HP 64730

H8/570 Emulator Softkey Interface

User's Guide

HEWLETT PACKARD

HP Part No. 64730-97005
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.

Copyright 1992,1993, Hewlett-Packard Company.

This document contains proprietary information, which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another language without the prior written consent of Hewlett-Packard Company. The information contained in this document is subject to change without notice.

UNIX is a registered trademark of UNIX System Laboratories Inc. in the U.S.A. and other countries.

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

LCA is a trademark of Xilinx Inc.

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

RESTRICTED RIGHTS LEGEND Use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in subparagraph (C) (1) (ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013. Hewlett-Packard Company, 3000 Hanover Street, Palo Alto, CA 94304 U.S.A. Right for non-DOD U.S. Government Department and Agencies are as set forth in FAR 52.227-19(c)(1,2).

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	64730-97002, May 1992
Edition 2	64730-97005, February 1993

Using This Manual

This manual will show you how to use the HP 64730 H8/570 Emulator with the Softkey Interface. This manual will also help define how these emulators differ from other HP 64700 Emulators.

This manual will:

- Show you how to use emulation commands by executing them on a sample program and describing their results.
- Show you how to configure the emulator for your development needs. Topics include: restricting the emulator to real-time execution, and selecting a target system clock source.
- Show you how to use the emulator in-circuit (connected to a target system).

This manual will not:

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

Organization

Chapter 1 Introduction to the H8/570 Emulator. This chapter briefly introduces you to the concept of emulation and lists the basic features of the H8/570 emulator.

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

Chapter 3 Debugging ISP Functions. This chapter shows you how to use the emulator to debug your ISP functions. This chapter describes how to: load ISP functions into the emulator, display ISP memory, display ISP registers, step through ISP functions, run ISP functions, and use the analyzer.

Chapter 4 "In-Circuit" Emulation. This chapter shows you how to install the emulator probe into a target system and how to use the "in-circuit" emulation features.

Chapter 5 Configuring the Emulator. This chapter shows you how to restrict the emulator to real-time execution, select a target system clock source, allow background cycles to be seen by the target system.

Chapter 6 Using the Emulator. This chapter describes emulation topics which are not covered in the "Getting Started" chapter.

Appendix A H8/570 Softkey Interface Specific Syntax. This appendix describes specific syntax to the H8/570 Softkey Interface.

Conventions

Example commands throughout the manual use the following conventions:

bold Commands, options, and parts of command syntax.

bold italic Commands, options, and parts of command syntax which may be entered by pressing softkeys.

normal User specified parts of a command.

$ Represents the HP-UX prompt. Commands which follow the "$" are entered at the HP-UX prompt.

<RETURN> The carriage return key.

Notes

Contents

3 Debugging ISP Functions

4 In-Circuit Emulation

5 Configuring the Emulator

A H8/570 Softkey Interface Specific Syntax

Illustrations

Notes

1

Introduction to the H8/570 Emulator

Introduction

The topics in this chapter include:

- Purpose of the H8/570 emulator.
- Features of the H8/570 emulator.

Purpose of the H8/570 Emulator

The H8/570 emulator is designed to replace the H8/570 microprocessor in your target system to help you debug/integrate target system software and hardware. The emulator performs just like the processor which it replaces, but at the same time, it gives you information about the bus cycle operation of the processor. The emulator gives you control over target system execution and allows you to view or modify the contents of processor registers, target system memory.

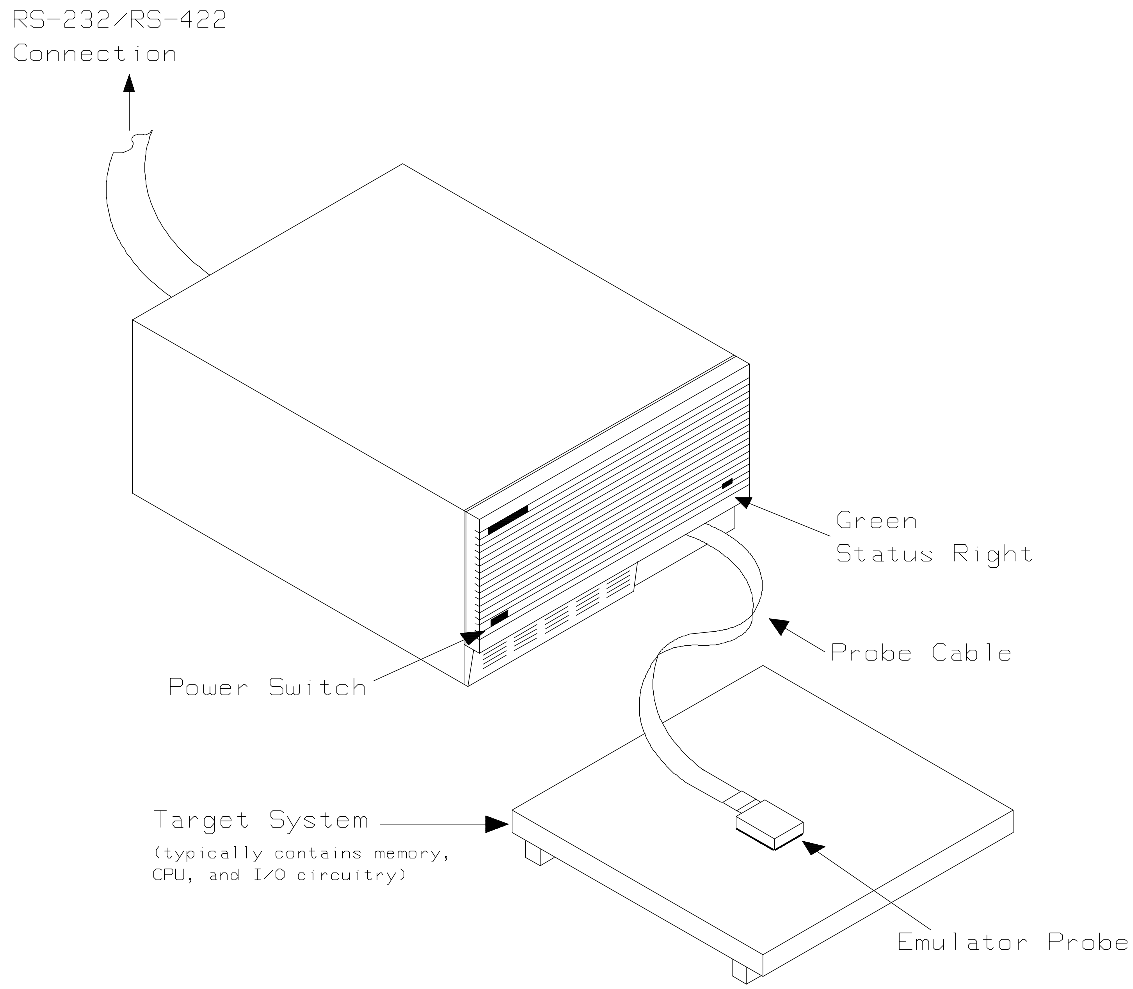


Figure 1-1. HP 64730 Emulator for the H8/570 Emulator

Features of the H8/570 Emulator

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

Supported Microprocessors

HITACHI HD6475708F (H8/570) microprocessor is supported.

Clock Speeds

Maximum external clock speed is 12 MHz (system clock). Internal clock of the emulator is 10 MHz.

Emulation memory

The HP 64730 H8/570 emulator is used with one of the following Emulation Memory Cards.

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

The emulator uses 4K bytes of emulation memory, and the rest of emulation memory is available for user program. You can define up to 15 memory ranges (at 128 byte boundaries and at least 128 byte in length). You can characterize memory ranges as emulation RAM, emulation ROM, target system RAM, target system ROM, or as guarded memory. The emulator generates an error message when accesses are made to guarded memory locations. You can also configure the emulator so that writes to memory defined as ROM cause emulator execution to break out of target program execution.

Analysis

The HP 64730 H8/570 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 H8/570 internal register contents. This includes the ability to modify the program counter (PC) and code page register (CP) so you can control where the emulator begins executing a target system program.

Single-Step

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

Target System Interface

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

Breakpoints

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

You can also define software breakpoints in your program. The emulator uses one of H8/570 undefined opcode (1B hex) as software breakpoint interrupt instruction. When you define a software breakpoint, the emulator places the breakpoint interrupt instruction (1B hex) at the specified address; after the breakpoint interrupt instruction causes emulator execution to break out of your program, the emulator replaces the original opcode. Refer to the "Using Software Breakpoints" section of "Getting Started" chapter for more information.

Reset Support

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

Real-Time Execution

Real-time execution signifies continuous execution of your program without interference from the emulator. (Such interference occurs when the emulator temporarily breaks into the monitor so that it can access register contents or target system memory.) Emulator features performed in real time include: running and analyzer tracing.

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

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.

Features for ISP debug

The ISP (Intelligent Subprocessor) is a programmable internal peripheral device of the H8/570 processor. The HP 64730A emulator provides useful features to debug ISP functions.

ISP Function Load

You can load your ISP functions into the microprogram memory and SCM (Sequence Control Matrix) of the emulator.

Execution Control

You can direct the ISP to run, halt, or execute a specified number of instructions.

Memory Display

You can display the contents of ISP microprogram memory in mnemonic format.

Register Display

You can display/modify the contents of H8/570 ISP registers.

Analysis

You can direct the emulator to monitor the execution of CPU program or ISP functions, or both of them.

Limitations, Restrictions

DMA Support

Direct memory access to H8/570 emulation memory is not permitted.

Sleep and Software Stand-by Mode

When the emulator breaks into the emulation monitor, H8/570 microprocessor sleep or software stand-by mode is released and comes to normal processor mode.

Watch Dog Timer in Background

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

ISP Microprogram Modify

The contents of ISP microprogram memory cannot be modified by emulation commands. To modify your ISP program, you need to re-assemble/link your program, and load it into the emulator.

Symbolic Information for ISP Functions

The H8/570 Softkey Interface does not support symbolic information for ISP functions. No symbolic information for ISP functions is dispalyed in ISP memory display and trace listing.

RAM Enable Bit

The internal RAM of H8/510 processor can be enabled/disabled by RAME (RAM enable bit). However, the H8/570 emulator 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 HP 64730 emulator with the Softkey Interface.

This chapter will:

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

This chapter will show you how to:

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

Before You Begin

Prerequisites

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

1. Connected the emulator to your computer. The *HP 64700 Series Installation/Service* manual show you how to do this.

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

3. In addition, you should read and understand the concepts of emulation presented in the *Concepts of Emulation and Analysis* manual. The *Installation/Service* manual also covers HP 64700 system architecture. A brief understanding of these concepts may help avoid questions later.

   You should read the *Softkey Interface Reference* manual to learn how to use the Softkey Interface in general. For the most part, this manual contains information specific to the H8/570 emulator.

A Look at the Sample Program

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

/usr/hp64000/demo/emul/hp64730/cmd_rds.src

Data Declarations

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

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

WCR             .EQU        H’FF48

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


                .SECTION    Prog,CODE
;*****************************************************
;* Sets up the stack pointer and the Wait-state
;* controller.
;*****************************************************
Init            MOV.W       #Stack,R7
                MOV.B       #H’f0,@WCR
;*****************************************************
;* Clear previous command.
;*****************************************************
Read_Cmd        MOV.B       #0,@Cmd_Input
;*****************************************************
;* Read command input byte.  If no command has
;* been entered, continue to scan for input.
;*****************************************************
Scan            MOV.B       @Cmd_Input,R0
                BEQ         Scan
;*****************************************************
;* A command has been entered.  Check if it is
;* command A, command B, or invalid.
;*****************************************************
Exe_Cmd         CMP.B       #H’41,R0
                BEQ         Cmd_A
                CMP.B       #H’42,R0
                BEQ         Cmd_B
                BRA         Cmd_I
;*****************************************************
;* Command A is entered.  R1 = the number of
;* bytes in message A.  R4 = location of the
;* message.  Jump to the routine which writes
;* the messages.
;*****************************************************
Cmd_A           MOV.W       #Msg_B-Msg_A-1,R1
                MOV.W       #Msg_A,R4
                BRA         Write_Msg
;*****************************************************
;* Command B is entered.
;*****************************************************
Cmd_B           MOV.W       #Msg_I-Msg_B-1,R1
                MOV.W       #Msg_B,R4
                BRA         Write_Msg
;*****************************************************
;* An invalid command is entered.
;*****************************************************
```

Figure 2-1. Sample Program Listing

```
Cmd_I           MOV.W       #End_Msgs-Msg_I-1,R1
                MOV.W       #Msg_I,R4
;*****************************************************
;* Message is written to the destination.
;*****************************************************
Write_Msg       MOV.W       #Msg_Dest,R5
Again           MOV.B       @R4+,R3
                MOV.B       R3,@R5+
                SCB/EQ      R1,Again
;*****************************************************
;* The rest of the destination area is filled
;* with zeros.
;*****************************************************
Fill_Dest       MOV.B       #0,@R5+
                CMP.W       #Msg_Dest+H'20,R5
                BNE         Fill_Dest
;*****************************************************
;* Go back and scan for next command.
;*****************************************************
                BRA         Read_Cmd

                .SECTION    Data,COMMON
;*****************************************************
;* Command input byte.
;*****************************************************
Cmd_Input       .RES.B      H'1
                .RES.B      H'1
;*****************************************************
;* Destination of the command messages.
;*****************************************************
Msg_Dest        .RES.B      H'3E
                .RES.W      H'80       ; Stack area.
Stack

                .END        Init
```

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

Initialization

The program instructions at the **Init** label initializes the stack pointer and the wait state controller.

Reading Input

The instruction at the **Read_Cmd** label clears any random data or previous commands from the **Cmd_Input** byte. The **Scan** loop continually reads the **Cmd_Input** byte to see if a command is entered (a value other than 0 hex).

Processing Commands

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

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

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

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

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

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

The Destination Area

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

This program emulates a primitive command interpreter.

Sample Program Assembly

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

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

Linking the Sample Program

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

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

```
debug
input cmd_rds
start Prog(1000), Table(2000), Data(0FC00)
outpur cmd_rds
print cmd_rds
exit
```

Figure 2-2. Linkage Editor Subcommand File

Generate HP Absolute file

To generate HP Absolute file for the Softkey Interface, you need to use **"h8cnvhp"**absolute file format converter program. To generate HP Absolute file, enter following command:

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

You will see that cmd_rds.X, cmd_rds.L, and cmd_rds.A are generated.

Refer to Chapter 6 of this manual for more detail of **h8cnvhp** converter.

Note

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

Entering the Softkey Interface

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

From the "pmon" User Interface

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

```
$ pmon <RETURN>
```

If you have not already created a measurement system for the H8/570 emulator, you can do so with the following commands. First you must initialize the measurement system with the following command.

```
MEAS_SYS msinit <RETURN>
```

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

```
msconfig <RETURN>
```

To define a measurement system for the H8/570 emulator, enter:

```
make_sys emh8 <RETURN>
```

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

```
add <module_number> naming_it h8 <RETURN>
```

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

```
end <RETURN>
```

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

```
emh8 default h8 <RETURN>
```

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

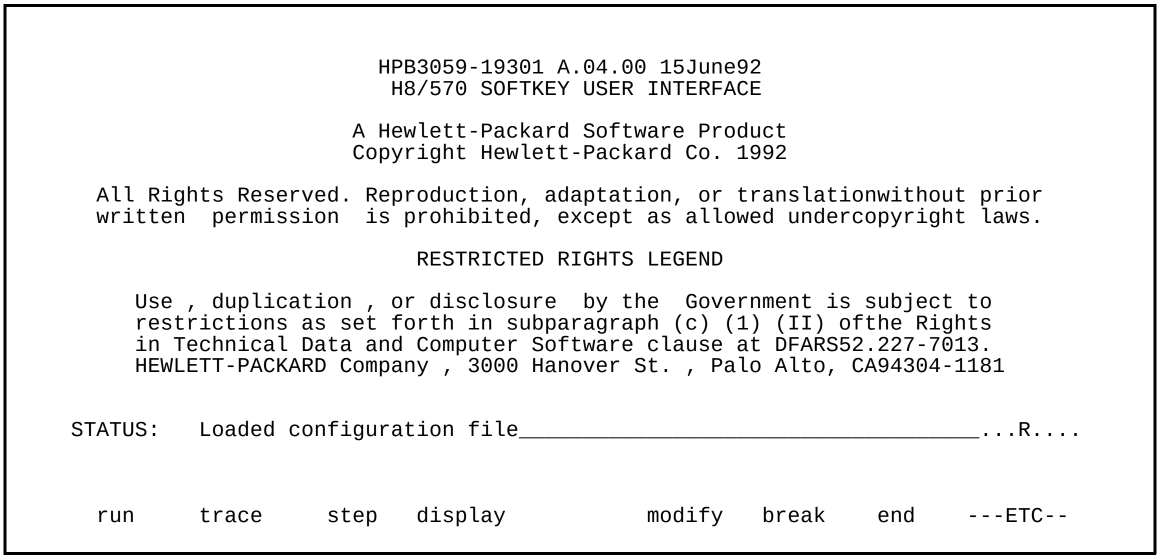
For more information on creating measurements systems, refer to the *Softkey Interface Reference* manual.

From the HP-UX Shell

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

```
$ emul700 <emul_name> <RETURN>
```

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



```
                         HPB3059-19301 A.04.00 15June92
                          H8/570 SOFTKEY USER INTERFACE

                       A Hewlett-Packard Software Product
                       Copyright Hewlett-Packard Co. 1992

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

                            RESTRICTED RIGHTS LEGEND

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


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



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

Figure 2-3. Softkey Interface Display

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

Using the Default Configuration

The default emulator configuration is used with the following examples.

The address range 0 hex through 7FFF hex is mapped as emulation ROM, and F680 hex through FE7F hex as emulation RAM. The emulator operates in mode 1.

On-Line Help

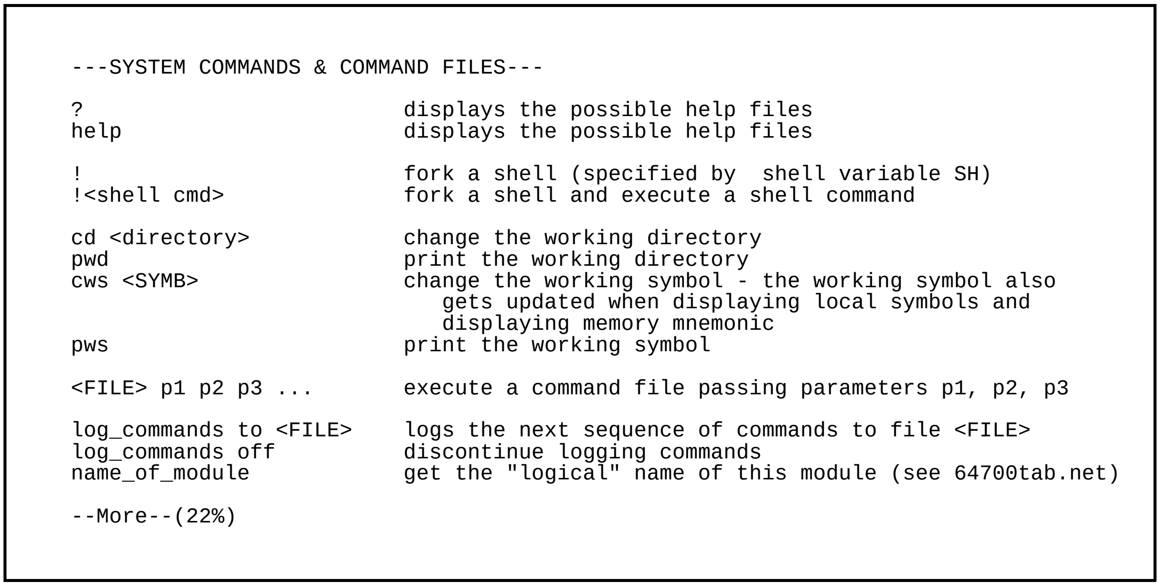
There are two ways to access on-line help in the Softkey Interface. The first is by using the Softkey Interface help facility. The second method allows you to access the firmware resident Terminal Interface on-line help information.

Softkey Driven Help

To access the Softkey Interface on-line help information, type either "help" or "?" on the command line; you will notice a new set of softkeys. By pressing one of these softkeys and < RETURN> , you can cause information on that topic to be displayed on your screen. For example, you can enter the following command to access "system command" help information.

```
? system_commands <RETURN>
```

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



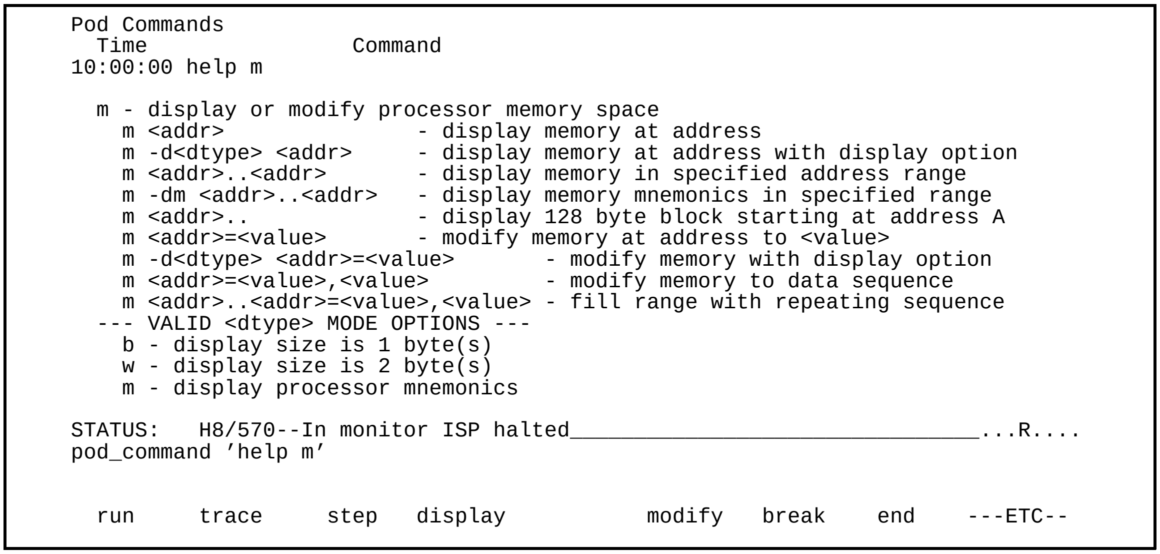
```
---SYSTEM COMMANDS & COMMAND FILES---

?                          displays the possible help files
help                       displays the possible help files

!                          fork a shell (specified by  shell variable SH)
!<shell cmd>               fork a shell and execute a shell command

cd <directory>             change the working directory
pwd                        print the working directory
cws <SYMB>                 change the working symbol - the working symbol also
                              gets updated when displaying local symbols and
                              displaying memory mnemonic
pws                        print the working symbol

<FILE> p1 p2 p3 ...        execute a command file passing parameters p1, p2, p3

log_commands to <FILE>     logs the next sequence of commands to file <FILE>
log_commands off           discontinue logging commands
name_of_module             get the "logical" name of this module (see 64700tab.net)

--More--(22%)
```

Pod Command Help

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

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

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



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

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

STATUS:   H8/570--In monitor ISP halted________________________________...R....
pod_command 'help m'


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

Loading Absolute Files

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

```
load cmd_rds <RETURN>
```

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

Displaying Symbols

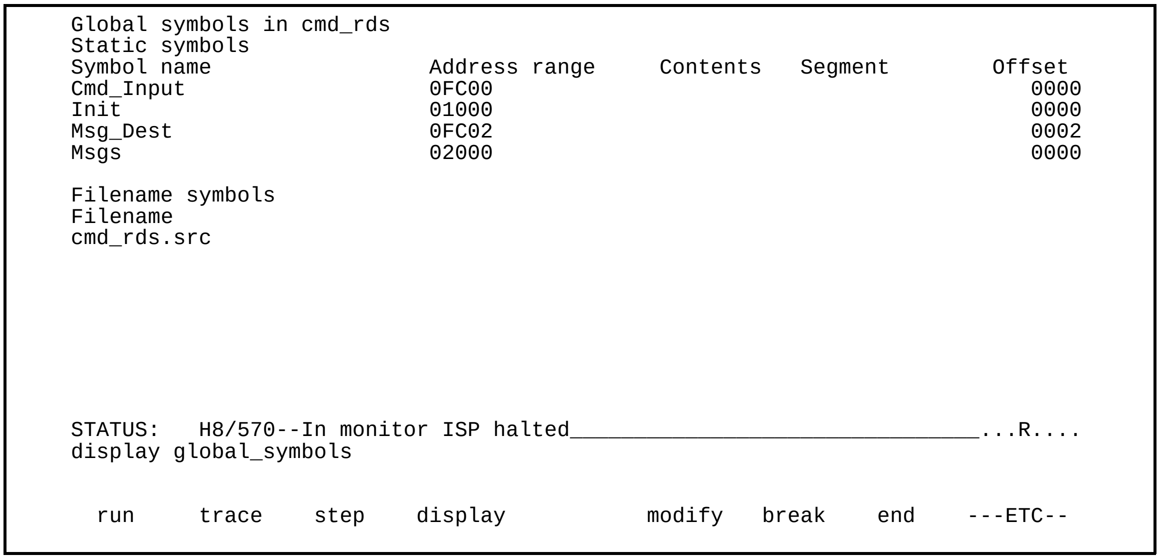
When you load an absolute file into memory (unless you use the "nosymbols" option), symbol information is loaded. Both global symbols and symbols that are local to a source file can be displayed.

Global

To display global symbols, enter the following command.

```
display global_symbols <RETURN>
```

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



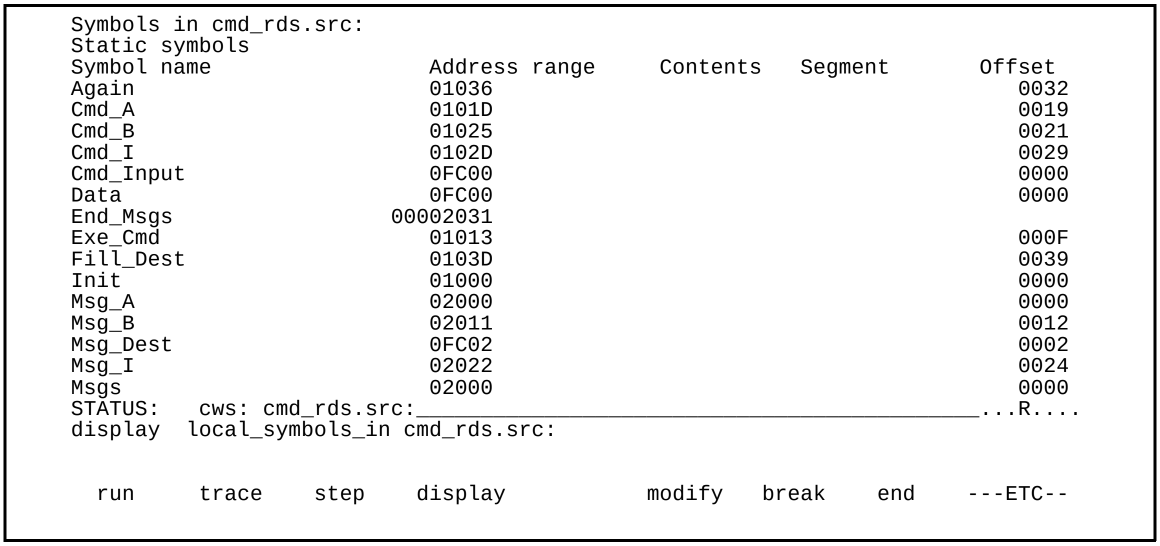
```
Global symbols in cmd_rds
Static symbols
Symbol name                 Address range     Contents   Segment        Offset
Cmd_Input                   0FC00                                          0000
Init                        01000                                          0000
Msg_Dest                    0FC02                                          0002
Msgs                        02000                                          0000

Filename symbols
Filename
cmd_rds.src







STATUS:   H8/570--In monitor ISP halted________________________________...R....
display global_symbols


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

Local

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

```
display local_symbols_in cmd_rds.src:
<RETURN>
```

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



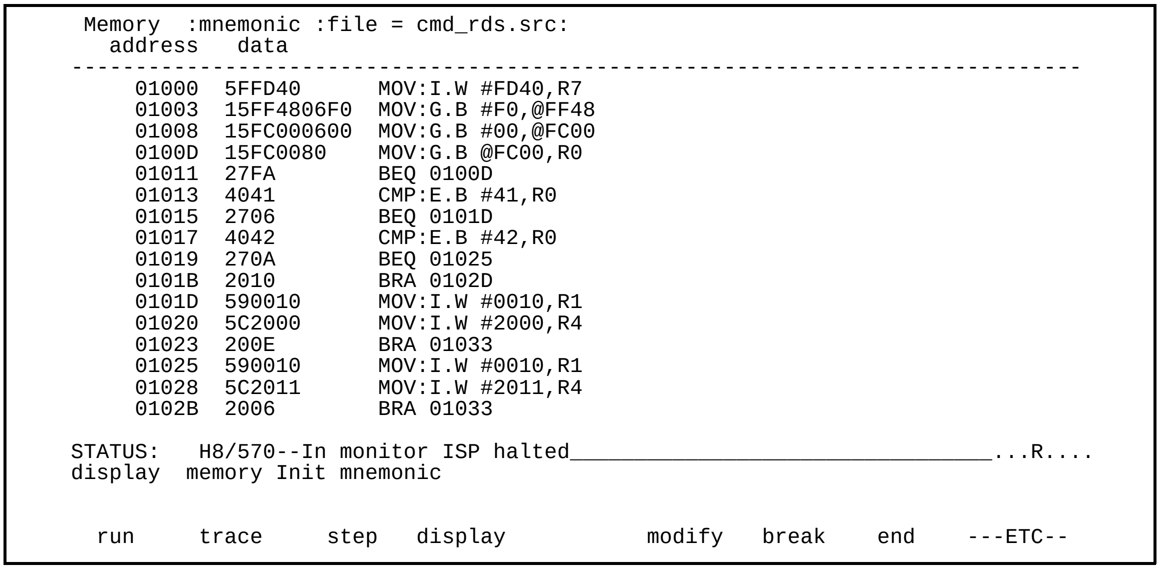
```
Symbols in cmd_rds.src:
Static symbols
Symbol name                 Address range     Contents   Segment       Offset
Again                       01036                                         0032
Cmd_A                       0101D                                         0019
Cmd_B                       01025                                         0021
Cmd_I                       0102D                                         0029
Cmd_Input                   0FC00                                         0000
Data                        0FC00                                         0000
End_Msgs                 00002031
Exe_Cmd                     01013                                         000F
Fill_Dest                   0103D                                         0039
Init                        01000                                         0000
Msg_A                       02000                                         0000
Msg_B                       02011                                         0012
Msg_Dest                    0FC02                                         0002
Msg_I                       02022                                         0024
Msgs                        02000                                         0000
STATUS:   cws: cmd_rds.src:____________________________________________...R....
display  local_symbols_in cmd_rds.src:


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

Displaying Memory in Mnemonic Format

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

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

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



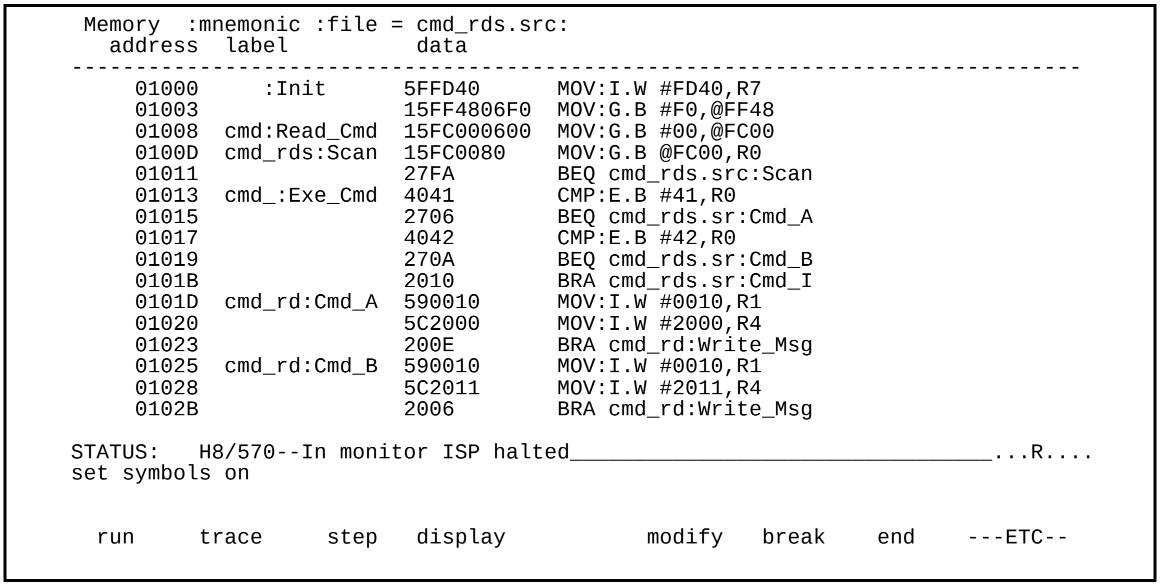
```
 Memory  :mnemonic :file = cmd_rds.src:
   address   data
-------------------------------------------------------------------------------
     01000  5FFD40      MOV:I.W #FD40,R7
     01003  15FF4806F0  MOV:G.B #F0,@FF48
     01008  15FC000600  MOV:G.B #00,@FC00
     0100D  15FC0080    MOV:G.B @FC00,R0
     01011  27FA        BEQ 0100D
     01013  4041        CMP:E.B #41,R0
     01015  2706        BEQ 0101D
     01017  4042        CMP:E.B #42,R0
     01019  270A        BEQ 01025
     0101B  2010        BRA 0102D
     0101D  590010      MOV:I.W #0010,R1
     01020  5C2000      MOV:I.W #2000,R4
     01023  200E        BRA 01033
     01025  590010      MOV:I.W #0010,R1
     01028  5C2011      MOV:I.W #2011,R4
     0102B  2006        BRA 01033

STATUS:   H8/570--In monitor ISP halted_________________________________...R....
display  memory Init mnemonic


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

Display Memory with Symbols

If you want to see symbol information with displaying memory in mnemonic format, the H8/570 emulator Softkey Interface provides "set symbols" command. To see symbol information, enter the following command.

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

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



```
 Memory  :mnemonic :file = cmd_rds.src:
   address  label          data
-------------------------------------------------------------------------------
     01000     :Init      5FFD40      MOV:I.W #FD40,R7
     01003                15FF4806F0  MOV:G.B #F0,@FF48
     01008  cmd:Read_Cmd  15FC000600  MOV:G.B #00,@FC00
     0100D  cmd_rds:Scan  15FC0080    MOV:G.B @FC00,R0
     01011                27FA        BEQ cmd_rds.src:Scan
     01013  cmd_:Exe_Cmd  4041        CMP:E.B #41,R0
     01015                2706        BEQ cmd_rds.sr:Cmd_A
     01017                4042        CMP:E.B #42,R0
     01019                270A        BEQ cmd_rds.sr:Cmd_B
     0101B                2010        BRA cmd_rds.sr:Cmd_I
     0101D  cmd_rd:Cmd_A  590010      MOV:I.W #0010,R1
     01020                5C2000      MOV:I.W #2000,R4
     01023                200E        BRA cmd_rd:Write_Msg
     01025  cmd_rd:Cmd_B  590010      MOV:I.W #0010,R1
     01028                5C2011      MOV:I.W #2011,R4
     0102B                2006        BRA cmd_rd:Write_Msg

STATUS:   H8/570--In monitor ISP halted_________________________________...R....
set symbols on


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

Running the Program

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

From Transfer Address

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

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

From Reset

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

Displaying Memory Repetitively

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

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

Modifying Memory

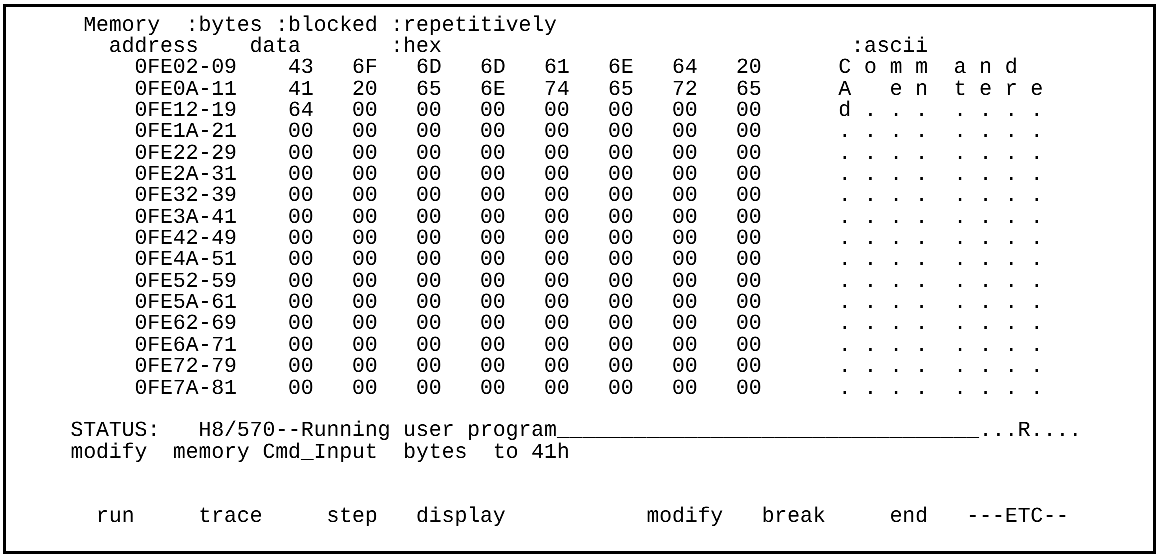
The sample program simulates a primitive command interpreter. Commands are sent to the sample program through a byte sized memory location labeled **Cmd_Input**. You can use the modify memory feature to send a command to the sample program. For example, to enter the command "A" (41 hex), use the following command.

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

Or:

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

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



```
      Memory  :bytes :blocked :repetitively
        address    data       :hex                                   :ascii
          0FE02-09    43   6F   6D   6D   61   6E   64   20      C o m m  a n d
          0FE0A-11    41   20   65   6E   74   65   72   65      A   e n  t e r e
          0FE12-19    64   00   00   00   00   00   00   00      d . . .  . . . .
          0FE1A-21    00   00   00   00   00   00   00   00      . . . .  . . . .
          0FE22-29    00   00   00   00   00   00   00   00      . . . .  . . . .
          0FE2A-31    00   00   00   00   00   00   00   00      . . . .  . . . .
          0FE32-39    00   00   00   00   00   00   00   00      . . . .  . . . .
          0FE3A-41    00   00   00   00   00   00   00   00      . . . .  . . . .
          0FE42-49    00   00   00   00   00   00   00   00      . . . .  . . . .
          0FE4A-51    00   00   00   00   00   00   00   00      . . . .  . . . .
          0FE52-59    00   00   00   00   00   00   00   00      . . . .  . . . .
          0FE5A-61    00   00   00   00   00   00   00   00      . . . .  . . . .
          0FE62-69    00   00   00   00   00   00   00   00      . . . .  . . . .
          0FE6A-71    00   00   00   00   00   00   00   00      . . . .  . . . .
          0FE72-79    00   00   00   00   00   00   00   00      . . . .  . . . .
          0FE7A-81    00   00   00   00   00   00   00   00      . . . .  . . . .

     STATUS:   H8/570--Running user program_________________________________...R....
     modify  memory Cmd_Input  bytes  to 41h


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

Breaking into the Monitor

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

```
break <RETURN>
```

Using Software Breakpoints

Software breakpoints are provided with one of H8/570 undefined opcode (1B hex) as breakpoint interrupt instruction. When you define or enable a software breakpoint, the emulator will replace the opcode at the software breakpoint address with the breakpoint interrupt instruction.

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

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

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

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

Up to 32 software breakpoints may be defined.

Note

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

Note

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

Note

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

Enabling/Disabling Software Breakpoints

When you initially enter the Softkey Interface, software breakpoints are disabled. To enable the software breakpoints feature, enter the following command.

```
modify software_breakpoints enable <RETURN>
```

When software breakpoints are enabled and you set a software breakpoint, the breakpoint interrupt instruction (1B hex) will be placed at the address specified. When the special code is executed, program execution will break into the monitor.

Setting a Software Breakpoint

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

```
modify software_breakpoints set Cmd_I
<RETURN>
```

After the software breakpoint has been set, enter the following command to cause the emulator to continue executing the sample program.

```
run <RETURN>
```

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

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

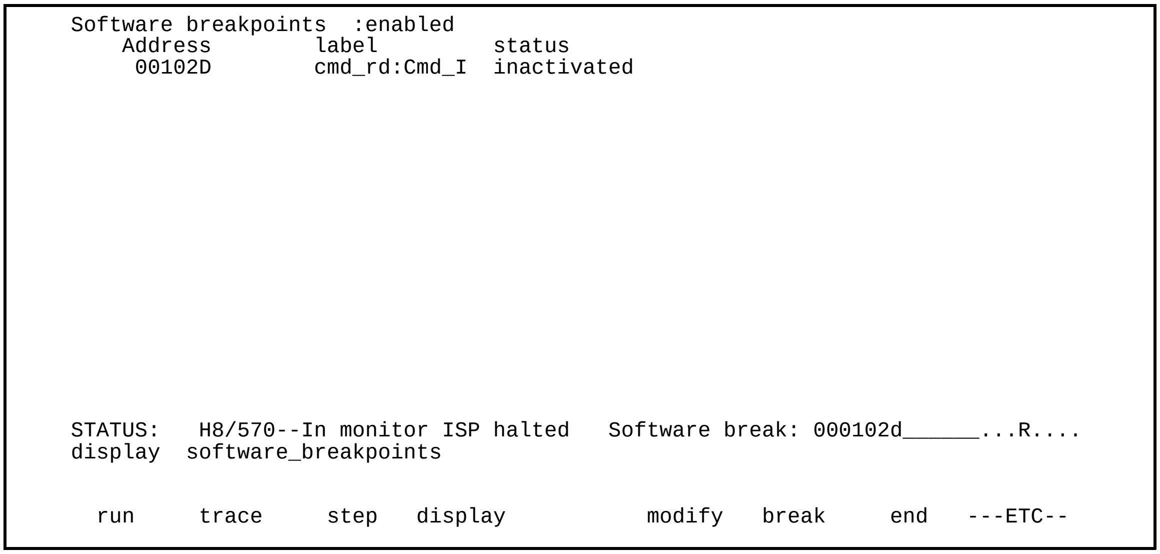
A message on the status line shows that the software breakpoint has been hit. The status line also shows that the emulator is now executing in the monitor.

Displaying Software Breakpoints

To display software breakpoints, enter the following command.

```
display software_breakpoints <RETURN>
```

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



```
Software breakpoints  :enabled
    Address        label         status
     00102D        cmd_rd:Cmd_I  inactivated










STATUS:   H8/570--In monitor ISP halted   Software break: 000102d______...R....
display  software_breakpoints


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

Clearing a Software Breakpoint

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

```
modify software_breakpoints clear Cmd_I
<RETURN>
```

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

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

```
modify software_breakpoints clear <RETURN>
```

Running the Program to A Specified Address

Enter the following command to run the program and break into monitor before execution of the instruction at the **Again** label.

```
run until Again <RETURN>
```

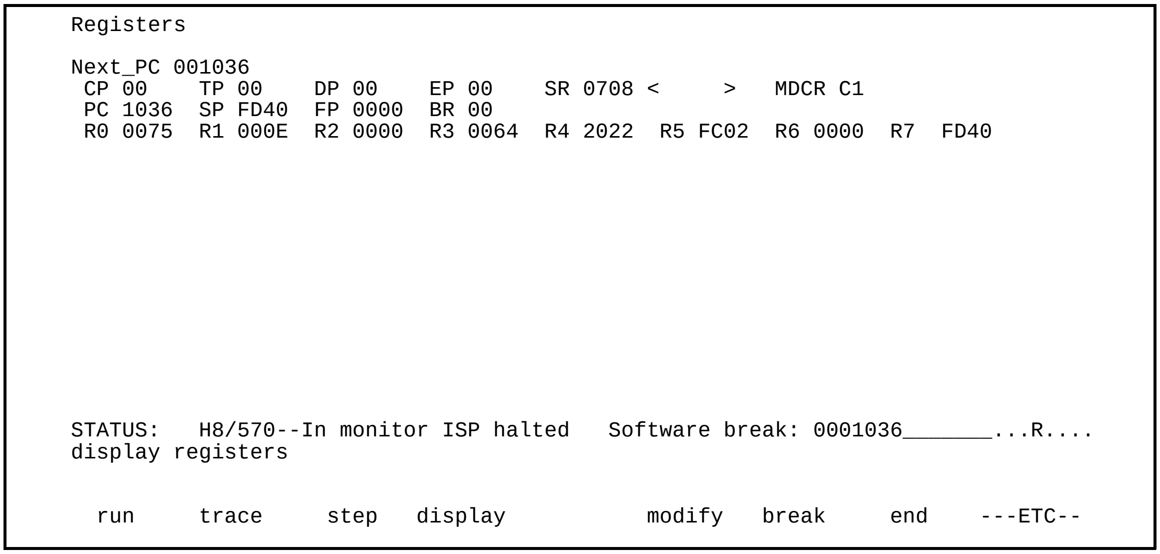
An message on the emulator status line shows that a software breakpoint has been hit. The status line also shows that the emulator is executing in the monitor.

This command is realized by setting a software breakpoint to the specified address. Therefore, you need to notice that the same limitations as the software breakpints are applied to this command.

Displaying Registers

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

```
display registers <RETURN>
```



```
Registers

Next_PC 001036
 CP 00    TP 00    DP 00    EP 00    SR 0708 <     >   MDCR C1
 PC 1036  SP FD40  FP 0000  BR 00
 R0 0075  R1 000E  R2 0000  R3 0064  R4 2022  R5 FC02  R6 0000  R7  FD40









STATUS:   H8/570--In monitor ISP halted   Software break: 0001036_______...R....
display registers


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

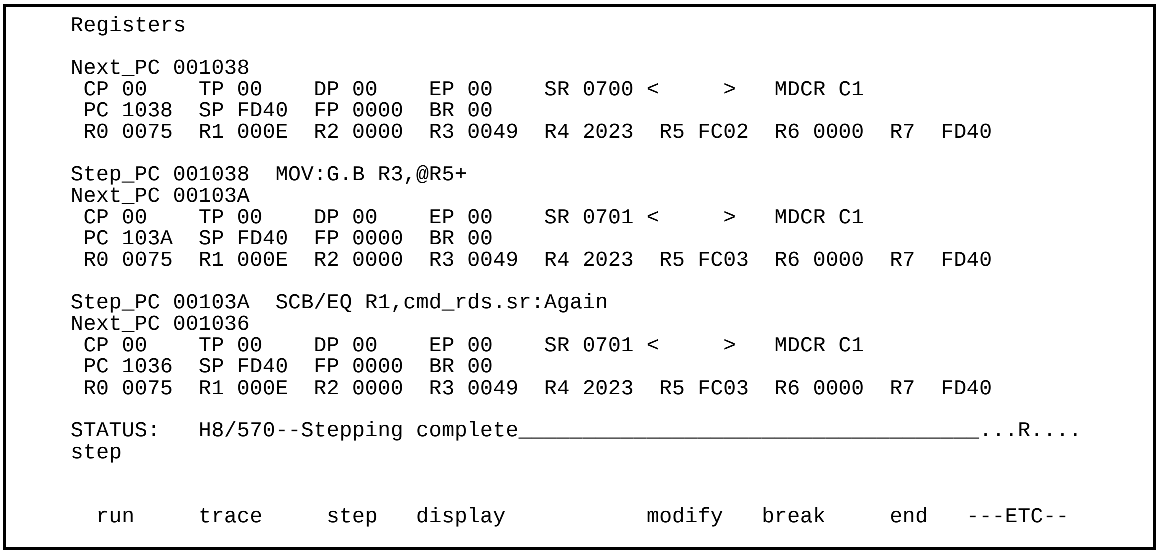
You can use "register class" and "register name" to display registers. Refer to "Register Names and Classes" section in chapter 5.

Stepping Through the Program

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

```
step <RETURN>, <RETURN>, <RETURN>, ...
```

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



```
Registers

Next_PC 001038
 CP 00    TP 00    DP 00    EP 00    SR 0700 <     >   MDCR C1
 PC 1038  SP FD40  FP 0000  BR 00
 R0 0075  R1 000E  R2 0000  R3 0049  R4 2023  R5 FC02  R6 0000  R7  FD40

Step_PC 001038  MOV:G.B R3,@R5+
Next_PC 00103A
 CP 00    TP 00    DP 00    EP 00    SR 0701 <     >   MDCR C1
 PC 103A  SP FD40  FP 0000  BR 00
 R0 0075  R1 000E  R2 0000  R3 0049  R4 2023  R5 FC03  R6 0000  R7  FD40

Step_PC 00103A  SCB/EQ R1,cmd_rds.sr:Again
Next_PC 001036
 CP 00    TP 00    DP 00    EP 00    SR 0701 <     >   MDCR C1
 PC 1036  SP FD40  FP 0000  BR 00
 R0 0075  R1 000E  R2 0000  R3 0049  R4 2023  R5 FC03  R6 0000  R7  FD40

STATUS:   H8/570--Stepping complete____________________________________...R....
step


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

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

```
run <RETURN>
```

Using the Analyzer

HP 64700 emulators contain an emulation analyzer. The emulation analyzer monitors the internal emulation lines (address, data, and status). Optionally, you may have an additional 16 trace signals which monitor external input lines. The analyzer collects data at each pulse of a clock signal, and saves the data (a trace state) if it meets a "storage qualification" condition.

Specifying a Simple Trigger

Suppose you want to trace program execution after the point at which the sample program reads the "B" (42 hex) command from the command input byte. To do this, you would trace after the analyzer finds a state in which a value of 42xxh is read from the **Cmd_Input** byte. The following command makes this trace specification.

```
trace after Cmd_Input data 42xxh status read
<RETURN>
```

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

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

The status line now shows "Emulation trace complete".

Notice that the data was specified with the don't care bits (**xx**). When a byte access is performed, the data appears on the upper 8 bit of analyzer data bus.

H8/570 Analysis Status Qualifiers

The status qualifier "read" was used in the example trace command used before in this chapter. The following analysis status qualifiers may also be used with the H8/570 emulator.

```
Qualifier   Description                        Status Bits  (36..63)
--------    -------------------------------    -----------------------------------
backgrnd    Background cycle                   xxxx xxxx xxxx xxx0 0xxx xxxx xxxxB
brelease    Bus release cycle                  xxxx xxxx xxxx xxx0 x11x xxxx xxxxB
byte        Byte Access                        xxxx xxxx xxxx xxx0 x10x xxxx xx1xB
cpu         CPU cycle                          xxxx xxxx xxxx xxx0 x101 1xxx xxxxB
data        Data access                        xxxx xxxx xxxx xxx0 x10x xxxx x1xxB
dtc         DTC cycle                          xxxx xxxx xxxx xxx0 x101 0xxx xxxxB
exec        Instruction execution cycle        xxxx xxxx xxxx xxx0 x01x xxxx xxxxB
fetch       Program fetch cycle                xxxx xxxx xxxx xxx0 x101 1xxx x001B
foregrnd    Foreground cycle                   xxxx xxxx xxxx xxx0 1xxx xxxx xxxxB
grd         Guarded memory access              xxxx xxxx xxxx xxx0 x10x x011 xxxxB
io          Internal I/O access                xxxx xxxx xxxx xxx0 x10x xxx0 xxxxB
isp         Memory cycle by ISP                xxxx xxxx xxxx xxx0 xx00 1xxx xxxxB
ispexec     ISP instruction execution cycle    xxxx xxxx xxxx x0xx xxxx xxxx xxxxB
memory      Memory access                      xxxx xxxx xxxx xxx0 x10x xxx1 xxxxB
read        Read cycle                         xxxx xxxx xxxx xxx0 x10x xxxx xxx1B
refresh     Refresh cycle                      xxxx xxxx xxxx xxx0 x000 1xxx xxxxB
word        Word Access                        xxxx xxxx xxxx xxx0 x10x xxxx xx0xB
write       Write cycle                        xxxx xxxx xxxx xxx0 x10x xxxx xxx0B
wrrom       Write to ROM cycle                 xxxx xxxx xxxx xxx0 x10x x101 xxx0B
```

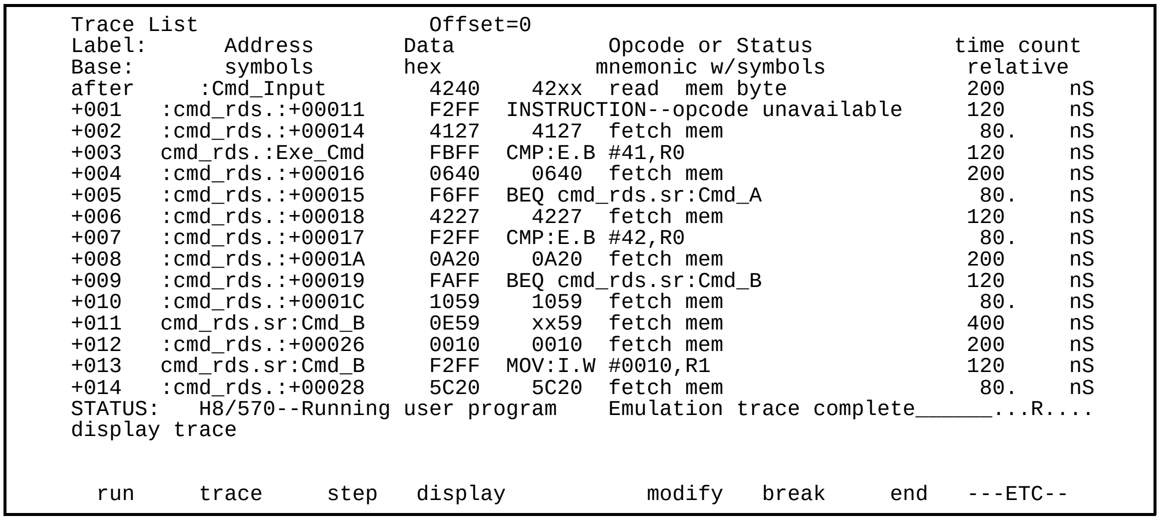
Note

You need to specify the "**exec**" status qualifier to trigger the analyzer by an execution cycle.

Displaying the Trace

The trace listings which follow are of program execution on the H8/570 emulator. To display the trace, enter:

```
display trace <RETURN>
```

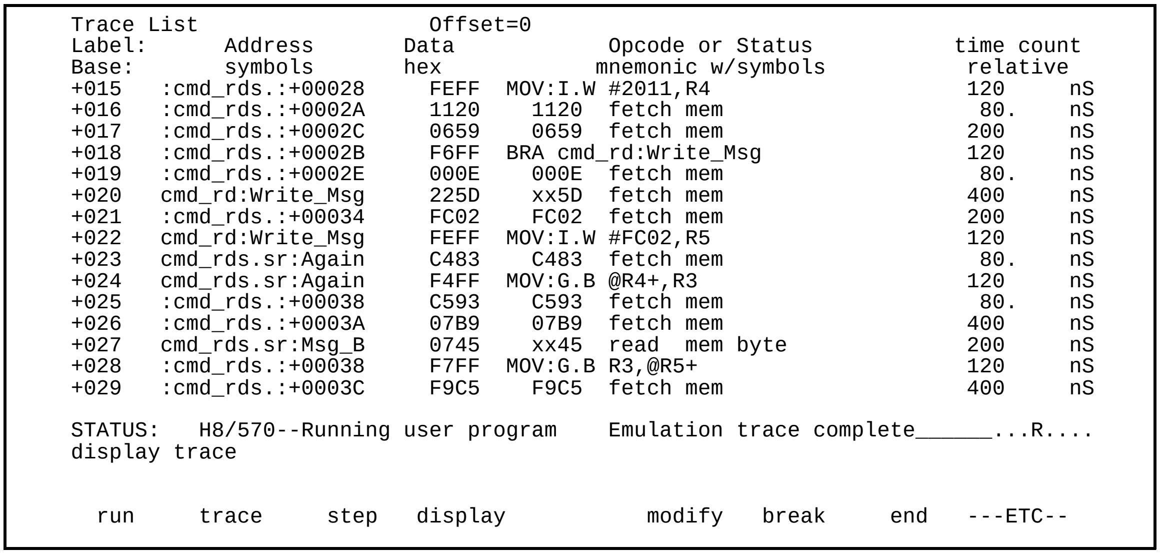


```
Trace List                  Offset=0
Label:      Address       Data          Opcode or Status           time count
Base:       symbols       hex          mnemonic w/symbols           relative
after     :Cmd_Input        4240    42xx  read  mem byte              200     nS
+001   :cmd_rds.:+00011     F2FF  INSTRUCTION--opcode unavailable     120     nS
+002   :cmd_rds.:+00014     4127    4127  fetch mem                    80.    nS
+003   cmd_rds.:Exe_Cmd     FBFF  CMP:E.B #41,R0                      120     nS
+004   :cmd_rds.:+00016     0640    0640  fetch mem                   200     nS
+005   :cmd_rds.:+00015     F6FF  BEQ cmd_rds.sr:Cmd_A                 80.    nS
+006   :cmd_rds.:+00018     4227    4227  fetch mem                   120     nS
+007   :cmd_rds.:+00017     F2FF  CMP:E.B #42,R0                       80.    nS
+008   :cmd_rds.:+0001A     0A20    0A20  fetch mem                   200     nS
+009   :cmd_rds.:+00019     FAFF  BEQ cmd_rds.sr:Cmd_B                120     nS
+010   :cmd_rds.:+0001C     1059    1059  fetch mem                    80.    nS
+011   cmd_rds.sr:Cmd_B     0E59    xx59  fetch mem                   400     nS
+012   :cmd_rds.:+00026     0010    0010  fetch mem                   200     nS
+013   cmd_rds.sr:Cmd_B     F2FF  MOV:I.W #0010,R1                    120     nS
+014   :cmd_rds.:+00028     5C20    5C20  fetch mem                    80.    nS
STATUS:   H8/570--Running user program    Emulation trace complete______...R....
display trace


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

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

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

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



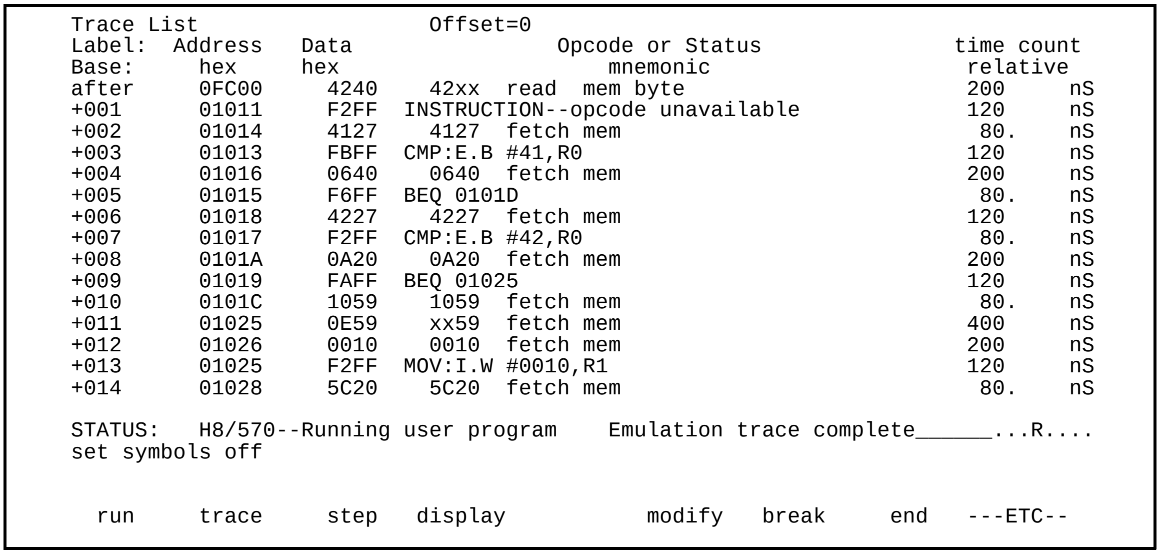
```
Trace List                  Offset=0
Label:      Address       Data            Opcode or Status           time count
Base:       symbols       hex            mnemonic w/symbols           relative
+015   :cmd_rds.:+00028     FEFF  MOV:I.W #2011,R4                    120     nS
+016   :cmd_rds.:+0002A     1120    1120  fetch mem                    80.    nS
+017   :cmd_rds.:+0002C     0659    0659  fetch mem                   200     nS
+018   :cmd_rds.:+0002B     F6FF  BRA cmd_rd:Write_Msg                120     nS
+019   :cmd_rds.:+0002E     000E    000E  fetch mem                    80.    nS
+020   cmd_rd:Write_Msg     225D    xx5D  fetch mem                   400     nS
+021   :cmd_rds.:+00034     FC02    FC02  fetch mem                   200     nS
+022   cmd_rd:Write_Msg     FEFF  MOV:I.W #FC02,R5                    120     nS
+023   cmd_rds.sr:Again     C483    C483  fetch mem                    80.    nS
+024   cmd_rds.sr:Again     F4FF  MOV:G.B @R4+,R3                     120     nS
+025   :cmd_rds.:+00038     C593    C593  fetch mem                    80.    nS
+026   :cmd_rds.:+0003A     07B9    07B9  fetch mem                   400     nS
+027   cmd_rds.sr:Msg_B     0745    xx45  read  mem byte              200     nS
+028   :cmd_rds.:+00038     F7FF  MOV:G.B R3,@R5+                     120     nS
+029   :cmd_rds.:+0003C     F9C5    F9C5  fetch mem                   400     nS

STATUS:   H8/570--Running user program    Emulation trace complete______...R....
display trace


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

Displaying Trace with No Symbol

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

```
    set symbols off
```

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

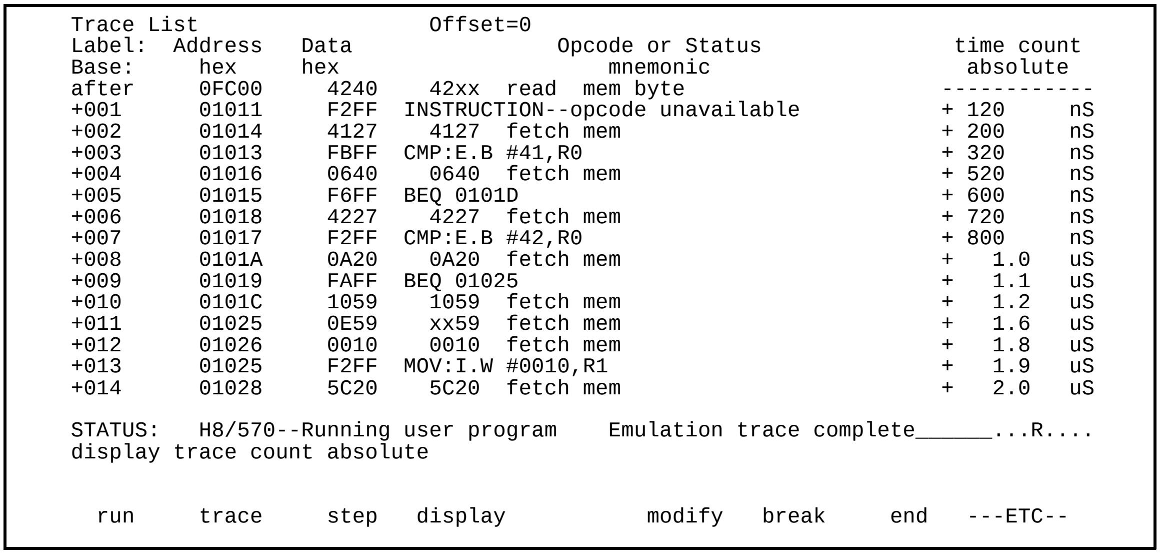


```
Trace List                  Offset=0
Label:  Address   Data                Opcode or Status              time count
Base:     hex     hex                     mnemonic                   relative
after     0FC00     4240    42xx  read  mem byte                       200     nS
+001      01011     F2FF  INSTRUCTION--opcode unavailable              120     nS
+002      01014     4127    4127  fetch mem                             80.    nS
+003      01013     FBFF  CMP:E.B #41,R0                               120     nS
+004      01016     0640    0640  fetch mem                            200     nS
+005      01015     F6FF  BEQ 0101D                                     80.    nS
+006      01018     4227    4227  fetch mem                            120     nS
+007      01017     F2FF  CMP:E.B #42,R0                                80.    nS
+008      0101A     0A20    0A20  fetch mem                            200     nS
+009      01019     FAFF  BEQ 01025                                    120     nS
+010      0101C     1059    1059  fetch mem                             80.    nS
+011      01025     0E59    xx59  fetch mem                            400     nS
+012      01026     0010    0010  fetch mem                            200     nS
+013      01025     F2FF  MOV:I.W #0010,R1                             120     nS
+014      01028     5C20    5C20  fetch mem                             80.    nS

STATUS:   H8/570--Running user program    Emulation trace complete______...R....
set symbols off


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

Displaying Trace with Time Count Absolute

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

```
    display trace count absolute <RETURN>
```

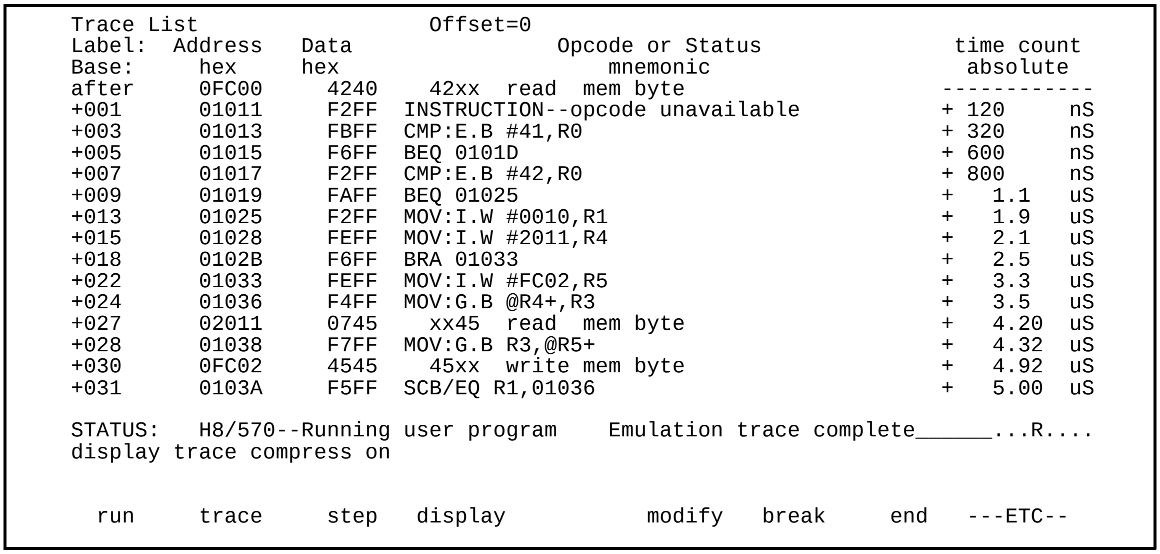


```
Trace List                  Offset=0
Label:  Address   Data                Opcode or Status               time count
Base:     hex     hex                     mnemonic                    absolute
after     0FC00     4240    42xx  read  mem byte                    ------------
+001      01011     F2FF  INSTRUCTION--opcode unavailable           + 120     nS
+002      01014     4127    4127  fetch mem                         + 200     nS
+003      01013     FBFF  CMP:E.B #41,R0                            + 320     nS
+004      01016     0640    0640  fetch mem                         + 520     nS
+005      01015     F6FF  BEQ 0101D                                 + 600     nS
+006      01018     4227    4227  fetch mem                         + 720     nS
+007      01017     F2FF  CMP:E.B #42,R0                            + 800     nS
+008      0101A     0A20    0A20  fetch mem                         +   1.0   uS
+009      01019     FAFF  BEQ 01025                                 +   1.1   uS
+010      0101C     1059    1059  fetch mem                         +   1.2   uS
+011      01025     0E59    xx59  fetch mem                         +   1.6   uS
+012      01026     0010    0010  fetch mem                         +   1.8   uS
+013      01025     F2FF  MOV:I.W #0010,R1                          +   1.9   uS
+014      01028     5C20    5C20  fetch mem                         +   2.0   uS

STATUS:   H8/570--Running user program    Emulation trace complete______...R....
display trace count absolute


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

Displaying Trace with Compress Mode

If you want to see more executed instructions on a display, the H8/570 emulator Softkey Interface provides **compress mode** for analysis display. To see trace display with compress mode, enter the following command:

```
display trace compress on <RETURN>
```



```
Trace List                  Offset=0
Label:  Address   Data                Opcode or Status               time count
Base:     hex     hex                     mnemonic                    absolute
after     0FC00     4240    42xx  read  mem byte                    ------------
+001      01011     F2FF  INSTRUCTION--opcode unavailable           + 120     nS
+003      01013     FBFF  CMP:E.B #41,R0                            + 320     nS
+005      01015     F6FF  BEQ 0101D                                 + 600     nS
+007      01017     F2FF  CMP:E.B #42,R0                            + 800     nS
+009      01019     FAFF  BEQ 01025                                 +   1.1   uS
+013      01025     F2FF  MOV:I.W #0010,R1                          +   1.9   uS
+015      01028     FEFF  MOV:I.W #2011,R4                          +   2.1   uS
+018      0102B     F6FF  BRA 01033                                 +   2.5   uS
+022      01033     FEFF  MOV:I.W #FC02,R5                          +   3.3   uS
+024      01036     F4FF  MOV:G.B @R4+,R3                           +   3.5   uS
+027      02011     0745    xx45  read  mem byte                    +   4.20  uS
+028      01038     F7FF  MOV:G.B R3,@R5+                           +   4.32  uS
+030      0FC02     4545    45xx  write mem byte                    +   4.92  uS
+031      0103A     F5FF  SCB/EQ R1,01036                           +   5.00  uS

STATUS:   H8/570--Running user program    Emulation trace complete______...R....
display trace compress on


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

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

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

```
display trace compress off <RETURN>
```

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

Changing the Trace Depth

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

```
display trace depth 512 <RETURN>
```

Now the states displayed in the trace list is changed to 512 states.

For a Complete Description

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

Exiting the Softkey Interface

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

End Release System

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

```
end release_system <RETURN>
```

Ending to Continue Later

You may also exit the Softkey Interface without specifying any options; this causes the emulator to be locked. When the emulator is locked, other users are prevented from using it and the emulator configuration is saved so that it can be restored the next time you enter (continue) the Softkey Interface.

```
end <RETURN>
```

Ending Locked from All Windows

When using the Softkey Interface from within window systems, the "end" command with no options causes an exit only in that window. To end locked from all windows, enter the following command.

```
end locked <RETURN>
```

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

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

Selecting the Measurement System Display or Another Module

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

```
end select measurement_system <RETURN>
```

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

3

Debugging ISP Functions

The HP 64730 H8/570 emulator is equipped with commands for debugging ISP functions. You can direct the ISP to run, halt, or execute a specified number of instructions. The analyzer allows you to monitor the execution of your program, or ISP functions, or both of them.

In this chapter, we use a sample program and learn how to use the emulator to debug the ISP functions. When you have completed this chapter, you will be able to perform these tasks:

- Load ISP functions into the emulator
- Use run/stop controls to control operation of your ISP functions
- Use register display command to view the contents of ISP registers
- Use analyzer commands to view the real time execution of your ISP functions

Sample Program with Small ISP Functions

In the "Getting Started" chapter, we looked at a sample program which functioned as a primitive command interpreter. It wrote various messages to an output buffer, depending on the character you inserted in the input buffer.

In this chapter, we use a modified version of the "Getting Started" program. It still performs the same function, but works with a small ISP function. The ISP function takes charge of the transfer of the messages. Once a command is written to the input buffer, the sample program determines the message to be written and pass the source address to an ISP register. The ISP function starts to transfer the message when an ISP flag is cleared by the program. When the transfer is finished, the program goes back to read the next command. Figure 3-1 lists the sample program and Figure 3-2 lists the sample ISP functions.

Processing Commands

The instructions at **Cmd_A**, **Cmd_B**, and **Cmd_I** each load ISP data register 2 with the length of the message to be written and ISP data register 0 with the starting location of the message. Then, execution transfers to **Write_Msg** which loads the destination address into the ISP data register 1.

The ISP starts transferring a message by clearing an ISP flag. The program will wait the completion of the transfer.

ISP Function 0

ISP function 0 performs data transfer from a specified address to a destination address. ISP data register 0 is used to contain the source address. ISP data register 1 is used to contain the destination address. When the ISFL (Interrupt Status Flag) 0 is cleared, the function starts transferring data.

ISP Function 1 and 2

Function 1 and 2 are dummy functions.

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

WCR             .EQU        H'FF48
ISP_DR0         .EQU        H'FEC0
ISP_DR1         .EQU        H'FEC2
ISP_DR2         .EQU        H'FEC4
ISP_ISFL        .EQU        H'FEB1
ISP_ICSR        .EQU        H'FF19

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

                .SECTION    Prog,CODE
;*****************************************************
;* Sets up the stack pointer and the Wait-state
;* controller.  Enables the ISP.
;*****************************************************
Init            MOV.W       #Stack,R7
                MOV.W       #H'f0,@WCR
                BCLR.B      #5,@ISP_ICSR
;*****************************************************
;* Clear previous command.
;*****************************************************
Read_Cmd        MOV.B       #0,@Cmd_Input
;*****************************************************
;* Read command input byte.  If no command has
;* been entered, continue to scan for input.
;*****************************************************
Scan            MOV.B       @Cmd_Input,R0
                BEQ         Scan
;*****************************************************
;* A command has been entered.  Check if it is
;* command A, command B, or invalid.
;*****************************************************
Exe_Cmd         CMP.B       #H'41,R0
                BEQ         Cmd_A
                CMP.B       #H'42,R0
                BEQ         Cmd_B
                BRA         Cmd_I
;*****************************************************
;* Command A is entered.  R1 = the number of
;* bytes in message A.  R4 = location of the
;* message.  Jump to the routine which writes
;* the messages.
;*****************************************************
Cmd_A           MOV.W       #Msg_B-Msg_A,@ISP_DR2
                MOV.W       #Msg_A,@ISP_DR0
                BRA         Write_Msg
;*****************************************************
;* Command B is entered.
;*****************************************************
```

Figure 3-1. Sample Program with ISP

```
Cmd_B           MOV.W       #Msg_I-Msg_B,@ISP_DR2
                MOV.W       #Msg_B,@ISP_DR0
                BRA         Write_Msg
;*****************************************************
;* An invalid command is entered.
;*****************************************************
Cmd_I           MOV.W       #End_Msgs-Msg_I,@ISP_DR2
                MOV.W       #Msg_I,@ISP_DR0
;*****************************************************
;* Message is written to the destination.
;*****************************************************
Write_Msg       MOV.W       #Msg_Dest,@ISP_DR1
;*****************************************************
;* Clear ISFL0 to start the DMA.
;*****************************************************
                BCLR.B      #0,@ISP_ISFL
Wait_ISP        BTST.B      #0,@ISP_ISFL
                BEQ         Wait_ISP
;*****************************************************
;* The rest of the destination area is filled
;* with zeros.
;*****************************************************
Fill_Dest       MOV.W       @ISP_DR1,R5
Fill_Loop       MOV.B       #0,@R5+
                CMP.W       #Msg_Dest+H'20,R5
                BNE         Fill_Loop
;*****************************************************
;* Go back and scan for next command.
;*****************************************************
                BRA         Read_Cmd

                .SECTION    Data,COMMON
;*****************************************************
;* Command input byte.
;*****************************************************
Cmd_Input       .RES.B      H'1
                .RES.B      H'1
;*****************************************************
;* Destination of the command messages.
;*****************************************************
Msg_Dest        .RES.B      H'3E
                .RES.W      H'80        ; Stack area.
Stack

                .END        Init
```

Figure 3-2. Sample Program with ISP (Cont'd)

```
.program sample;

.SCM;
        func0/R, func1/R, func0/R, func2/R;
.end;

/* Function 0
 *      dr0: source address
 *      dr1: destination address
 *      dr2: loop counter
 *      isfl0: DMA starts when CPU sets this flag to 0 */
.function func0, ar0;
init:   out() 1, isfl0;
        next (isfl0) $, label;
label:  next() loop;
loop:   read.b dr0, mab         next(!c) $, labelS;
labelS: add.w 0, #1, dr0;
        write.b dr1, mab        next(!c) $, labelD;
labelD: add.w 0, #1, dr1;
        sub.w 0, #1, dr2        next(!z) loop2, exit;
loop2:  next() loop;
exit:   next() init;
.end;

.function func1, ar1;
loop1:  mov.w #3, dr3;
        mov.w #0, dr3;
        next() loop1;
.end;

.function func2, ar2;
loop2:  mov.w #4, dr4;
        mov.w #0, dr4;
        next() loop2;
.end;
.end;
```

Figure 3-2. Sample ISP Function

Sample Program Locations

The sample program is written for the HP 64869 H8/500 Assembler/Linkage Editor. The sample ISP function is written for Hitachi ISP Assembler. The sample programs are shipped with the Softkey Interface, and may be copied from the following locations.

/usr/hp64000/demo/emul/hp64730/cmd_rds2.src

/usr/hp64000/demo/emul/hp64730/ispsamp.mar

Assembling the Sample Program

You can assemble and link the sample program with the following commands:

```
$ h8asm -debug cmd_rds2.src <RETURN>
$ h8lnk -subcommand=cmd_rds2.k <RETURN>
$ h8cnvhp cmd_rds2 <RETURN>
```

In the above command, **cmd_rds2.k** is a linkage editor command file, and its contents is as follows:

```
debug
input cmd_rds2
start Prog(1000), Table(2000), Data(0FC00)
output cmd_rds2
print cmd_rds2
exit
```

Assembling the Sample ISP Functions

You can assemble the sample ISP functions by HITACHI ISP Assembler. Refer to the manual provided with the tool for information on the usage of the ISP assembler.

Converting Your ISP Functions

The HITACHI ISP Assembler generates absolute file in Motorola-S records. To load the file into the emulator, you need to convert the file format with the **xlate** utility provided with the Softkey Interface. The utility converts the Motorola format into HP format which can be consumed by the Softkey Interface.

Suppose that you assembled the sample ISP function with the HITACHI ISP Assembler, and got an absolute file with filename "**ispsamp.mot**". To convert the file format, enter the following command:

```
$ xlate -tmot ispsamp.mot <RETURN>
```

An HP absolute file **ispsamp.X** is generated.

Entering the Softkey Interface

Start the Softkey Interface with the following command:

```
   $ emul700 <emul_name> <RETURN>
```

If you have been working with the emulator and the Softkey Interface is already running, please "**end release**" the interface and restart it. You should follow the steps to ensure that the emulator will work as described in the examples below.

Loading Absolute Files

Load the sample program with the following command:

```
   load cmd_rds2 <RETURN>
```

To load ISP functions, the ISP must be in the halt state. Halt the ISP with the following command:

```
   break with_isp <RETURN>
```

Load the sample ISP function:

```
   load isp_memory ispsamp <RETURN>
```

Note

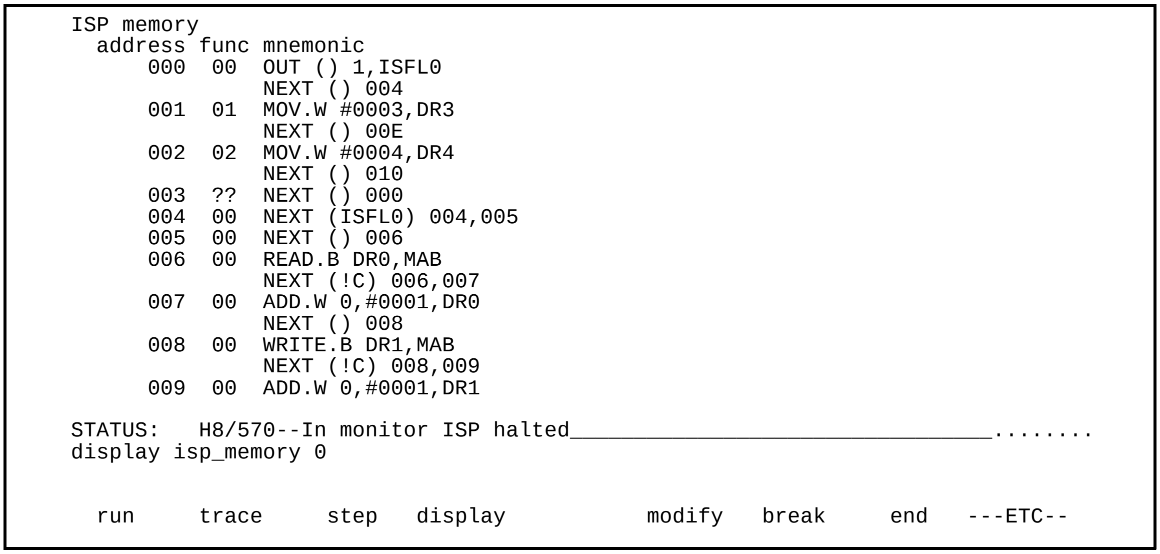
The only way to modify ISP microprogram memory is loading ISP functions with the **load** command. You cannot modify the memory with any emulation commands.

Looking at Your ISP Code

Now that you have loaded the sample ISP function into the emulator, you can display it in mnemonic format. To display the ISP microprogram memory from address 0, type:

```
display isp_memory 0 <RETURN>
```

You will see:

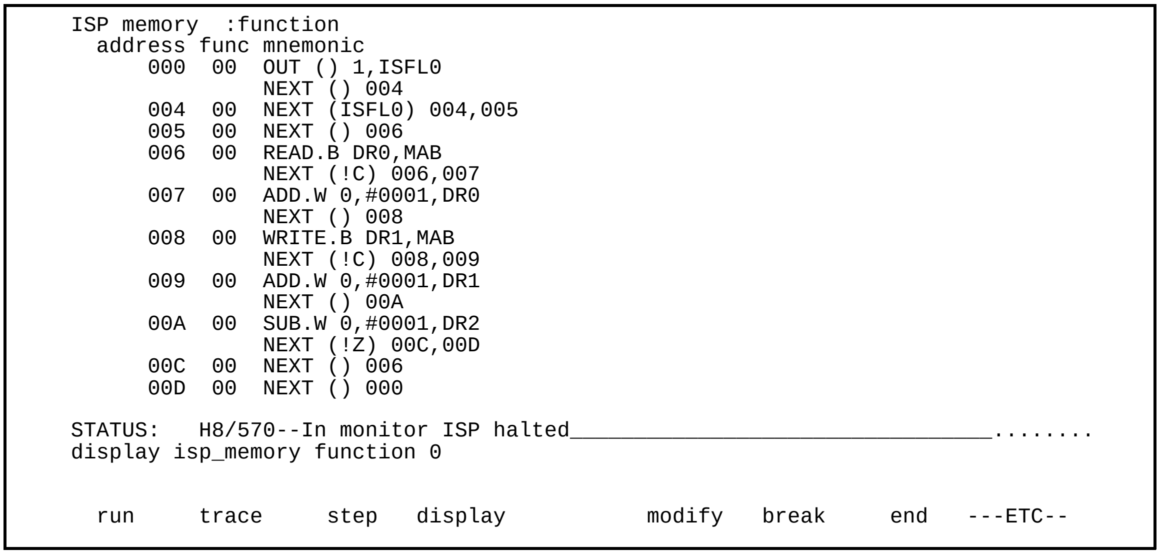


```
ISP memory
  address func mnemonic
      000  00  OUT () 1,ISFL0
               NEXT () 004
      001  01  MOV.W #0003,DR3
               NEXT () 00E
      002  02  MOV.W #0004,DR4
               NEXT () 010
      003  ??  NEXT () 000
      004  00  NEXT (ISFL0) 004,005
      005  00  NEXT () 006
      006  00  READ.B DR0,MAB
               NEXT (!C) 006,007
      007  00  ADD.W 0,#0001,DR0
               NEXT () 008
      008  00  WRITE.B DR1,MAB
               NEXT (!C) 008,009
      009  00  ADD.W 0,#0001,DR1

STATUS:   H8/570--In monitor ISP halted_________________________________........
display isp_memory 0


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

The contents of ISP microprogram memory is displayed in mnemonic format. The first column shows the address in the microprogram memory. The second column is the number of the function to which each instruction belongs. If this field shows "??", the address is not used by any functions defined in the SCM. The third column is the instruction at the address.

You can also display instructions which belong to a specified function. For example, to see only instructions of function 0, enter:

```
display isp_memory function 0 <RETURN>
```



```
ISP memory  :function
  address func mnemonic
      000  00  OUT () 1,ISFL0
               NEXT () 004
      004  00  NEXT (ISFL0) 004,005
      005  00  NEXT () 006
      006  00  READ.B DR0,MAB
               NEXT (!C) 006,007
      007  00  ADD.W 0,#0001,DR0
               NEXT () 008
      008  00  WRITE.B DR1,MAB
               NEXT (!C) 008,009
      009  00  ADD.W 0,#0001,DR1
               NEXT () 00A
      00A  00  SUB.W 0,#0001,DR2
               NEXT (!Z) 00C,00D
      00C  00  NEXT () 006
      00D  00  NEXT () 000

STATUS:   H8/570--In monitor ISP halted_________________________________........
display isp_memory function 0


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

Note

The H8/570 Softkey Interface does **not** support symbolic information for ISP functions. Symbolic information for ISP functions is not displayed in memory display and trace listing.

Controlling ISP Execution

Reset the emulator with the following command:

```
   reset <RETURN>
```

Run the ISP with the following command:

```
   run isp <RETURN>
```

The status message will be displayed as follows:

```
STATUS:  H8/570--Running in monitor
```

The ISP started execution from current ISP address by the **run** command. The emulator breaks into the monitor when the command is used while the emulator is in the reset state.

Halt the ISP with the following command:

```
   break with_isp <RETURN>
```

```
STATUS:  H8/570--In monitor ISP halted
```

The **break with_isp** command breaks the emulator into the monitor, and halts the ISP.

Run the sample program from the **Init** label:

```
   run from Init <RETURN>
```

The ISP is enabled by the sample program, and starts execution. Now break the execution into the monitor:

```
   break <RETURN>
```

```
STATUS:  H8/570--In monitor ISP halted
```

By default, the ISP is halted when the emulator breaks into the monitor. You can configure the emulator not to halt the ISP on emulation break. Refer to Chapter 5 of this manual.

Stepping ISP Function

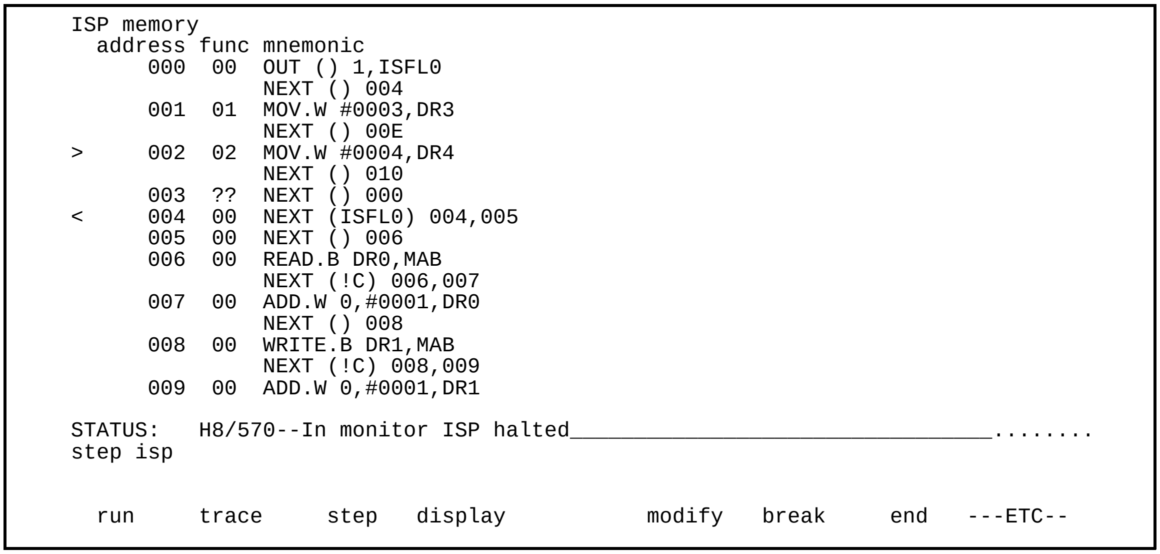
You can direct the emulator to execute one or specified number of ISP instructions. Before you step through the ISP function, display the ISP memory from address 0:

```
display isp_memory 0 <RETURN>
```

Now, step the sample ISP function. Type:

```
step isp <RETURN>, <RETURN>, <RETURN>,...
```

You will see a similar display to the following:



```
ISP memory
  address func mnemonic
      000  00  OUT () 1,ISFL0
               NEXT () 004
      001  01  MOV.W #0003,DR3
               NEXT () 00E
>     002  02  MOV.W #0004,DR4
               NEXT () 010
      003  ??  NEXT () 000
<     004  00  NEXT (ISFL0) 004,005
      005  00  NEXT () 006
      006  00  READ.B DR0,MAB
               NEXT (!C) 006,007
      007  00  ADD.W 0,#0001,DR0
               NEXT () 008
      008  00  WRITE.B DR1,MAB
               NEXT (!C) 008,009
      009  00  ADD.W 0,#0001,DR1

STATUS:   H8/570--In monitor ISP halted_________________________________........
step isp


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

You will see a left bracket (<) at the beginning of a line in the memory display. This shows that the instruction at the line was executed by the step command. You may also see a right bracket (>) at an another line. This shows that the instruction at the line will be executed next.

You can also step through instructions of a specified function.

For example, to step through the function 1, enter:

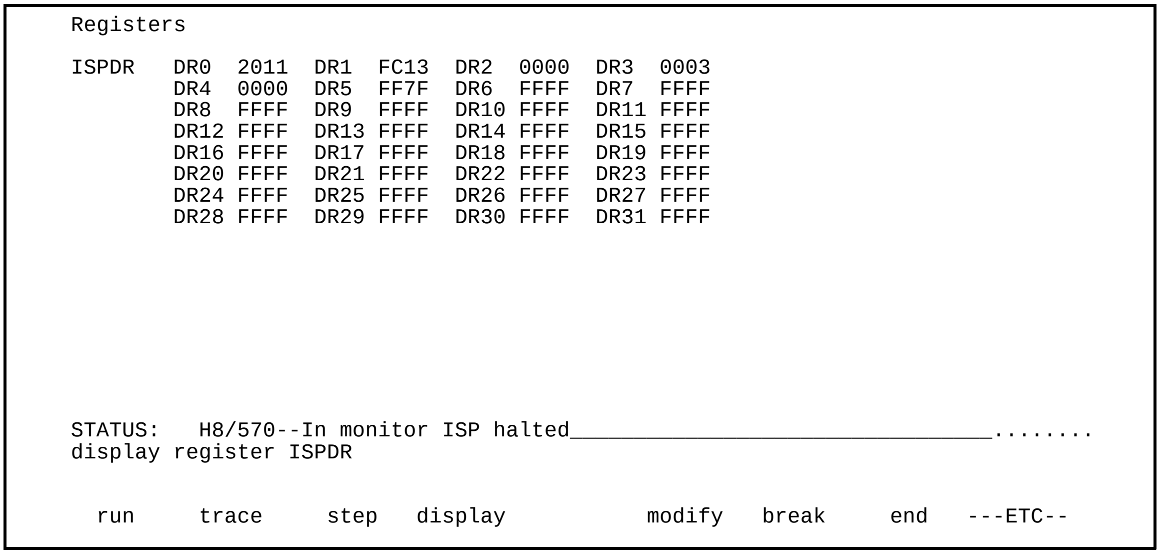
```
step isp function 1 <RETURN>, <RETURN>,
<RETURN>,....
```

Every time you enter the above command, the emulator will run the ISP until an instruction of the specified function is executed.

Displaying/ Modifying ISP Registers

You can display/modify ISP registers. Registers are grouped in several "register classes." For example, to display ISP data registers, use the **ISPDR** register class as follows:

```
display register ISPDR <RETURN>
```



```
Registers

ISPDR   DR0  2011  DR1  FC13  DR2  0000  DR3  0003
        DR4  0000  DR5  FF7F  DR6  FFFF  DR7  FFFF
        DR8  FFFF  DR9  FFFF  DR10 FFFF  DR11 FFFF
        DR12 FFFF  DR13 FFFF  DR14 FFFF  DR15 FFFF
        DR16 FFFF  DR17 FFFF  DR18 FFFF  DR19 FFFF
        DR20 FFFF  DR21 FFFF  DR22 FFFF  DR23 FFFF
        DR24 FFFF  DR25 FFFF  DR26 FFFF  DR27 FFFF
        DR28 FFFF  DR29 FFFF  DR30 FFFF  DR31 FFFF










STATUS:   H8/570--In monitor ISP halted_________________________________........
display register ISPDR


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

You can use the "register name" to display/modify registers. For example, to modify ISP data register 31, use the **DR31** register name as follows:

```
modify register DR31 to 0 <RETURN>
```

Note

Modifying registers in the **ISPSCM** register class is not allowed while the ISP is running. Displaying and modifying registers in the **ISPDR** register class is not allowed while the ISP is running.

Refer to the Chapter 6 of this manual for the list of register classes and names.

Using the Analyzer to Debug ISP Functions

Tracing ISP Execution

You can configure the emulator to trace execution of the CPU, or ISP, or both of them. To configure the emulator to trace only execution of ISP, type:

```
modify configuration <RETURN>
```

Answer the configuration questions as follows:

```
Micro-processor clock source? internal
Enter monitor after configuration? yes
Restrict to real-time runs? no
Modify memory configuration? no
Modify emulator pod configuration? no
Modify debug/trace options? yes
Break processor on write to ROM? yes
Trace CPU or ISP operation by emulation analyzer? isp
Trace refresh cycles by emulation analyzer? no
Modify simulated I/O configuration? no
Modify interactive measurement specification? no
Configuration file name? trace_isp
```

To start the trace when the instruction at ISP address 6 hex, enter the following command:

```
trace after ispaddr 6 <RETURN>
```

Run the sample program:

```
run from Init <RETURN>
```

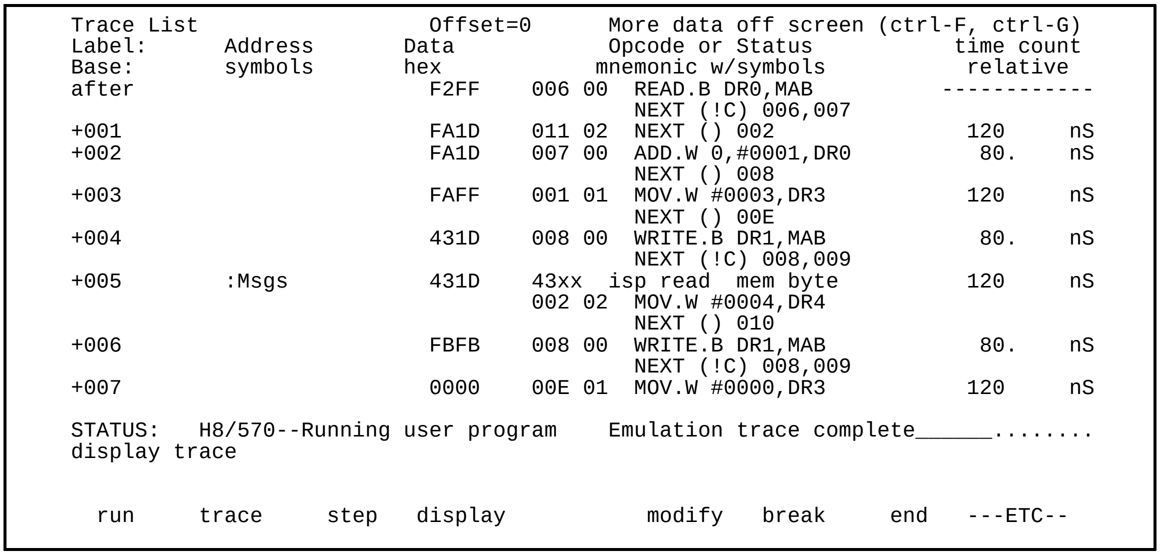
Modify memory to let the ISP function jump to the address specified by the **trace** command.

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

Now display the trace list:

```
display trace <RETURN>
set symbols on <RETURN>
```

You will see a display similar to the following:



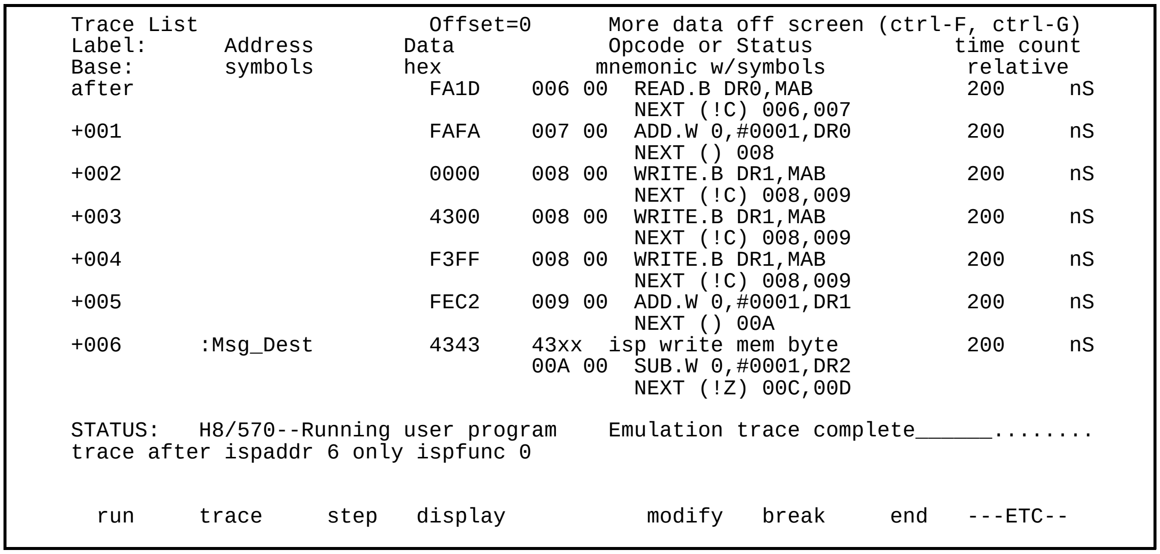
```
Trace List                Offset=0      More data off screen (ctrl-F, ctrl-G)
Label:      Address       Data             Opcode or Status          time count
Base:       symbols       hex             mnemonic w/symbols          relative
after                       F2FF    006 00  READ.B DR0,MAB          ------------
                                            NEXT (!C) 006,007
+001                        FA1D    011 02  NEXT () 002               120     nS
+002                        FA1D    007 00  ADD.W 0,#0001,DR0          80.    nS
                                            NEXT () 008
+003                        FAFF    001 01  MOV.W #0003,DR3           120     nS
                                            NEXT () 00E
+004                        431D    008 00  WRITE.B DR1,MAB            80.    nS
                                            NEXT (!C) 008,009
+005        :Msgs           431D    43xx  isp read  mem byte          120     nS
                                    002 02  MOV.W #0004,DR4
                                            NEXT () 010
+006                        FBFB    008 00  WRITE.B DR1,MAB            80.    nS
                                            NEXT (!C) 008,009
+007                        0000    00E 01  MOV.W #0000,DR3           120     nS

STATUS:   H8/570--Running user program    Emulation trace complete______........
display trace


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

The first column in the mnemonic field shows address of ISP microprogram memory. The second column is function number of the instruction. The third column is the mnemonic of the instruction executed.

As you can see in the above trace listing. the analyzer was triggered by an instruction at address 6.

You also can use ISP function number for trace specification. For example, to trace only execution of ISP function 0, enter:

```
trace after ispaddr 6 only ispfunc 0 <RETURN>
modify memory Cmd_Input bytes to 41h <RETURN>
```



```
Trace List                 Offset=0      More data off screen (ctrl-F, ctrl-G)
Label:      Address       Data              Opcode or Status           time count
Base:       symbols       hex             mnemonic w/symbols            relative
after                      FA1D    006 00  READ.B DR0,MAB               200     nS
                                           NEXT (!C) 006,007
+001                       FAFA    007 00  ADD.W 0,#0001,DR0            200     nS
                                           NEXT () 008
+002                       0000    008 00  WRITE.B DR1,MAB              200     nS
                                           NEXT (!C) 008,009
+003                       4300    008 00  WRITE.B DR1,MAB              200     nS
                                           NEXT (!C) 008,009
+004                       F3FF    008 00  WRITE.B DR1,MAB              200     nS
                                           NEXT (!C) 008,009
+005                       FEC2    009 00  ADD.W 0,#0001,DR1            200     nS
                                           NEXT () 00A
+006      :Msg_Dest        4343    43xx  isp write mem byte             200     nS
                                   00A 00  SUB.W 0,#0001,DR2
                                           NEXT (!Z) 00C,00D

STATUS:   H8/570--Running user program    Emulation trace complete______........
trace after ispaddr 6 only ispfunc 0


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

As you can see, only instructions of ISP function 0 were traced.

Tracing CPU/ISP Execution

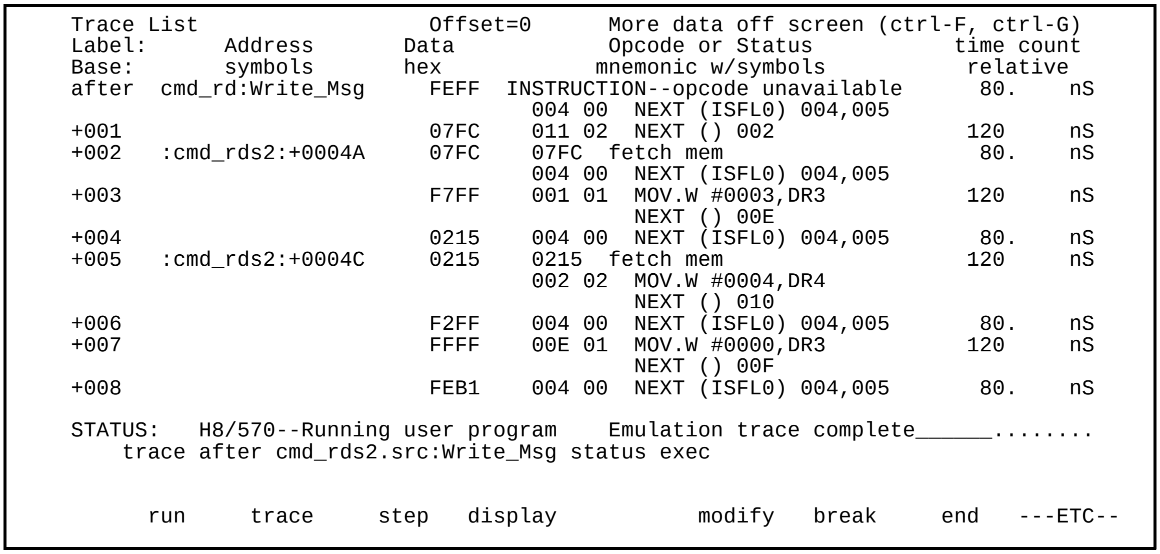
To trace execution of both CPU and ISP, configure the emulator as follows:

```
modify configuration <RETURN>
```

```
Micro-processor clock source? internal
Enter monitor after configuration? yes
Restrict to real-time runs? no
Modify memory configuration? no
Modify emulator pod configuration? no
Modify debug/trace options? yes
Break processor on write to ROM? yes
Trace CPU or ISP operation by emulation analyzer? both
Trace refresh cycles by emulation analyzer? no
Modify simulated I/O configuration? no
Modify interactive measurement specification? no
Configuration file name? trace_both
```

To trace all states after the instruction at **Write_Msg** label is executed, enter:

```
trace after cmd_rds2.src:Write_Msg status
exec <RETURN>
modify memory Cmd_Input bytes to 41h <RETURN>
```



```
Trace List                  Offset=0      More data off screen (ctrl-F, ctrl-G)
Label:      Address       Data            Opcode or Status           time count
Base:       symbols       hex            mnemonic w/symbols           relative
after  cmd_rd:Write_Msg     FEFF  INSTRUCTION--opcode unavailable      80.    nS
                                    004 00  NEXT (ISFL0) 004,005
+001                        07FC    011 02  NEXT () 002               120     nS
+002   :cmd_rds2:+0004A     07FC    07FC  fetch mem                    80.    nS
                                    004 00  NEXT (ISFL0) 004,005
+003                        F7FF    001 01  MOV.W #0003,DR3           120     nS
                                            NEXT () 00E
+004                        0215    004 00  NEXT (ISFL0) 004,005       80.    nS
+005   :cmd_rds2:+0004C     0215    0215  fetch mem                   120     nS
                                    002 02  MOV.W #0004,DR4
                                            NEXT () 010
+006                        F2FF    004 00  NEXT (ISFL0) 004,005       80.    nS
+007                        FFFF    00E 01  MOV.W #0000,DR3           120     nS
                                            NEXT () 00F
+008                        FEB1    004 00  NEXT (ISFL0) 004,005       80.    nS

STATUS:   H8/570--Running user program    Emulation trace complete______........
    trace after cmd_rds2.src:Write_Msg status exec


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

The examples in this chapter is not complete description of each ISP debug commands. Refer to Appendix A of this manual for more detail.

4

In-Circuit Emulation

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

This chapter will:

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

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

Prerequisites

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

Installing the Target System Probe

Caution

DAMAGE TO THE EMULATOR CIRCUITRY MAY RESULT IF THESE PRECAUTIONS ARE NOT OBSERVED. The following precautions should be taken while using the H8/570 emulator.

Power Down Target System. Turn off power to the user target system and to the H8/570 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 H8/570 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 H8/570 emulator; when powering down, turn off the emulator first, then turn off power to the target system.

Target System Adaptor

The HP 64730 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 H8/570 microprocessor.

Pin Protector

The HP 64730 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 short 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 pin protector as shown in Figure 4-1.

Note

You can order additional target system adaptor and short pin protector with part number 64732-61613 and 64732-61614, respectively. Contact your local HP sales representative to purchase additional adaptor and protector.

Optional Pin Extender

If the target system probe is installed on a densely populated circuit board, there may not be a enough room to accommodate the plastic shoulders of the probe. If this occurs, you can use optional long pin protector and pin extender to avoid the conjunction with the target system components. Order the long pin protector and the pin extenders with part number 64732-61615 and 64732-61616, respectively.

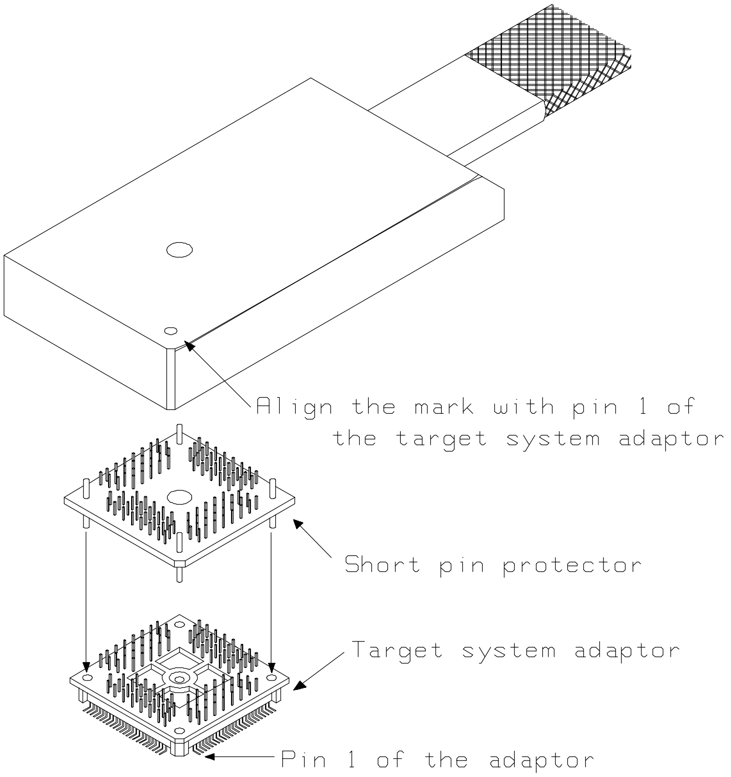


Figure 4-1. Installing the Emulation Probe

Target System Interface

Refer to the *H8/570 Terminal Interface User's Guide* for information on the target system interface of the emulator.

In-Circuit Configuration Options

The H8/570 emulator provides configuration options for the following in-circuit emulation issues. Refer to Chapter 5 for more information on these configuration options.

Using the Target System Clock Source

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

Enabling Bus Arbitration

You can configure the emulator to enable/disable bus arbitration.

Enabling NMI from the Target

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

Enabling /RES from the Target

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

Enabling /RES Output to the Target

You can configure the emulator to drive the /RES on emulation reset or watchdog timer reset.

Selecting Visible/Hidden Background Cycles

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

Running the Emulator from Target Reset

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

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

To specify a run from target system reset, select:

```
run from reset <RESET>
```

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

5

Configuring the Emulator

Introduction

The H8/570 emulator can be used in all stages of target system development. For instance, you can run the emulator out-of-circuit when developing target system software, or you can use the emulator in-circuit when integrating software with target system hardware. Emulation memory can be used in place of, or along with, target system memory. You can use the emulator's internal clock or the target system clock. You can execute target programs in real-time or allow emulator execution to be diverted into the monitor when commands request access of target system resources (target system memory, register contents, etc.)

The emulator is a flexible instrument and it may be configured to suit your needs at any stage of the development process. This chapter describes the options available when configuring the H8/570 emulator.

The configuration options are accessed with the following command.

```
   modify configuration <RETURN>
```

After entering the command above, you will be asked questions regarding the emulator configuration. The configuration questions are listed below and grouped into the following classes.

General Emulator Configuration:

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

Memory Configuration:

- Mapping memory.

Emulator Pod Configuration:

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

Debug/Trace Configuration:

- Enabling breaks on writes to ROM.
- Selecting the trace mode.
- Specifying tracing of foreground/background cycles.
- Enabling tracing refresh cycles.
- Enabling tracing bus release cycles.

Simulated I/O Configuration: Simulated I/O is described in the *Simulated I/O* reference manual.

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

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

General Emulator Configuration

The configuration questions described in this section involve general emulator operation.

Micro-processor clock source?

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

internal	Selects the internal clock oscillator as the emulator clock source. The emulators' internal clock speed is 10 MHz (system clock).
external	Selects the clock input to the emulator probe from the target system. You must use a clock input conforming to the specifications for the H8/570 microprocessor. The maximum external clock speed is 12 MHz (system clock).

Note

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

Enter monitor after configuration?

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

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

When an external clock source is specified, this question becomes "Enter monitor after configuration (using external clock)?" and the default answer becomes "no".

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

no — After the configuration is complete, the emulator will be held in the reset state.

Restrict to real-time runs?

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

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

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

Commands Not Allowed when Real-Time Mode is Enabled

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

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

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

Breaking out of Real-Time Execution

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

```
reset
run
break
step
```

Memory Configuration

The memory configuration questions allows you to map memory. To access the memory configuration questions, you must answer "yes" to the following question.

Modify memory configuration?

Mapping Memory

The H8/570 emulator contains high-speed emulation memory (no wait states required) that can be mapped at a resolution of 128 bytes.

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

Blocks of memory can also be characterized as guarded memory. Guarded memory accesses will generate "break to monitor" requests. Writes to ROM will generate "break to monitor" requests if the "Enable breaks on writes to ROM?" configuration item is enabled (see the "Debug/Trace Configuration" section which follows).

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

Note

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

Note

The default emulator configuration maps location 0 hex through 7FFF hex as emulation ROM, and location F680 hex through FE7F hex as emulation RAM. You cannot delete the term for the internal RAM (F680 hex through FE7F hex).

Note

The emulator uses 4K bytes of emulation memory, and the rest of the emulation memory is available for user program.

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

Note

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

Emulator Pod Configuration

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

Modify emulator pod configuration?

Processor operation mode?

This configuration defines operation mode in which the emulator works.

external — The emulator will work using the mode setting by the target system. The target system must supply appropriate input to MD0, MD1 and MD2. If you are using the emulator out of circuit when "external" is selected, the emulator will operate in mode 1.

When mode_1 through mode_6 is selected, the emulator will operate in selected mode regardless of the mode setting by the target system.

Selection	Description
mode_1	The emulator will operate in mode 1. (expanded minimum mode with 16 bit data bus)
mode_3	The emulator will operate in mode 3. (expanded maximum mode with 16 bit data bus)
mode_4	The emulator will operate in mode 4. (expanded minimum mode with 8 bit data bus)
mode_5	The emulator will operate in mode 5. (expanded maximum mode with 16 bit data bus)
mode_6	The emulator will operate in mode 6. (expanded maximum mode with 8 bit data bus)

Enable bus arbitration?

The bus arbitration configuration question defines how your emulator responds to bus request signals from the target system during foreground operation. The /BREQ signal from the target system is always ignored when the emulator is running the background monitor. This configuration item is only available for the H8/570 emulator.

yes

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

Note

You cannot perform DMA (direct memory access) transfers between your target system and emulation memory by using DMA controller on your target system; the H8/570 emulator does not support such a feature.

no

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

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

Enable NMI input from the target system?

This configuration allows you to specify whether or not the emulator responds to NMI (non-maskable interrupt request) signal from the target system while user program is running.

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

The emulator does not accept any interrupt while it is running in monitor. NMI is latched last one during in monitor, and such interrupt will occur when context is changed to user program. /IRQ0 and internal interrupts are ignored during in monitor operation.

Enable /RES input from the target system?

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

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

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

Note

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

Drive emulation reset to the target system?

This question is asked when you answer "yes" to the previous question. This configuration allows you to select whether or not the emulator will drive the /RES signal to the target system during emulation reset and reset by the Watchdog timer.

no Specifies that the emulator will not drive the /RES signal during emulation reset and reset by the Watchdog timer. The configuration of RSTOE (Reset output enable bit) is ignored.

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

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

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, answering "yes" to this configuration may result in damage to target system or emulation circuitry.

Drive background cycles to the target system?

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

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

yes Specifies that background cycles are driven to the target system. Emulation processor's address and control strobes (except /HWR and /LWR) are driven during background cycles.

Background write cycles won't appear to the target system.

Note

Memory cycles by the ISP are driven to the target system while the emulator is in the monitor.

Break ISP into halt state on CPU break?

This configuration allows you to select whether the emulator halts the ISP when the emulator breaks into the monitor.

yes

The emulator halts the ISP when the "**break**" command is issued.

no

The emulator doesn't halt the ISP when the "**break**" command is issued. You can halt the ISP by specifying the "**with_isp**" syntax in the "**break**" command.

Reset value for stack pointer?

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

The address specified in response to this question must be a 24-bit hexadecimal even address.

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

- Odd address
- Internal I/O register address

Note ☝

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

Debug/Trace Configuration

The debug/trace configuration questions allows you to specify breaks on writes to ROM, and specify that the analyzer trace foreground/background execution, and bus release cycles. To access the trace/debug configuration questions, you must answer "yes" to the following question.

Modify debug/trace options?

Break processor on write to ROM?

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

yes — Causes the emulator to break into the emulation monitor whenever the user program attempts to write to a memory region mapped as ROM.

no — The emulator will not break to the monitor upon a write to ROM. The emulator will not modify the memory location if it is in emulation ROM.

Note

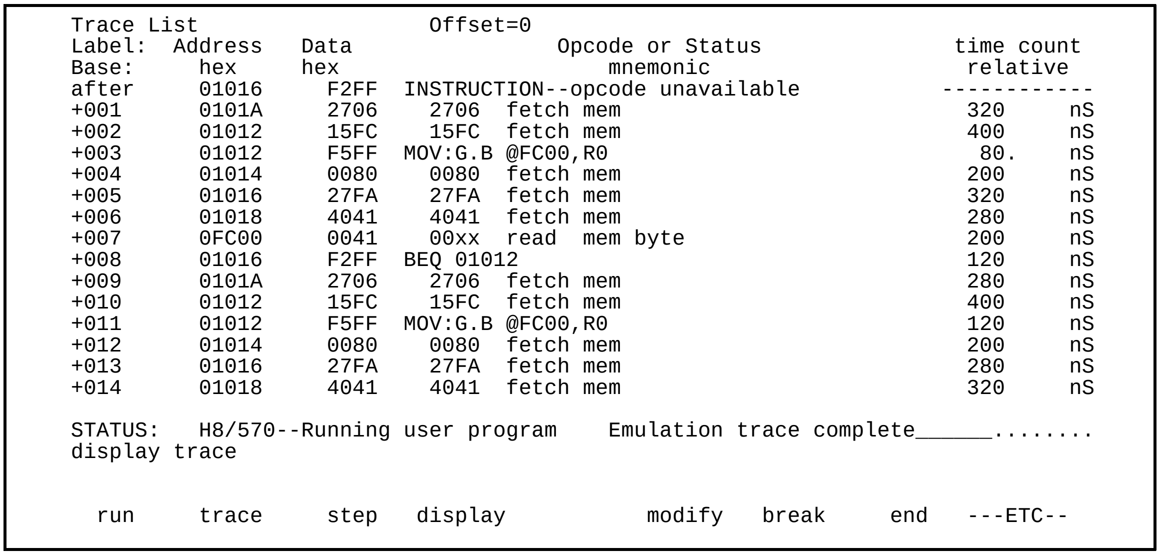
The **wrrom** trace command status options allow you to use "write to ROM" cycles as trigger and storage qualifiers. For example, you could use the following command to trace about a write to ROM: **trace about status wrrom** < RETURN>

Trace CPU or ISP operation by emulation analyzer

This configuration allows you to select the trace mode. The emulation analyzer can trace execution of CPU or ISP or both of them.

cpu

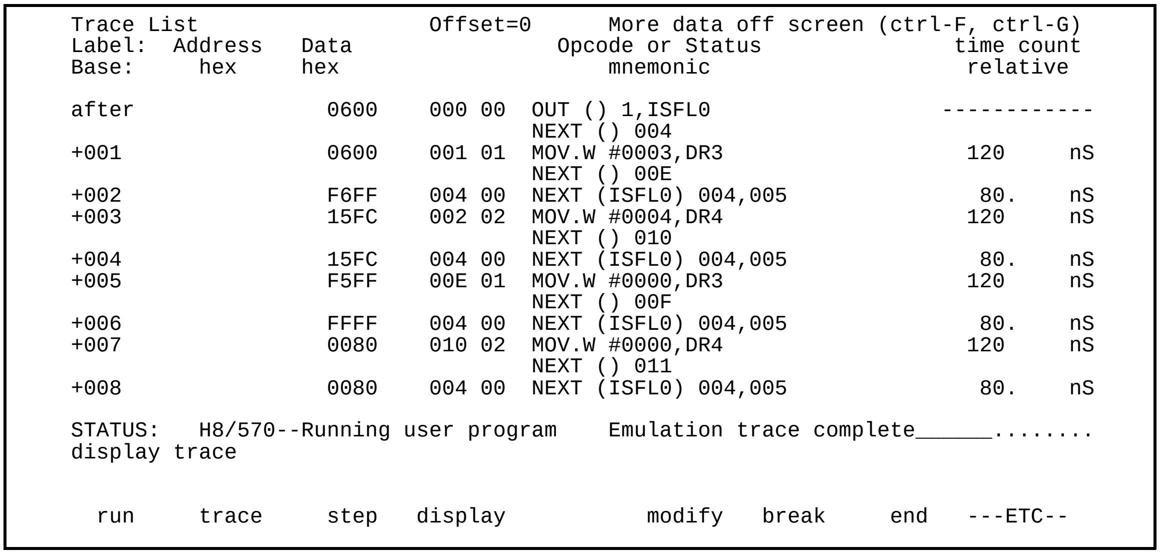
The emulation analyzer doesn't trace ISP execution. The following is a sample trace listing of this trace mode.



```
Trace List                    Offset=0
Label:  Address   Data                 Opcode or Status               time count
Base:     hex     hex                      mnemonic                     relative
after     01016     F2FF  INSTRUCTION--opcode unavailable           ------------
+001      0101A     2706    2706  fetch mem                              320     nS
+002      01012     15FC    15FC  fetch mem                              400     nS
+003      01012     F5FF  MOV:G.B @FC00,R0                                80.    nS
+004      01014     0080    0080  fetch mem                              200     nS
+005      01016     27FA    27FA  fetch mem                              320     nS
+006      01018     4041    4041  fetch mem                              280     nS
+007      0FC00     0041    00xx  read  mem byte                         200     nS
+008      01016     F2FF  BEQ 01012                                      120     nS
+009      0101A     2706    2706  fetch mem                              280     nS
+010      01012     15FC    15FC  fetch mem                              400     nS
+011      01012     F5FF  MOV:G.B @FC00,R0                               120     nS
+012      01014     0080    0080  fetch mem                              200     nS
+013      01016     27FA    27FA  fetch mem                              280     nS
+014      01018     4041    4041  fetch mem                              320     nS

STATUS:   H8/570--Running user program    Emulation trace complete______........
display trace


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

isp

The emulation analyzer traces only ISP execution and memory cycles by the ISP. The following is a sample listing of this trace mode.

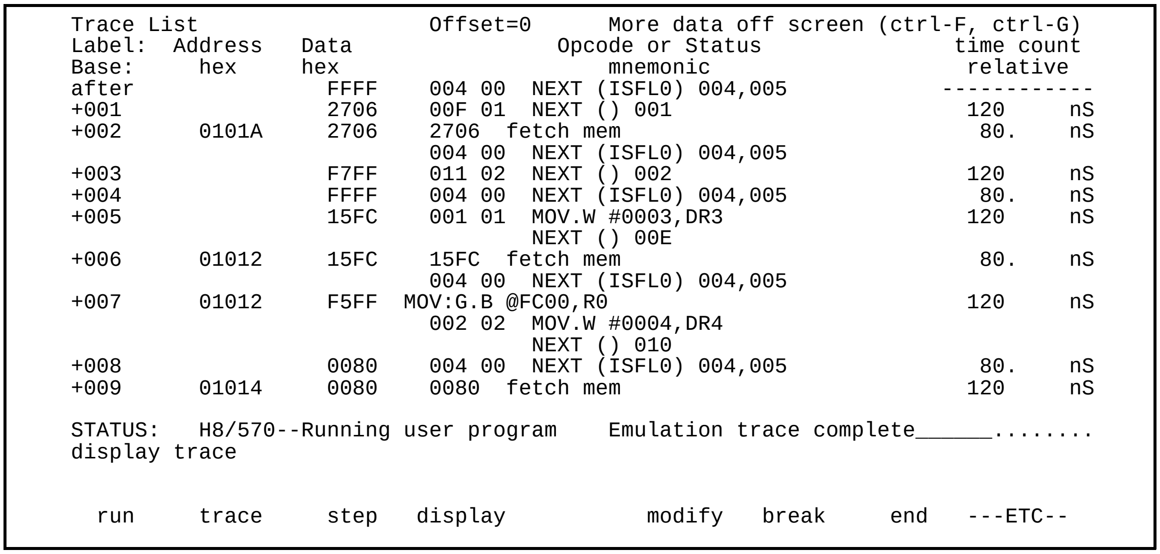


```
Trace List                  Offset=0      More data off screen (ctrl-F, ctrl-G)
Label:  Address   Data                Opcode or Status               time count
Base:     hex     hex                     mnemonic                    relative

after               0600    000 00  OUT () 1,ISFL0                  ------------
                                    NEXT () 004
+001                0600    001 01  MOV.W #0003,DR3                   120     nS
                                    NEXT () 00E
+002                F6FF    004 00  NEXT (ISFL0) 004,005               80.    nS
+003                15FC    002 02  MOV.W #0004,DR4                   120     nS
                                    NEXT () 010
+004                15FC    004 00  NEXT (ISFL0) 004,005               80.    nS
+005                F5FF    00E 01  MOV.W #0000,DR3                   120     nS
                                    NEXT () 00F
+006                FFFF    004 00  NEXT (ISFL0) 004,005               80.    nS
+007                0080    010 02  MOV.W #0000,DR4                   120     nS
                                    NEXT () 011
+008                0080    004 00  NEXT (ISFL0) 004,005               80.    nS

STATUS:   H8/570--Running user program    Emulation trace complete______........
display trace


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

The first column in the mnemonic field shows address of ISP microprogram memory. The second column is function number of the instruction. The third column is the mnemonic of the ISP instruction executed.

both

The emulation analyzer traces both CPU and ISP execution. The following is a sample listing of this trace mode.



```
Trace List                  Offset=0      More data off screen (ctrl-F, ctrl-G)
Label:  Address   Data                Opcode or Status               time count
Base:     hex     hex                     mnemonic                    relative
after               FFFF    004 00  NEXT (ISFL0) 004,005            ------------
+001                2706    00F 01  NEXT () 001                       120     nS
+002      0101A     2706    2706  fetch mem                            80.    nS
                            004 00  NEXT (ISFL0) 004,005
+003                F7FF    011 02  NEXT () 002                       120     nS
+004                FFFF    004 00  NEXT (ISFL0) 004,005               80.    nS
+005                15FC    001 01  MOV.W #0003,DR3                   120     nS
                                    NEXT () 00E
+006      01012     15FC    15FC  fetch mem                            80.    nS
                            004 00  NEXT (ISFL0) 004,005
+007      01012     F5FF  MOV:G.B @FC00,R0                            120     nS
                            002 02  MOV.W #0004,DR4
                                    NEXT () 010
+008                0080    004 00  NEXT (ISFL0) 004,005               80.    nS
+009      01014     0080    0080  fetch mem                           120     nS

STATUS:   H8/570--Running user program    Emulation trace complete______........
display trace


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

Trace background or foreground operation?

This question is asked when you answer "**cpu**" or "**both**" to the previous question. This question allows you to specify whether the analyzer trace only foreground emulation processor cycles, only background cycles, or both foreground or background cycles. When background cycles are stored in the trace, all but mnemonic lines are tagged as background cycles.

foreground — Specifies that the analyzer trace only foreground cycles. This option is specified by the default emulator configuration.

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

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

Trace refresh cycles?

You can direct the emulator to trace refresh cycles or not.

yes When you enable tracing refresh cycles, the analyzer will trace refresh cycles.

no The analyzer will not trace refresh cycles.

Trace bus release cycles?

You can direct the emulator to send bus release cycle data to emulation analyzer or not to send it.

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

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

Simulated I/O Configuration

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

Interactive Measurement Configuration

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

External Analyzer Configuration

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

Saving a Configuration

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

Configuration file name? < FILE>

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

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

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

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

Loading a Configuration

Configuration files which have been previously saved may be loaded with the following Softkey Interface command.

```
   load configuration <FILE> <RETURN>
```

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

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

```
   load configuration <RETURN>
```

Notes

6

Using the Emulator

Introduction

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

This chapter discusses:

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

synchronous instruction.

This chapter shows you how to:

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

Features Available via Pod Commands

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

```
display pod_command <RETURN>
pod_command '<Terminal Interface command>'
<RETURN>
```

Some of the most notable Terminal Interface features not available in the softkey Interface are:

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

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

Note

Be careful when using the "pod_command". The Softkey Interface, and the configuration files in particular, assume that the configuration of the HP 64700 pod is NOT changed except by the Softkey Interface. Be aware that what you see in "modify configuration" will NOT reflect the HP 64700 pod's configuration if you change the pod's configuration with this command. Also, commands which affect the communications channel should NOT be used at all. Other commands may confuse the protocol depending upon how they are used. The following commands are not recommended for use with "pod_command":

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

Using a Command File

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

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

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

```
load configuration cmd_rds <RETURN>
```

To load the program into memory:

```
load cmd_rds <RETURN>
```

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

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

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

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

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

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

Debugging C Programs

Softkey Interface has following functions to debug C programs.

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

The following section describes such features.

Displaying Memory with C Sources

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

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

You can display source lines highlighted with the following command.

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

To display only source lines, use the following command.

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

Specifying Address with Line Numbers

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

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

Displaying Trace with C Sources

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

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

Stepping C Sources

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

```
step source from _main <RETURN>
```

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

```
step source <RETURN>
```

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

```
step 5 source <RETURN>
```

E clock synchronous instructions

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

```
display io_port
modify io_port
```

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

Limitations, Restrictions

DMA Support

Direct memory access to H8/570 emulation memory is not permitted.

Sleep and Software Stand-by Mode

When the emulator breaks into the monitor (foreground/background), the H8/570 sleep or software stand-by mode is released and comes to normal processor mode.

Watchdog Timer

When the emulator breaks into background, the emulation processor’s watchdog timer suspends count up in background cycles.

Address Error and Register Values

In operation of the H8/570 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)

ISP Microprogram Modify

The contents of ISP microprogram memory cannot be modified by emulation commands. To modify your ISP program, you need to re-assemble/link your program, and load it into the emulator.

Symbolic Information for ISP Functions

The H8/570 Softkey Interface does not support symbolic information for ISP functions. No symbolic information for ISP functions is dispalyed in ISP memory display and trace listing.

RAM Enable Bit

The internal RAM of H8/510 processor can be enabled/disabled by RAME (RAM enable bit). However, the H8/570 emulator accesses emulation RAM even if the internal RAM is disabled by RAME.

Storing Memory Contents to an Absolute File

The "Getting Started" chapter shows you how to load absolute files into emulation or target system memory. You can also store emulation or target system memory to an absolute file with the following command.

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

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

Coordinated Measurements

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

Register Names and Classes

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

Summary

H8/570 register designators. All available register class names and register names are listed below.

BASIC Class

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

SYS Class

System control registers

Register name	Description
WCR	Wait control register
MDCR	Mode control register
SBYCR	Software stand-by control register
RAMCR	RAM control register
SYSCR1	System control register 1

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

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

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

PORT Class

I/O port registers

Register name	Description
P1DDR	Port 1 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
P10DDR	Port 10 data direction register
P11DDR	Port 11 data direction register
P12DDR	Port 12 data direction register
P1DR	Port 1 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
P10DR	Port 10 data register
P11DR	Port 11 data register
P12DR	Port 12 data register

PWM Class

PWM timer registers

TCR	Timer control register
TSR	Timer status register
ODL	Output data latch
ODR0	Output data register 0
ODR1	Output data register 1
ODR2	Output data register 2
OCR0	Output compare register 0
OCR1	Output compare register 1
OCR2	Output compare register 2
TMR	Timer

WDT Class

Watchdog timer registers

Register name	Description
WDTCSR	Timer control/status register
WDTCNT	Timer counter
RSTCSR	Reset control/status register

SCI Class

Serial communication interface registers.

RDR	Receive data register
TDR	Transmit data register
SMR	Serial mode register
SCR	Serial control register
SSR	Serial status register
BRR	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 C
ADDRD	A/D data register D
ADCSR	A/D control/status register
ADCR	A/D control register

ISPSCM Class

ISP SCM

Register name	Description
AR0	ISP address register 0
AR1	ISP address register 1
AR2	ISP address register 2
:	:
:	:
AR9	ISP address register 9
AR10	ISP address register 10
AR11	ISP address register 11

ISPDR Class

ISP data registers

DR0	ISP data register 0
DR1	ISP data register 1
DR2	ISP data register 2
DR3	ISP data register 3
:	:
:	:
DR30	ISP data register 30
DR31	ISP data register 31

ISPF Class

ISP flags

ICF	Interconnction flag
IOF0	Input/output flag 0
IOF1	Input/output flag 1
IOF2	Input/output flag 2
EGF	Edge flag
ISF	Interrupt status flag

ISPC Class

ISP control registers

Register name	Description
IEF	Interrupt enable flag
IOIEF	I/O interrupt enable flag
CLE	Clear enable register
EVER	Event enable register
IPR	ISP page register
ICSR	ISP control status register
REDGE	Rising edge enable register
FEDGE	Falling edge enable register
SYSCR8	System control register 8
SYSCR9	System control register 9
SYSCR10	System control register 10

Using the Format Converter

Description

The format converter is a program that generates HP format files from a HP 64869 format file. This means you can use available language tools to create HP 64869 format file, then load the file into the emulator.

Synopsis

To execute the converter program, use the following command:

```
$ h8cnvhp [options] <file_name>
```

< file_name> is the name of HP 64869 format file without suffix. The converter program will read the HP 64869 format file (with .abs suffix). It will generate the following HP format files:

- HP Absolute file (with .X suffix)
- HP Linker symbol file (with .L suffix)
- HP Assembler symbol file (with .A suffix)

Options

THe following options are available:

-x	create the absolute file
-l	create the linker symbol file
-a	create the assembler symbols files. The HP 64869 format file must contain local symbol information.

Example

Suppose that you have the following file:

```
sample.abs (HP 64869 format file)
```

You can generate HP format files from this file with the following command:

```
$ h8cnvhp sample <RETURN>
```

A

H8/570 Softkey Interface Specific Syntax

This appendix describes specific syntax of H8/570 Softkey Interface.

Items explained in this appendix includes:

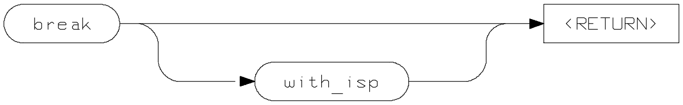
- Syntax of **break** command
- Syntax of **display isp_memory** command
- Syntax of **display trace** command
- Syntax of **run** command
- Syntax of **step** command

The explanation in this appendix is addendum to the *Softkey Interface Reference* manual. Refer to the manual for complete description of each command.

break

This command causes the emulator to leave user program execution and begin executing in the monitor.

Syntax



Function

The behavior of **break** depends on the state of the emulator:

running	Break diverts the processor from execution of your program to the emulation monitor. The ISP execution is halted if you specify the **with_isp** syntax, or you configure the emulator to halt the ISP on break.
reset	Break releases the processor from reset, and diverts execution to the monitor. The ISP is held at the halt state.
running in monitor	The **break** command does not perform any operation to the processor. The ISP is halted if you specify the **with_isp** syntax, or you configure the emulator to halt the ISP on break.
In monitor ISP halted	The **break** command does not perform any operation.

Parameters

with_isp	This allows you to halt the ISP. By default, you don't have to specify this parameter to halt the ISP. When you configure the emulator not to halt the ISP on emulation break, you need to specify this parameter to halt the ISP.

Example

```
break <RETURN>
break with_isp <RETURN>
```

Related Commands

help ***break***

modify configuration

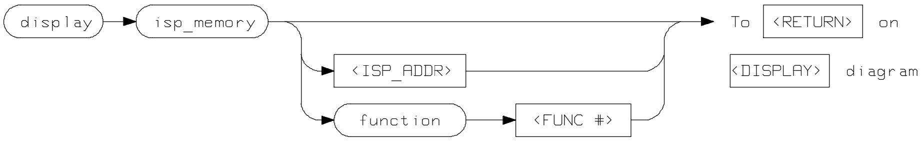
run

step

display isp_memory

Displays the contents of the ISP microprogram memory in mnemonic format.

Syntax



Function

display isp_memory can display the contents of the ISP microprogram memory in mnemonic format. You can specify a function number to display instructions of an ISP function.

Note

No symbolic information is displayed in ISP memory display.

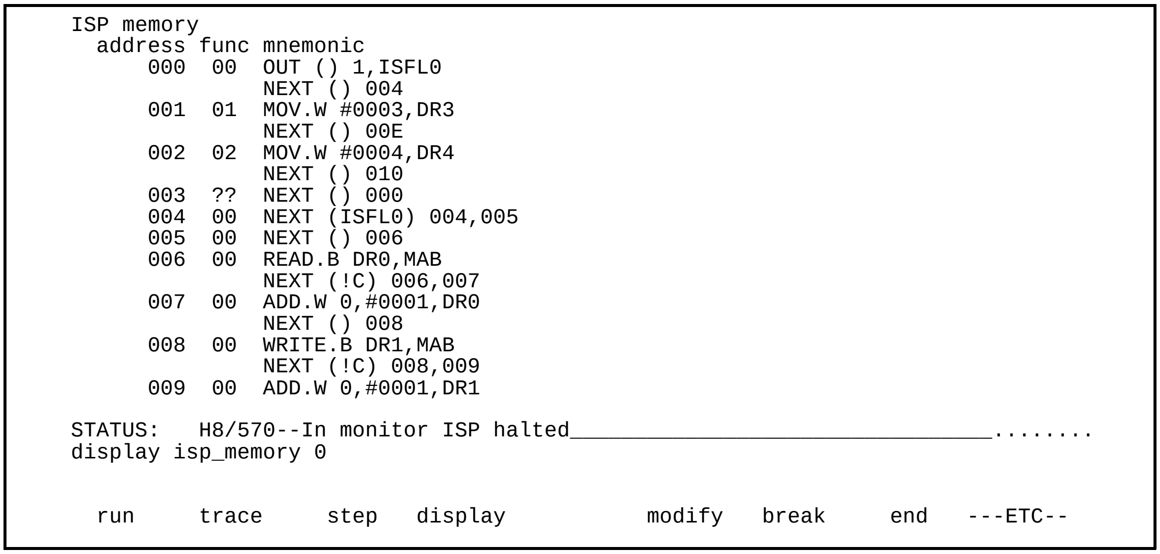
Parameters

< ISP_ADDR>	The start address to be displayed.
function	This allows you to specify a function number to be displayed.

Examples

```
display isp_memory 0 <RETURN>
```

The result of this command may resemble:

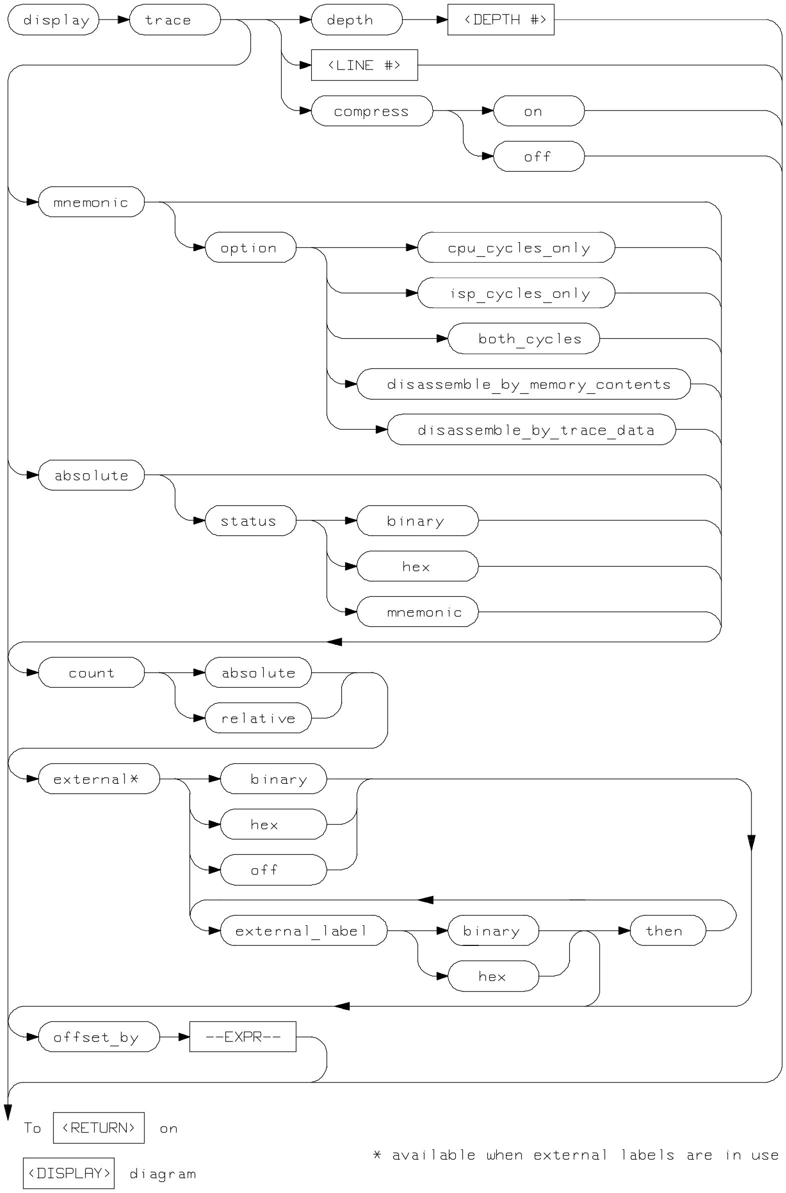


```
ISP memory
  address func mnemonic
      000  00  OUT () 1,ISFL0
               NEXT () 004
      001  01  MOV.W #0003,DR3
               NEXT () 00E
      002  02  MOV.W #0004,DR4
               NEXT () 010
      003  ??  NEXT () 000
      004  00  NEXT (ISFL0) 004,005
      005  00  NEXT () 006
      006  00  READ.B DR0,MAB
               NEXT (!C) 006,007
      007  00  ADD.W 0,#0001,DR0
               NEXT () 008
      008  00  WRITE.B DR1,MAB
               NEXT (!C) 008,009
      009  00  ADD.W 0,#0001,DR1

STATUS:   H8/570--In monitor ISP halted_________________________________........
display isp_memory 0


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

display trace

This command displays the contents of the trace buffer.

Syntax



Function

You can specify to display CPU instruction or ISP instructions or both of them.

Parameters

cpu_cycles_only — When you configure the emulator to trace both of CPU and ISP cycles, the display may too complex to find information you need. In this case, you can display only CPU cycles by specifying this option.

isp_cycles_only — displays ISP cycles only.

both_cycles — displays both of CPU cycles and ISP cycles.

disassemble_by_memory_contents

Use data in memory to disassemble the trace data. By default, the emulator disassembles by data in the trace buffer to display the trace listing. Therefore, if you specify the **exec** status for the store condition, the emulator cannot disassemble the trace data. When this option is specified, the emulator can disassemble the trace even if the **exec** is specified for store condition. This would be useful when you don’t have to see any memory cycles.

disassemble_by_trace_data

Use data in the trace buffer to disassemble.

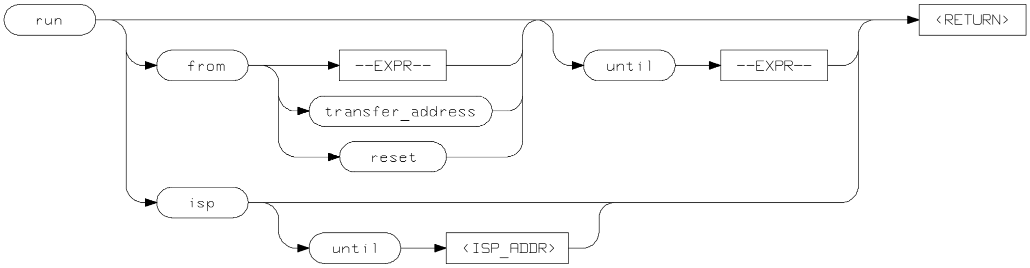
Note

When you specify the **disassemble_by_memory_contents** syntax, the emulator may need to suspend user program execution to see the contents of target memory.

run

This command causes the emulator to execute a program or ISP function.

Syntax



Function

The **run isp** command causes the ISP to start execution.

Parameters

isp — Allows you to cause the ISP to start execution.

until — Allows you to cause the ISP to start execution, and halts the execution after the instruction at the specified address is executed.

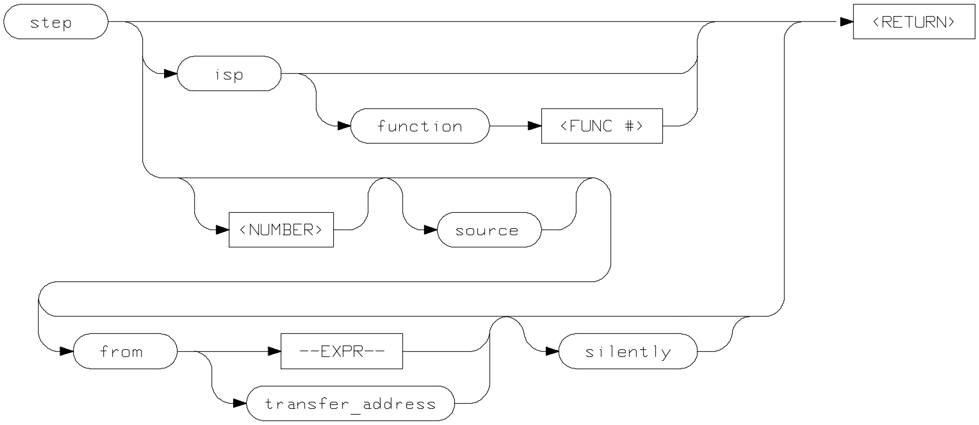
Examples

```
run isp
run isp until 12
```

step

The **step** command allows you sequential analysis of program instructions by causing the emulation processor or ISP to execute a specified number of instructions.

Syntax



Function

You can step ISP instructions. You also can step through instructions of a specified ISP function.

Parameters

isp — Allows you to step ISP instructions.

function — Allows you to step through instructions of a specified ISP functions. When you specify this option, the emulator runs the ISP until an instruction of the specified function is executed. Instructions of other functions are also executed until the emulator halts ISP after an instruction of the specified function is executed.

Notes

Index

Notes