HP 64732

H8/510 Emulator Terminal Interface

User's Guide

HEWLETT PACKARD

HP Part No. 64732-97003
Printed in U.S.A.
February 1993

Edition 2

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
P.O. Box 2197
1900 Garden of the Gods Road
Colorado Springs, CO 80901-2197, U.S.A.

Printing History

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

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

Edition 1 **64732-97000, April 1991**

Edition 2 **64732-97003, February 1993**

Using This Manual

This manual is designed to give you an introduction to the HP 64732 H8/510 Emulator. This manual will also help define how these emulators differ from other HP 64700 Emulators.

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 H8/510 Emulator

This manual will not:

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

Organization

Chapter 1 An introduction to the H8/510 emulator features and how they can help you in developing new hardware and software.

Chapter 2 A brief introduction to using the H8/510 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 Using a foreground monitor program; advantages and disadvantages.

Appendix B H8/510 Emulator Specific Command Syntax

Appendix C H8/510 Emulator Specific Error Messages

Table of Contents

Illustrations

Notes

1

Introduction to the H8/510 Emulator

Introduction

The topics in this chapter include:

- Purpose of the H8/510 Emulator
- Features of the H8/510 Emulator

Purpose of the H8/510 Emulator

The HP 64732 H8/510 mulator is designed to replace the H8/510 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 H8/510 emulator performs just like the H8/510 microprocessor, but is a device that allows you to control the H8/510 directly. These features allow you to easily debug software before any hardware is available, and ease the task of integrating hardware and software.

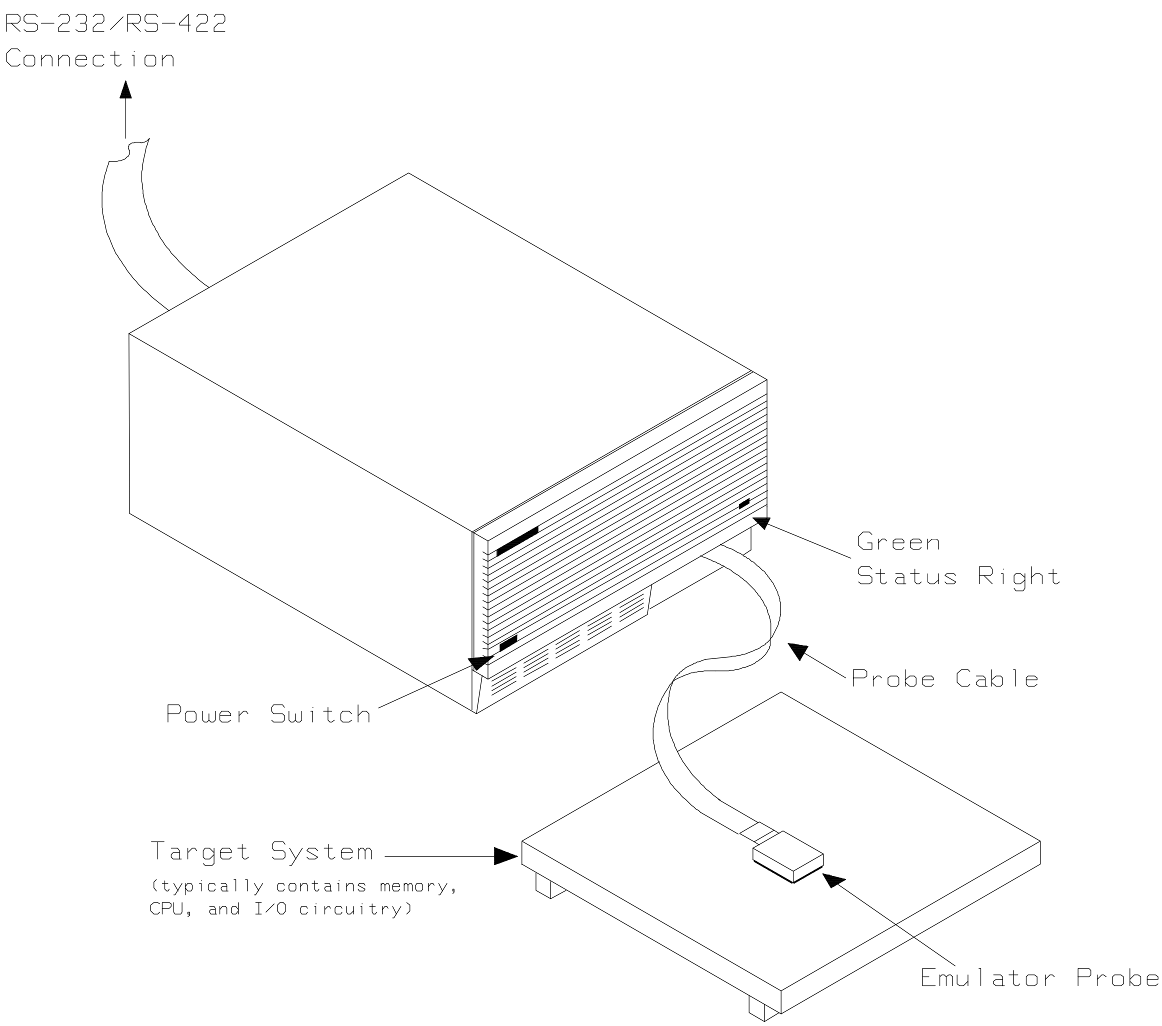


Figure 1-1. HP 64732 Emulator for the H8/510 Processor

Features of the H8/510 Emulator

Supported Microprocessors

HITACHI HD6415108F (H8/510) microprocessor is supported. An adaptor is provided to connect the emulator probe to your target system.

Clock Speeds

Maximum clock speed is 10 MHz (system clock).

Emulation memory

The HP 64732 H8/510 emulator is used with one of the following Emualtion Memory Cards.

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

The emulation memory can be configured into 256 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 H8/510 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 HP 64732 H8/510 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

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 H8/510 internal register contents. This includes the ability to modify the program counter (PC) and the code page register (CP) values so you can control where the emulator starts a program run.

Single-Step

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

Target System Interface

You can set the interface to the target system to be active or passive during background monitor operation.

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 (1B 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 any memory, display or modification of registers, and single step.

Easy Products Upgrades

Because the HP 64700 Series development tools (emulator, analyzer, LAN board) contain programmable parts, it is possible to reprogram the firmware and some of the hardware without disassembling the HP 64700A Card Cage. This means that you'll be able to update product firmware, if desired, without having to call an HP file representative to your site.

Limitations, Restrictions

DMA Support

Direct memory access to emulation memory is not allowed.

Monitor Break at Sleep/Standby Mode

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

Watch Dog Timer in Background

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

Reset Output Enable Bit

The RSTOE (Reset output enable bit) is used to determine whether the H8/510 processor outputs reset signal when the processor is reset by the watchdog timer. However, the HP 64732 emulator ignores the configuration of the RSTOE, and works as it is configured with **cf drst** command.

Notes

2

Getting Started

Introduction

This chapter will lead you through a basic, step by step tutorial designed to familiarize you with the use of the HP 64700 emulator for the H8/510 microprocessor. 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

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. If you are still unable to get a command prompt, refer to the *HP 64700 Support Services Guide*. The guide gives basic troubleshooting procedures. If this fails, call the local HP sales and service office listed in the *Support Services Guide.*

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 figure 2-1. The program emulates a primitive command interpreter.

Data Declarations

INPUT_POINTER and OUTPUT_POINTER define the address locations of an input area and an output area to be used by the program. MESSAGE_A, MESSAGE_B and INVALID_INPUT are the messages used by the program to respond to various command inputs.

Initialization

The locations of the input and output areas are moved into address registers for use by the program. Next, the CLEAR routine clears the command byte (the first byte location pointed to by the input area address - fc00 hex).

READ_INPUT

This routine continuously reads the byte at location fc00 hex until it is something other than a null character (00 hexadecimal); when this occurs, the PROCESS_COMM routine is executed.

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 R6 register and the base address of the message in the data area is moved to address register R0.

```
1000                                           .SECTION       SAMPDATA,DATA,LOCATE=H'1000
1000  FC00                  INPUT_POINTER      .DATA.W        H'FC00
1002  FD00                  OUTPUT_POINTER     .DATA.W        H'FD00

1004  5448495320495320      MESSAGE_A          .SDATA         "THIS IS MESSAGE A"
100C  4D45535341474520
1014  41
1015  5448495320495320      MESSAGE_B          .SDATA         "THIS IS MESSAGE B"
101D  4D45535341474520
1025  42
1026  494E56414C494420      INVALID_INPUT      .SDATA         "INVALID COMMAND"
102E  434F4D4D414E44

2000                                           .SECTION       SAMPPROG,CODE,LOCATE=H'2000
2000  5FFE40                INIT               MOV.W          #STACK,R7
2003  1D100082                                 MOV.W          @INPUT_POINTER,R2
2007  1D100283                                 MOV.W          @OUTPUT_POINTER,R3

200B  D20600                CLEAR              MOV.B          #H'00,@R2

200E  D20400                READ_INPUT         CMP.B          #H'00,@R2
2011  27FB                                     BEQ            READ_INPUT

013  D20441                 PROCESS_COMM       CMP.B          #H'41,@R2
2016  2707                                     BEQ            COMMAND_A
2018  D20442                                   CMP.B          #H'42,@R2
201B  2709                                     BEQ            COMMAND_B
201D  200E                                     BRA            UNRECOGNIZED

201F  5611                  COMMAND_A          MOV.B          #H'11,R6
2021  581004                                   MOV.W          #MESSAGE_A,R0
2024  200C                                     BRA            OUTPUT

2026  5611                  COMMAND_B          MOV.B          #H'11,R6
2028  581015                                   MOV.W          #MESSAGE_B,R0
202B  2005                                     BRA            OUTPUT

202D  560F                  UNRECOGNIZED       MOV.B          #H'0F,R6
202F  581026                                   MOV.W          #INVALID_INPUT,R0

2032  AB81                  OUTPUT             MOV.W          R3,R1
2034  5D0020                CLEAR_OLD          MOV.W          #H'0020,R5

2037  C10600                CLEAR_LOOP         MOV.B          #H'00,@R1+
203A  AD0C                                     ADD            #-1,R5
203C  26F9                                     BNE            CLEAR_LOOP

203E  A381                                     MOV.B          R3,R1
2040  C085                  OUTPUT_LOOP        MOV.B          @R0+,R5
2042  C195                                     MOV.B          R5,@R1+
2044  AE0C                                     ADD            #-1,R6
2046  26F8                                     BNE            OUTPUT_LOOP
2048  20C1                                     BRA            CLEAR

FE00                                           .SECTION       STACKAREA,STACK,LOCATE=H'FE00
FE00  0040                                     .RES.B         H'40
FE40                        STACK

                                               .END
```

Figure 2-1. Sample Program Listing

OUTPUT

First the base address of the output area is copied to R1 (this preserves R3 for use in later program passes). 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 sysmbol 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 ba=en
cf clk=int
cf dbc=en
cf drst=dis
cf mode=ext
cf mon=bg
cf nmi=en
cf rrt=dis
cf rsp=9
cf tbusrel=en
cf trfsh=en
cf trst=en
```

Note

The individual configuration items won't be explained in this example; refer to Chapter 4 of this manual and the *User's Reference* manual for details.

2. 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 ba=en
cf clk=ext
cf dbc=en
cf drst=dis
cf mode=ext
cf mon=bg
cf nmi=en
cf rrt=en
cf rsp=9
cf tbusrel=en
cf trfsh=en
cf trst=en
```

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

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

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

R> **cf rsp=0fe40**

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

R> **bc**

You will see:

```
bc -d bp #disable
bc -d 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 configure as you desire. You can define emulation memory as emulation RAM, emulation ROM, target RAM, target ROM or guarded memory. For this example, map the address 0 hex through 2fff hex as emulation ROM, and fc00 hex through feff hex as emulation RAM.

Type:

R> **map 0..2fff erom**
R> **map 0fc00..0feff eram**

To verify that memory blocks are mapped properly, type:

R> **map**

You will see:

```
# remaining number of terms   : 14
# remaining emulation memory  : 1c500h bytes
map  000000..002fff     erom   # term 1
map  00fc00..00feff     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 1000..100f=0fc,00,0fd,00,54,48,49,53,20,49,53,20,4d,45,53,53
R m 1010..101f=41,47,45,20,41,54,48,49,53,20,49,53,20,4d,45,53
R> m 1020..102f=53,41,47,45,20,42,49,4e,56,41,4c,49,44,20,43,4f
R> m 1030..1034=4d,4d,41,4e,44
```

   (note the hex letters must be preceded by a digit)

   You could also type the following line instead:

```
R> m 1000=0fc,00,0fd,00,"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 1000..1034
```

   You should see:

```
001000..00100f      fc 00 fd 00 54 48 49 53 20 49 53 20 4d 45 53 53
001010..00101f      41 47 45 20 41 54 48 49 53 20 49 53 20 4d 45 53
001020..00102f      53 41 47 45 20 42 49 4e 56 41 4c 49 44 20 43 4f
001030..001034      4d 4d 41 4e 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 1000..100f shows these values:

```
001000..00100f     0fc 00 00 0fd 00 54 48 49 53 20 49 53 20 4d 45 53
```

you can correct this row of memory by typing:

```
R> m 1000..100f=0fc,00,0fd,00,54,48,49,53,20,49,53,20,4d,45,53,53
```

Or, you might need to modify only one location, as in the instance where address 100f equals 55 hex rather than 53 hex. Type:

```
R> m 100f=53
```

3. Enter the program information by typing the following commands:

```
M> m 2000..200f=5f,0fe,40,1d,10,00,82,1d,10,02,83,0d2,06,00,0d2,04
M> m 2010..201f=00,27,0fb,0d2,04,41,27,07,0d2,04,42,27,09,20,0e,56
M> m 2020..202f=11,58,10,04,20,0c,56,11,58,10,15,20,05,56,0f,58
M> m 2030..203f=10,26,0ab,81,5d,00,20,0c1,06,00,0ad,0c,26,0f9,0a3,81
M> m 2040..2049=0c0,85,0c1,95,0ae,0c,26,0f8,20,0c1
```

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

```
R> m 2000..2049
```

You should see:

```
002000..00200f     5f fe 40 1d 10 00 82 1d 10 02 83 d2 06 00 d2 04
002010..00201f     00 27 fb d2 04 41 27 07 d2 04 42 27 09 20 0e 56
002020..00202f     11 58 10 04 20 0c 56 11 58 10 15 20 05 56 0f 58
002030..00203f     10 26 ab 81 5d 00 20 c1 06 00 ad 0c 26 f9 a3 81
002040..002049     c0 85 c1 95 ae 0c 26 f8 20 c1
```

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 64869 H8/500 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 64869 H8/500 Assembler/Linkage Editor, you may be able to adapt the methods below to load your code into the emulator (refer to the *HP 64700 User's 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 *User's 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.

$ **h8asm 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 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 64732 Softkey Interface Users' Guide* for more details.

$ **h8cnvhp -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 1000..1034**

You should see:

```
001000..00100f      fc 00 fd 00 54 48 49 53 20 49 53 20 4d 45 53 53
001010..00101f      41 47 45 20 41 54 48 49 53 20 49 53 20 4d 45 53
001020..00102f      53 41 47 45 20 42 49 4e 56 41 4c 49 44 20 43 4f
001030..001034      4d 4d 41 4e 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 2000..2049**

You will see:

```
002000  -          MOV:I.W #fe40,R7
002003  -          MOV:G.W @1000,R2
002007  -          MOV:G.W @1002,R3
00200b  -          MOV:G.B #00,@R2
00200e  -          CMP:G.B #00,@R2
002011  -          BEQ 0200e
002013  -          CMP:G.B #41,@R2
002016  -          BEQ 0201f
002018  -          CMP:G.B #42,@R2
00201b  -          BEQ 02026
00201d  -          BRA 0202d
00201f  -          MOV:E.B #11,R6
002021  -          MOV:I.W #1004,R0
002024  -          BRA 02032
002026  -          MOV:E.B #11,R6
002028  -          MOV:I.W #1015,R0
00202b  -          BRA 02032
00202d  -          MOV:E.B #0f,R6
00202f  -          MOV:I.W #1026,R0
002032  -          MOV:G.W R3,R1
002034  -          MOV:I.W #0020,R5
002037  -          MOV:G.B #00,@R1+
00203a  -          ADD:Q.W #-1,R5
00203c  -          BNE 02037
00203e  -          MOV:G.B R3,R1
002040  -          MOV:G.B @R0+,R5
002042  -          MOV:G.B R5,@R1+
002044  -          ADD:Q.W #-1,R6
002046  -          BNE 02040
002048  -          BRA 0200b
```

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 2000
```

You will see:

```
U>
```

The **r** command runs the processor from address 2000 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 2000**

The emulator changes state from background to foreground and begins running the sample program from location 2000 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 Tutorials chapter of this manual or the *HP 64700 User's 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=2011 cp=00 sr=0704 dp=00 ep=00 tp=00 br=00 r0=0000 r1=0000 r2=fc00
reg r3=fd00 r4=0000 r5=0000 r6=0000 r7=fe40 fp=0000 sp=fe40 mdcr=c1
```

Notice that R2 contains fc00 hex; R3 contains fd00 hex.

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

   Type:

```
U> m -db 0fc00
```

   You will see:

```
00fc00..00fc00     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 0fc00=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 0fd00..0fd1f
```

   You will see:

```
00FD00..00FD0f     54 48 49 53 20 49 53 20 4d 45 53 53 41 47 45 20
00FD10..00FD1f     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 fc00 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 2000;reg**

This command steps 1 instruction from address 2000 hex, and displays registers. You will see:

```
002000  -                 MOV:I.W #fe40,R7
PC = 002003
reg pc=2003 cp=00 sr=0708 dp=00 ep=00 tp=00 br=00 r0=1035 r1=fd0f r2=fc00
reg r3=fd00 r4=0000 r5=0044 r6=0000 r7=fe40 fp=0000 sp=fe40 mdcr=c1
```

Notice that PC contains 2003 hex.

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

M> **s;reg**

You will see:

```
002003  -                 MOV:G.W @1000,R2
PC = 002007
reg pc=2007 cp=00 sr=0708 dp=00 ep=00 tp=00 br=00 r0=1035 r1=fd0f r2=fc00
reg r3=fd00 r4=0000 r5=0044 r6=0000 r7=fe40 fp=0000 sp=fe40 mdcr=c1
```

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 analyzer begins writing data to the message output area.You can do this by specifying analyzer trigger upon encountering the address fd00 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 *User's Reference.*

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

M> **tg addr=0fd00**

To store only the accesses to the address range fd00 through fd11 hex, type:

M> **tsto addr=0fd00..0fd11**

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 2000**

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 fd00 hex). Type:

U> **m 0fc00=41**

To display the trace list, type:

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

You will see:

```
 Line  addr,H  data,A    count,R  seq
-----  ------  ------  ---------  ---
    0  00fd00  ..            ---   +
    1  00fd01  ..       10.00 uS   .
    2  00fd02  ..       10.00 uS   .
    3  00fd03  ..       10.00 uS   .
    4  00fd04  ..       10.00 uS   .
    5  00fd05  ..       10.00 uS   .
    6  00fd06  ..       10.60 uS   .
    7  00fd07  ..       10.00 uS   .
    8  00fd08  ..       10.00 uS   .
    9  00fd09  ..       10.00 uS   .
   10  00fd0a  ..       10.00 uS   .
   11  00fd0b  ..       10.00 uS   .
   12  00fd0c  ..       10.60 uS   .
   13  00fd0d  ..       10.00 uS   .
   14  00fd0e  ..       10.00 uS   .
   15  00fd0f  ..       10.00 uS   .
   16  00fd10  ..       10.00 uS   .
   17  00fd11  ..       10.00 uS   .
   18  00fd00  T.       152.5 uS   .
   19  00fd01  H.       12.12 uS   .
   20  00fd02  I.       11.48 uS   .
   21  00fd03  S.       12.12 uS   .
   22  00fd04  ..       12.08 uS   .
   23  00fd05  I.       12.12 uS   .
   24  00fd06  S.       11.48 uS   .
   25  00fd07  ..       12.12 uS   .
   26  00fd08  M.       12.08 uS   .
   27  00fd09  E.       11.52 uS   .
   28  00fd0a  S.       12.08 uS   .
   29  00fd0b  S.       12.12 uS   .
   30  00fd0c  A.       12.08 uS   .
   31  00fd0d  G.       11.52 uS   .
   32  00fd0e  E.       12.08 uS   .
   33  00fd0f  ..       12.12 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   00fd00  ..            ---   +
    1   00fd01  ..       10.00 uS   .
    2   00fd02  ..       10.00 uS   .
    3   00fd03  ..       10.00 uS   .
    4   00fd04  ..       10.00 uS   .
    5   00fd05  ..       10.00 uS   .
    6   00fd06  ..       10.60 uS   .
    7   00fd07  ..       10.00 uS   .
    8   00fd08  ..       10.00 uS   .
    9   00fd09  ..       10.00 uS   .
   10   00fd0a  ..       10.00 uS   .
   11   00fd0b  ..       10.00 uS   .
   12   00fd0c  ..       10.60 uS   .
   13   00fd0d  ..       10.00 uS   .
   14   00fd0e  ..       10.00 uS   .
   15   00fd0f  ..       10.00 uS   .
   16   00fd10  ..       10.00 uS   .
   17   00fd11  ..       10.00 uS   .
   18   00fd00  T.       152.5 uS   .
   19   00fd01  H.       12.12 uS   .
   20   00fd02  I.       11.48 uS   .
   21   00fd03  S.       12.12 uS   .
   22   00fd04  ..       12.08 uS   .
   23   00fd05  I.       12.12 uS   .
   24   00fd06  S.       11.48 uS   .
   25   00fd07  ..       12.12 uS   .
   26   00fd08  M.       12.08 uS   .
   27   00fd09  E.       11.52 uS   .
   28   00fd0a  S.       12.08 uS   .
   29   00fd0b  S.       12.12 uS   .
   30   00fd0c  A.       12.08 uS   .
   31   00fd0d  G.       11.52 uS   .
   32   00fd0e  E.       12.08 uS   .
   33   00fd0f  ..       12.12 uS   .
   34   00fd10  A.       12.08 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 (1B 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 User's 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_LOOP label.

M> **bp 2032**

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

M> **r 2000**

U> **m 0fc00=41**

```
!ASYNC_STAT  615! Software break point: 002032
```

M> **reg**

```
reg pc=2032 cp=00 sr=0700 dp=00 ep=00 tp=00 br=00 r0=1004 r1=fd11 r2=fc00
reg r3=fd00 r4=0000 r5=0041 r6=0011 r7=fe40 fp=0011 sp=fe40 mdcr=c1
```

Notice that PC contains 2032.

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 002032 #disabled
```

M> **bp -e 2032**

M> **bp**

```
###BREAKPOINT FEATURE IS ENABLED###
bp 002032 #enabled
```

M> **r 2000**

U> **m 0fc00=41**

```
!ASYNC_STAT  615! Software breakpoint: 002032
```

M> **bp**

```
###BREAKPOINT FEATURE IS ENABLED###
bp 002032 #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:  001004
```

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 27 hex, you can search for that value by typing:

M> **ser -db 2000..2049=27**

```
pattern match at address:  002011
pattern match at address:  002016
pattern match at address:  00201b
```

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 2000
M> m 0fc00=41
M> m 0fc00=42
M> m 0fc00=43
```

Make the coverage measurement:

```
U> cov 2000..2049
percentage of memory accessed:  % 100.0
```

You're now finished with the "Getting Started" example. You can proceed on with using the emulator and use this manual and the *HP 64700 Terminal Interface User's Reference* manual as needed to answer your questions.

Notes

3

Using the H8/510 Emulator In-Circuit

When you are ready to use the H8/510 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 H8/510 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 H8/510 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 adaptor 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 H8/510 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 H8/510 Emulator; when powering down, turn off the emulator first, then turn off power to the target system.

Pin Guard

HP 64732 H8/510 emulator is shipped with a non-conductive pin guard over the target system probe. This guard is designed to prevent impact damage to the pins and should be left in place while you are not using the emulator.

Target Sytem Adaptor

The HP 64732 emulator is shipped with a target system adaptor. The adaptor allows you to connect the emulation probe to your target system which is designed for the QFP package of H8/510 microprocessor.

Pin Protector

The HP 64732 emulator is shipped with a short pin protector that prevents damage to the target system adaptor when inserting and removing the emulation probe. **Do not** insert the probe without using a pin protector.

Installing the Target System Probe

1. Attach the adaptor to your target system. You can use a M2 screw to help attaching the adaptor to the target system.

2. Install the emulation probe using the short pin protector as shown in Figure 3-1.

Note

You can order additional target system adaptor and a short pin protector with part No. 64732-61613 and 64732-61614, respectively.

Note

You can use optional parts; a long pin protector and a pin extender to avoid conjunction with the emulation probe and target system components. Part No. are 64732-61615 and 64732-61616, respectively. Contact your local HP sales representative to purchase optional parts.

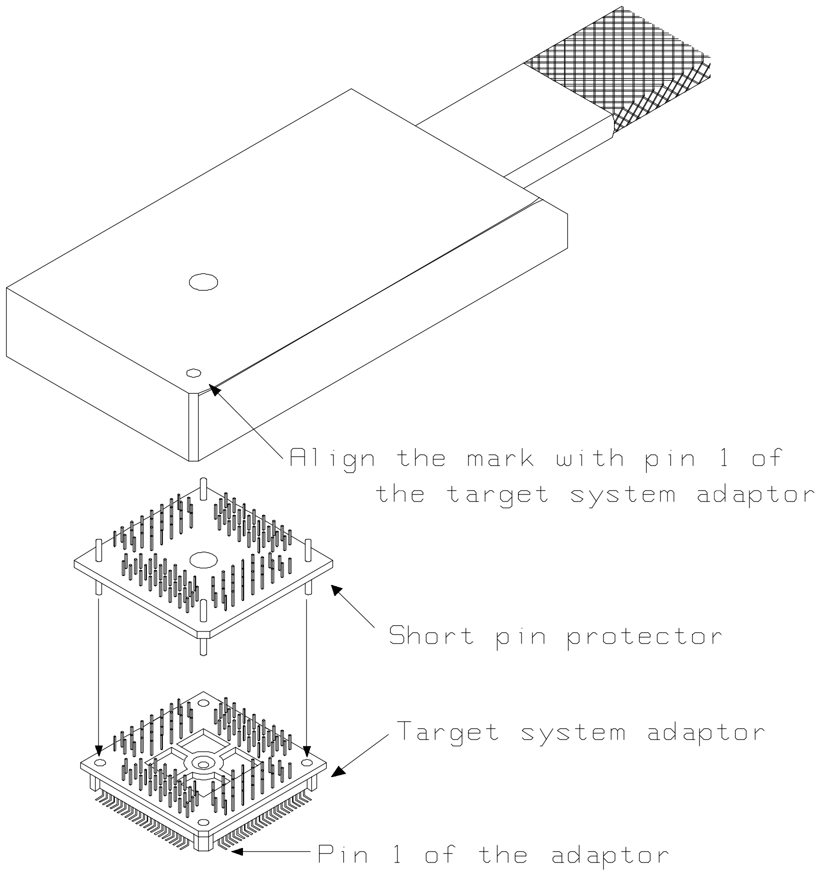


Figure 3-1. Installing Probe into the Target System

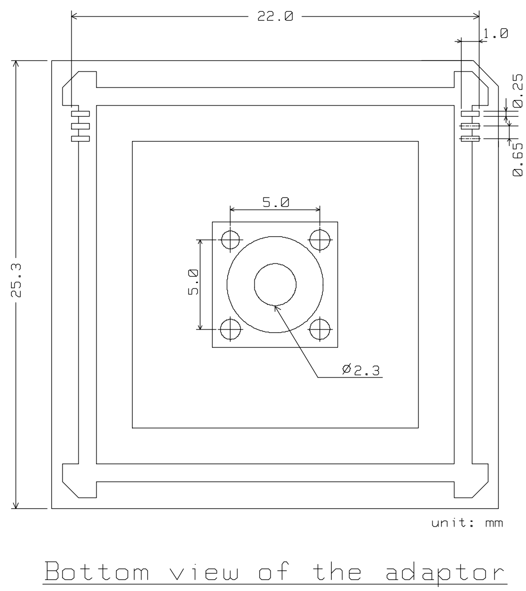
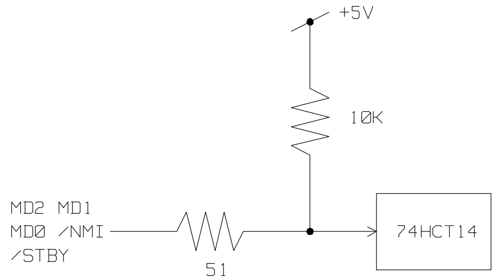


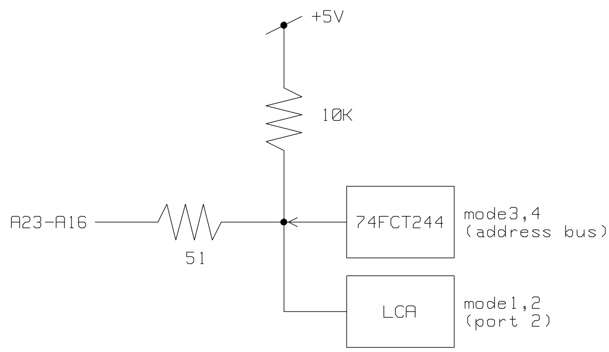
Figure 3-2. Dimension of the Target System Adaptor

Target System Interface

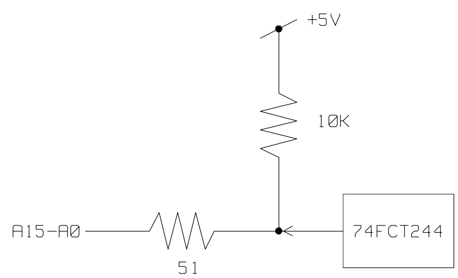
MD2 MD1 MD0 /NMI /STBY



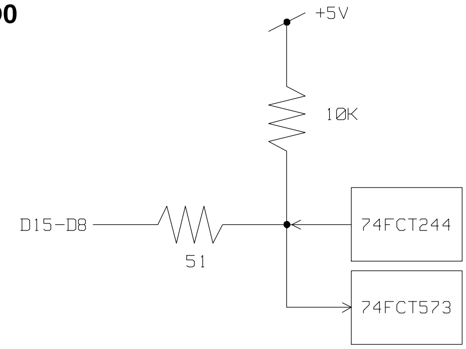
A23-A16



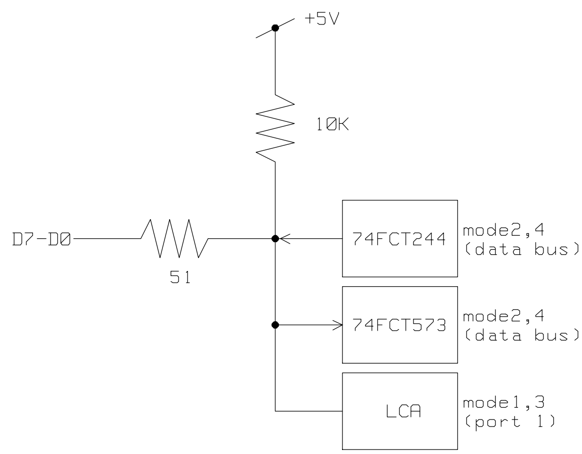
A15-A0



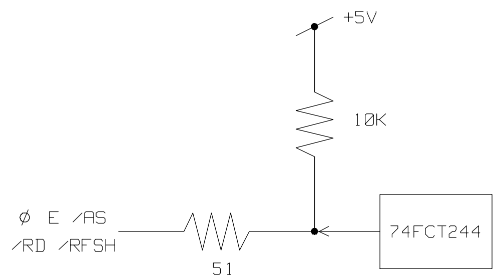
D15-D0



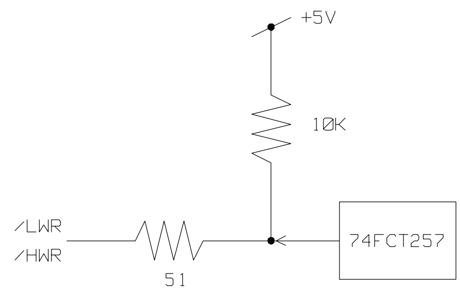
D7-D0



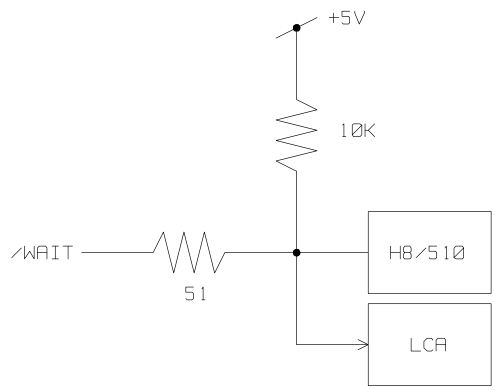
Q E /AS /RD /RFSH



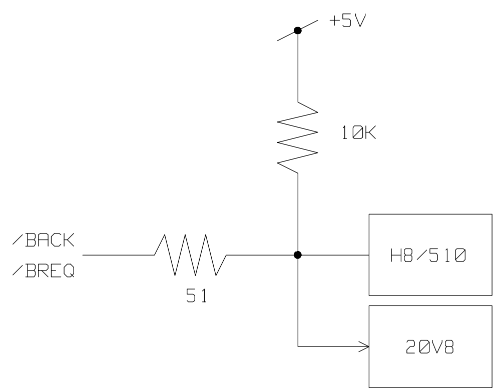
/LWR /HWR



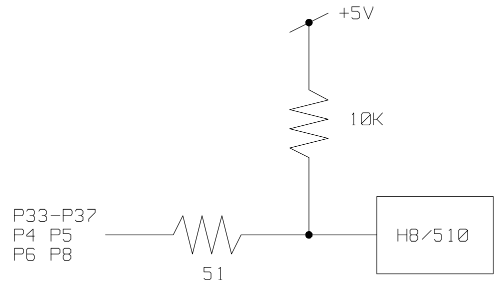
/WAIT



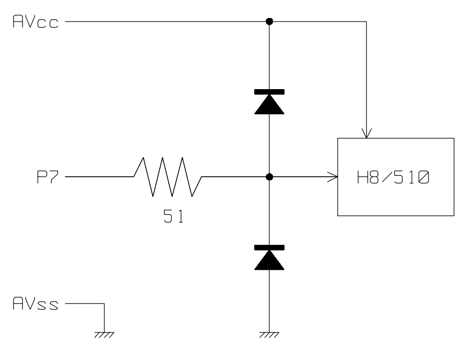
/BACK /BREQ



P33-P37 P4 P5 P6 P8



AVcc AVss



Notes

4

Configuring the H8/510 Emulator

In this chapter, we will discuss:

- how to configure the HP 64700 emulator for H8/510 microprocessor to fit your particular measurement needs.
- some restrictions of HP 64700 emulator for H8/510 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: User's 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: User's 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: User's 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 ba=en
cf clk=int
cf dbc=en
cf drst=dis
cf mode=ext
cf mon=bg
cf nmi=en
cf rrt=dis
cf rsp=9
cf tbusrel=en
cf trfsh=en
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 ba

The **ba** (bus arbitration) configuration item defines how your emulator responds to bus request signals from the target system during foreground operation. The /BREQ signal from the target system is always ignored when the emulator is running the background monitor.

```
M> cf ba=en
```

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

Note

External DMA (Direct Memory Access) device is prohibited from accessing to emulation memory.

```
M> cf ba=dis
```

When you disable bus arbitration by entering the above command, the emulator ignores the /BREQ signal from the target system. The emulation processor will never drive the /BACK line true; nor will it place the address, data and control signals into the tri-state mode.

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

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 H8/510 microprocessor.

Note

The HP 64732 H8/510 emulator can operate upto **10 MHz system clock**.

Note

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

cf dbc

The **dbc** (drive background cycles) option allows you to select whether or not the emulator will drive the target system bus on background cycles.

If you have selected to use a foreground monitor with the **cf mon= fg** command, emulator monitor cycles will appear at the target interface exactly as if they were user program cycles.

```
M> cf dbc=en
```

You can enable background cycle drive to target system by entering the above command. Emulation processor's address and control strobes (except /LWR and /HWR) are driven during background cycles.

Background write cycles won't appear to the target system. (/LWR and /HWR signals are always "high" when the **dbc** option is enabled.)

```
M> cf dbc=dis
```

If you specify the above command, background monitor cycles are not driven to the target system. When you select this option, the emulator will appear to the target system as if it continuously between bus cycles while it is operating in the background monitor.

You use the **dbc** option to avoid target system interaction problems. For example, your target system interaction scheme may depend on the constant repetition of bus cycles. In such case, using the **dbc** option will help avoid the problem.

Note Refresh cycles are always driven to the target system regardless of this configuration.

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

cf drst

The **drst** (drive reset) configuration item allows you to specify whether or not the mulator drives the /RES signal to the target system during emulation reset or reset by the Watchdog Timer.

```
M> cf drst=dis
```

The above command configures the emulator not to drive the reset signal to the target.

```
M> cf drst=en
```

The emulator will drive the reset signal to the target system during emulation or watchdog timer reset or reset by the Watchdog Timer.

To drive the reset signal, the emulator must be configured to respond to the target reset with the **cf trst= en** command.

Caution

To drive the reset signal to the target system, the driver of reset signal on your target system **must** be an open collector or open drain. Otherwise, enabling this configuration may result in damage to target system or emulation circuitry.

Note

The RSTOE (Reset output enable bit) is used to determine whether the H8/510 processor outputs reset signal when the processor is reset by the watchdog timer. However, the HP 64732 emulator ignores the configuration of the RSTOE, and works as it is configured with **cf drst** command.

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 input to MD0, MD1 and MD2. If you are using the emulator out of circuit when **ext** is selected, the emulator will operate in mode 1.

```
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 minimum mode with 8 bit data bus)
2	The emulator will operate in mode 2. (expanded minimum mode with 16 bit data bus)

3	The emulator will operate in mode 3. (expanded maximum mode with 8 bit data bus)
4	The emulator will operate in mode 4. (expanded maximum mode with 16 bit data bus)

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

cf mon

The **mon** (monitor) configuration item allows you to choose between a foreground monitor supplied by you or the background monitor supplied with the emulator.

The emulation monitor is the program that handles communication between the emulation controller and the emulation processor. For example, when you ask for a register display, the processor is broken to the monitor, executes some code to store its register contents in an array of memory locations, then returns to executing your program.

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

```
M> cf mon=bg
```

You select the use of the built-in background monitor through the above command. A memory overlay is created and the background monitor is loaded into that area.

```
M> cf mon=fg..XXXXXX
```

You select the use of your foreground monitor using this command.

XXXXXX defines an hexadecimal address where the monitor will be located. (Note: this will not load the monitor, it only specifies its location). You can define the location on a 2 kbyte boundary (address ending in 000 hex or 800 hex) of 000800 hex through 0fff800 hex. (When you are using the emulator in mode1 or mode2, 000800 hex through 00f000 hex is available.) 000000 hex and 00f800 hex are **not** available as the location.

Remember that you must assemble and link your foreground monitor starting at the 2 kbyte boundary specified in the command above. You must also load the monitor into emulation memory.

Note

If you intend to use a foreground monitor, the monitor must be loaded before attempting to load any information into target system memory.

Note

When you use the foreground monitor, the trace exception vector in the target system **must** point to TRACE_ENTRY in the foreground monitor to use single step command. (Refer to Appendix A of this manual.)

A memory mapper term is automatically created when you execute the **cf mon= fg** command to reserve 2 kilobytes of memory space for the monitor.

The memory map is reset any time **cf mon= fg** is entered. It is only reset when **cf mon= bg** if the emulator is not already configured to use the background monitor.

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.

If you are using the background monitor, the emulator does not accept any interrupt during background execution. NMI , /IRQ1, /IRQ2 and /IRQ3 are latched last one during in background, and such interrupts will occur when context is changed to foreground. /IRQ0 and internal interrupts are ignored during in background operation.

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 and stack page register will be set upon the transition from emulation reset into the emulation monitor.

```
R> cf rsp=XXXXXX
```

where **XXXXXX** is a 24-bit even address, will set the stack pointer and stack page register 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

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

For example, to set the stack pointer to 0fe00 hex, type:

```
R> cf rsp=0fe00
```

Now, if you break the emulator to monitor using the **b** command, the stack pointer will be modified to the value 0fe00 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 tbusrel

The **tbusrel** (trace bus release cycles) configuration item defines whether or not the emulator traces bus release cycles.

```
M> cf tbusrel=en
```

When you enable this item with the above command, each time bus release occurs, one emulation analyzer state will be generated to recognize the bus release cycle.

```
M> cf tbusrel=dis
```

When disabled, no analyzer state will be generated at the occurrence of bus release. Therefore, any bus release cycle will be ignored by the analyzer.

cf trfsh

The **trfsh** (trace refresh cycles) configuration item defines whether or not the emulator traces refresh cycles.

```
M> cf trfsh=en
```

When you enable this item with the above command, the analyzer will trace refresh cycles.

```
M> cf trfsh=dis
```

When disabled, the analyzer will ignore refresh cycles.

cf trst

The **trst** (target reset) configuration item allows you to specify whether or not the emulator responds to /RES and /STBY signals by the target system during foreground operation. While running the background monitor, the emulator ignores /RES and /STBY signals, 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 and /STBY input during foreground operation.

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

When disabled, the emulator won't respond to /RES and /STBY 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 HP 64732 H8/510 emulator memory mapper allows you to define up to 16 different map terms; each map term has a minimum size of 256 bytes. If you specify a value less than 256 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 2000 through 6fff 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 2000..6fff eram**

```
# remaining number of terms   : 14
# remaining emulation memory  : 1a800h bytes
map  002000..006fff     eram    # term 1
map  00f000..00f1ff     tram    # term 2
map  other tram
```

As you can see, the mapper rounded up the second term to 256 bytes block, since those are minimum size blocks supported by the H8/510 emulator.

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.

Watch Dog Timer in Background

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

Reset Output Enable Bit

The RSTOE (Reset output enable bit) is used to determine whether the H8/510 processor outputs reset signal when the processor is reset by the watchdog timer. However, the HP 64732 emulator ignores the configuration of the RSTOE, and works as it is configured with **cf drst** command.

Address Error and Register Values

In operation of the H8/510 microprocessor, the Stack Pointer must always contain an even value. If the Stack Pointer is odd, you will see the following error message when you breaks into the monitor.

```
Address error occurred while in monitor
```

In this case, the values of the following registers will be unreliable.

- Stack Pointer (SP)
- Code Page Register (CP)
- Status Register (SR)

A

Using the Optional Foreground Monitor

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

Comparison of Foreground and Background Monitors

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

Background Monitors

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

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

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

Foreground Monitors

A *foreground* monitor may be required for more complex debugging and integration applications. A foreground monitor is a block of code that runs in the same memory space as your program. You link this monitor with your code so that when control is passed to your program, the emulator can still service real-time events, such as interrupts or watchdog timers. For most multitasking, interrupt intensive applications, you will need to use a foreground monitor.

You can tailor the foreground monitor to meet your needs, such as servicing target system interrupts. However, the foreground monitor does use part of the processor's address space, which may cause problems in some target systems. You must also properly configure the emulator to use a foreground monitor (see Chapter 3 and the examples in this appendix); and, you must link the monitor with your other program code.

An Example Using the Foreground Monitor

In the following example, we will illustrate how to link a foreground monitor with the sample program from Chapter 2. By using the emulation analyzer, we will also show how the emulator switches from state to state using a foreground monitor.

Select A Monitor Suitable to Your Application

The H8/510 emulator is provided with two foreground monitor programs. When you are using the emulator in the minimum mode, use **fm510min.src** monitor program. When you are using the emulator in the maximum mode, use the **fm510max.src** monitor program.

For this example, we will use the **fm510min.src** monitor program, and will locate the monitor at 8000 hex; the sample program will be located at 2000 hex with its data at 1000 hex.

Modify Location Declaration Statement

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

```
LOCATE_ADRS:    .EQU    H’8000          ;start monitor on 2k boundary
 .SECTION fm510min,CODE,LOCATE=LOCATE_ADRS
```

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

```
LOCATE_ADRS:    .EQU    H’6000          ;start monitor on 2k boundary
 .SECTION fm510min,CODE,LOCATE=LOCATE_ADRS
```

Notice that the .SECTION statement is indented from the left margin; if it is not indented, the assembler will attempt to interpret the .SECTION as a label and will generate an error when processing the address portion of the statement. You can load the **fm510min.src** monitor on a 2k byte boundary of 00800 hex through 0f000 hex. When you are going to use **fm510max.src**, you can load it on a 2k byte boundary of 00800 hex through 0fff800 hex except 0f800 hex.

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

You can also specify monitor location when you link it. If you prefer this way, do the following:

- Change .EQU statement and .SECTION statement just after the first comment section into comment line by inserting ";" at the first column of these lines.
- Make the next two lines (.EQU statement and .SECTION statement which doesn't have LOCATE keyword) effective by deleting ";" at the first column of these lines.
- Specify the monitor location when you link the monitor program. To do this, you can use the START linker subcommand like this:

```
$h8lnk
:INPUT sampprog,fm510min
:OUTPUT testfg
:START fm510min(8000)
:EXIT
```

Configure the Emulator

Before configuring the emulator, you should initialize the emulator to a known state. Type:

```
M> init -p
```

You need to tell the emulator that you will be using a foreground monitor and allocate the memory space for the monitor. This is all done with one configuration command. To locate the monitor on a 2k boundary starting at 8000 hex, type:

```
R> cf mon=fg..8000
```

To see the new memory mapper term allocated for the foreground monitor, type:

```
R> map
```

```
# remaining number of terms   : 15
# remaining emulation memory  : 1f800h bytes
map  0008000..00087ff     eram   # term 1
map  other tram
```

Notice that a 2k byte block from 8000 through 87ff hex was mapped.

Now, you need to map memory space for the sample program. Type:

```
R> map 0..2fff erom
R> map 0fc00..0feff eram
```

Set a Stack Pointer

You need to set up the stack pointer for use by the foreground monitor. The foreground monitor use the stack when transit from foreground monitor to user program. You can use the **cf rsp** command to define the stack pointer location; the stack pointer will be initialized on each transition from emulation reset to the monitor. Type:

```
R> cf rsp=0fe40
```

Load the Program Code

Now it's time to load the sample program and monitor. Link the sample program provided in chapter 2 and monitor program into absolute file named testfg.abs, and covert it into HP format absolute file named testfg.X. In the example shown, we're loading the program from a host with the emulator in Transparent Configuration. If you're using the standalone configuration with a data terminal, you will need to enter the data using the **m** command. (You can get the data from your assembly listings.) See Chapter 2 for information.

Load the program by typing:

```
R> load -hbs "transfer -tb testfg.X"
```

############

You can also load the sample program and the monitor separately. In this case, you don't have to link the sample program with the monitor.

Before we forget, let's initialize the stack pointer by breaking the emulator out of reset:

```
R> b
```

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

```
M> r 2000
```

Single Step and Foreground Monitors

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

Limitations of Foreground Monitors

Synchronized measurements

You cannot perform synchronized measurements over the CMB when using a foreground monitor. If you need to make such measurements, set the foreground/background configuration option to **cf mon= bg**.

B

H8/510 Emulator Specific Command Syntax

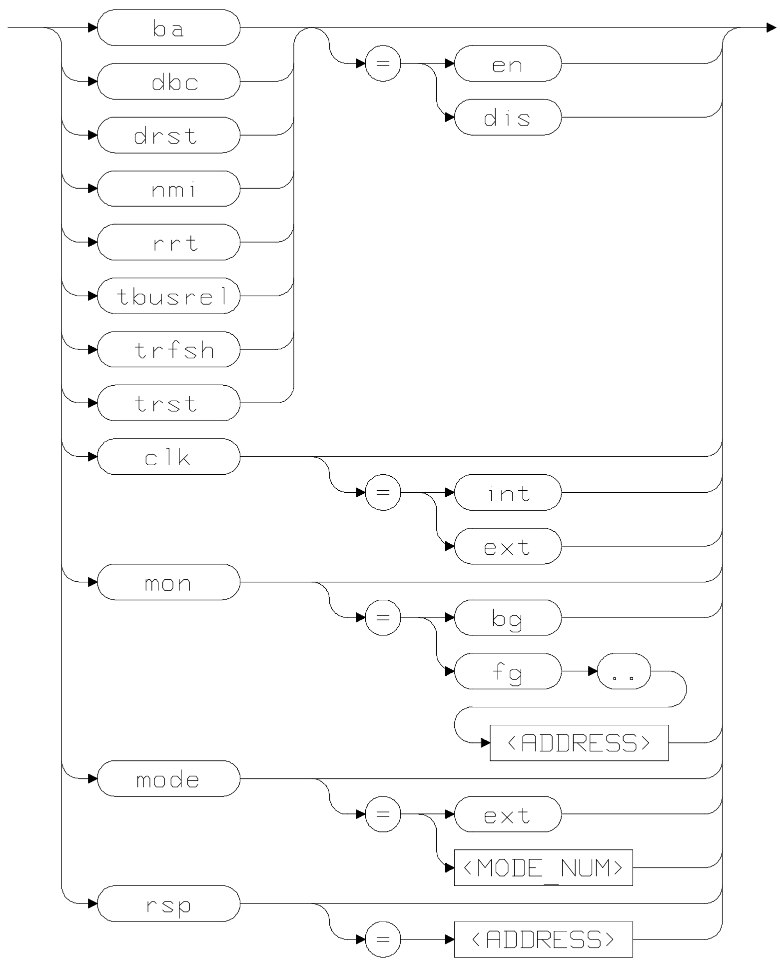
The following pages contain descriptions of command syntax specific to the H8/510 emulator. The following syntax items are included (several items are part of other command syntax):

- < 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.
- < Register Classes and Names> . May be specified in the **reg** (register) command.

CONFIG_ITEMS

Summary **H8/510 emulator configuration items.**

Syntax



Description

The H8/510 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:

ba	Enable/disable bus arbitration with target system.
clk	Select internal/external clock source.
dbc	Enable/disable to drive background cycles to target system.
drst	Enable/disable to drive emulation reset to target system.
mode	Determine emulator processor operation mode.
mon	Select background or foreground monitor.
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.
tbusrel	Enable/disable tracing bus release cycles.
trfsh	Enable/disable tracing refresh cycles.
trst	Enable/disable target system reset.

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 mon=fg..08000 rrt=dis clk
```

```
clk=int
```

Here, we changed to a foreground monitor located at address 8000 hex, disabled the real-time runs restriction, and ask processor clock source. Notice that items which are changed do not have status printed; you could explicitly request the new status by repeating the configuration item on the command line after the change but without a value. For example:

```
R> cf mon=fg..2000 mon
```

```
cf mon=fg..2000
```

Related information

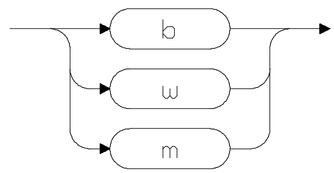
Refer to the **cf** syntax pages in the *User's Reference* manual. Also, refer to chapter 4 of this manual for complete information about each configuration item.

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.

Defaults

The **< DISPLAY_MODE>** and the **< ACCESS_MODE>** are **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.

Note

When the **< ACCESS_MODE>** is **w**, modifying target memory will fail if you try to modify memory from an odd address or with data which byte count is odd. Therefore, it is recommended to use the emlator with the default **< ACCESS_MODE>** (**b**).

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 24 bits address information.

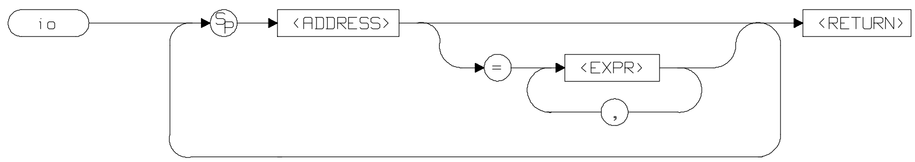
Examples

m 1000

m 20000..200ff

io Command

Syntax



Summary

The **io** command accesses devices on the target system with MOVFPE/MOVTPE instruction.

Restrictions

- The **io** command accesses target system memory regardless of memory mapping.
- Address range cannot be specified in **io** command.

Register Classes and Names

Summary

H8/510 register designators.

Description

The following register classes and names are used with the display/modify registers commands in H8/510 emulator.

* (Basic) Class

Register name	Description
pc	Program counter
cp	Code page register
sr	Status register
dp	Data page register
ep	Extended page register
tp	Stack page register
br	Base register
r0	Register R0
r1	Register R1
r2	Register R2
r3	Register R3
r4	Register R4
r5	Register R5
r6	Register R6
r7	Register R6
r7	Register R7
fp	Frame pointer
sp	Stack pointer
mdcr	Mode control register

sys Class

System control registers

Register name	Description
rfshcr	Refresh control register
wcr	Wait control register
arbt	Byte are top register
ar3t	3 state area top register
mdcr	Mode control register
sbycr	Software stand-by control register
brcr	Bus relaese control register

intc Class

Interrupt control registers

ipra	Interrupt priority register A
iprab	Interrupt priority register B
iprc	Interrupt priority register C
iprd	Interrupt priority register D
nmicr	NMI control register
irqcr	IRQ control register

dtc Class

Data transfer controller registers

dtea	DT enable register A
dteb	DT enable register B
dtec	DT enable register C
dted	DT enable register D

port Class

I/O port registers

Register name	Description
p1ddr	Port 1 data direction register
p2ddr	Port 2 data direction register
p3ddr	Port 3 data direction register
p4ddr	Port 4 data direction register
p5ddr	Port 5 data direction register
p6ddr	Port 6 data direction register
p8ddr	Port 8 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

frt1 Class

Free running timer 1 registers

frtcr1	Timer control register
frtcsr1	Timer control/status register
frc1	Free running counter
ocra1	Output compare register A
ocrb1	Output compare register B
icr1	Input capture register

frt2 Class

Free running timer 2 registers

Register name	Description
frtcr2	Timer control register
frtcsr2	Timer control/status register
frc2	Free running counter
ocra2	Output compare register A
ocrb2	Output compare register B
icr2	Input capture register

frt3 Class

Free running timer 3 registers

frtcr3	Timer control register
frtcsr3	Timer control/status register
frc3	Free running counter
ocra3	Output compare register A
ocrb3	Output compare register B
icr3	Input capture register

tmr Class

Timer registers

tcr	Timer control register
tcsr	Timer control/status register
tcora	Timer constant register A
tcorb	Timer constant register B
tcnt	Timer counter

wdt Class

Watchdog timer registers

wdtcsr	Timer control/status register
wdtcnt	Timer counter
rstcsr	Reset control/status register

sci1 Class

Serial communication interface 1 registers.

Register name	Description
rdr1	Receive data register
tdr1	Transmit data register
smr1	Serial mode register
scr1	Serial control register
ssr1	Serial status register
brr1	Bit rate register

sci2 Class

Serial communication interface 2 registers.

rdr2	Receive data register
tdr2	Transmit data register
smr2	Serial mode register
scr2	Serial control register
ssr2	Serial status register
brr2	Bit rate register

adc Class

A/D converter registers

addra	A/D data register A
addrb	A/D data register B
addrc	A/D data register D
addrd	A/D data register D
adcsr	A/D control/status register
adcr	A/D control register

C

H8/510 Emulator Specific Error Messages

The following pages document the error messages which are specific to the HP 64732 H8/510 emulator. The cause of the error is described, as well as the action you must take to remedy the situation.

Message

140 : Stack pointer not initialized

Cause

This error occurs when you attempt to execute user program (with **r** or **s** command) without set up the stack pointer. You will see this error message only when you are operating with the foreground monitor. When you are using the backgorund monitor, another error message will be reported.

Action

Set up the stack pointer with **cf rsp** command. Refer to chapter 3 of this manual for more information.

Message

141 : Stack is in I/O registers

Cause

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

Action

Set up the stack pointer at proper location with **cf rsp** command. Refer to chapter 3 of this manual for more information.

Message

144 : Invalid address for run or step in minimum mode

Cause

This error occurs when you attempt to run or step from address other than page 0 during the emulator operates in minimum mode.

Message

145 : Code page register not writable in minimum mode

Cause

This error occurs when you attempt to modify the Code Page Register during the emulator operates in miminum mode.

Message

147 : Address error occured while in monitor

Cause

This error occurs when you break into the monitor while the Stack Pointer contains an odd value.

Index

Notes