **cf** syntax pages in the *User's Reference* manual. Also, refer to chapter 3 of this manual for complete information about each configuration item.

ACCESS MODE and DISPLAY MODE

Summary

Specify the memory display format or the size of memory locations to be modified.

Syntax



b Byte. Memory is displayed in a byte format, and when memory locations are modified, bytes are changed.

w Word. Memory is displayed in a word format, and when memory locations are modified, words are changed.

m Mnemonic. Memory is displayed in mnemonic format; that is, the contents of memory locations are inverse-assembled into mnemonics and operands. When memory locations are modified, the last non-mnemonic display mode specification is used. You cannot specify this display mode in the **ser** (search memory for data) command.

Note

Only byte is valid to ACCESS MODE since H8/338 microprocessor have 8 bits external data bus width.

Defaults

The **<DISPLAY_MODE>** is **b** at power up initialization. Display mode specifications are saved; that is, when a command changes the display mode, the new display mode becomes the current default.

Related Information

Refer to the **mo** syntax information in the *User's Reference* manual for further information on use of the mode command.

ADDRESS

Summary

Address specification used in emulation commands.

Description

The **<ADDRESS>** parameter used in emulation commands is specified in 16 bits address information.

Examples

m 1000

m 2000..20ff

REGISTER CLASS and NAME (H8/338 Emulator)

Summary

H8/338 register designators. All available register class names and register names are listed below.

<REG_CLASS>

<REG_NAME> Description

*** (All basic registers)**

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 (System control)

stcr	Serial timer control register
syscr	System control register
mdcr	Mode control register
iscr	IRQ sense control register
ier	IRQ enable register

port (I/O port)

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
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 (16 bit free running timer)

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 (8 bit timer 0)

tcr0	Timer control register
tcsr0	Timer control/status register
tcora0	Timer constant register A
tcorb0	Timer constant register B
tcnt0	Timer counter

tmr1 (8 bit timer 1)

tcr1	Timer control register
tcsr1	Timer control/status register
tcora1	Timer constant register A
tcorb1	Timer constant register B
tcnt1	Timer counter

pwm0 (PWM timer 0)

pwmtcr0	Timer control register
dtr0	Duty register
pwmtcnt0	Timer counter

pwm1 (PWM timer 1)

pwmtcr1	Timer control register
dtr1	Duty register
pwmtcnt1	Timer counter

sci0 (Serial communication interface 0)

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 (Serial communication interface 1)

smr1	Serial mode register
brr1	Bit rate register
scr1	Serial control register
tdr1	Transmit data register
ssr1	Serial status register
rdr	Receive data register

adc (A/D converter)

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 (D/A converter)

dadr0	D/A data register 0
dadr1	D/A data register 1 register
dacr	D/A control register

NOCLASS

The following register names are not included in any register class.

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 CLASS and NAME (H8/329 Emulator)

Summary

H8/329 register designators. All available register class names and register names are listed below.

<REG_CLASS>

<REG_NAME> Description

*** (All basic registers)**

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 (System control)

stcr	Serial timer control register
syscr	System control register
mdcr	Mode control register
iscr	IRQ sense control register
ier	IRQ enable register

port (I/O port)

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
p3prc	Port 3 input pull up MOS control register

frt (16 bit free running timer)

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 (8 bit timer 0)

tcr0	Timer control register
tcsr0	Timer control/status register
tcora0	Timer constant register A
tcorb0	Timer constant register B
tcnt0	Timer counter

tmr1 (8 bit timer 1)

tcr1	Timer control register
tcsr1	Timer control/status register
tcora1	Timer constant register A
tcorb1	Timer constant register B
tcnt1	Timer counter

sci (Serial communication interface)

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 (A/D converter)

adcb	A/D data register A
addrc	A/D data register B
addrd	A/D data register C
adcsr	A/D data register D
adcr	A/D control/status register
	A/D control register

NOCLASS

The following register names are not included in any register class.

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

Emulator Specific Error Messages

The following is the error messages which are specific to the H8/338 emulator. The cause of the errors is described, as well as the action you must take to remedy the situation.

Message

141 : Stack is in I/O registers

Cause

This error occurs when you attempt to execute user program with the stack pointer set at internal I/O register area.

Action

Set up the stack pointer at proper location with **cf rsp** command. Refer to chapter 4 of this manual for more information.

Notes

Index

Notes