User's Guide for the Graphical User Interface

HP 64798 MC6830x Emulation/Analysis

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 1996 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.

HP is a trademark of Hewlett-Packard Company.

OSF/Motif and Motif are trademarks of the Open Software Foundation in the U.S. and other countries.

SunOS, SPARCsystem, OpenWindows, and SunView are trademarks of Sun Microsystems, Inc.

UNIX® is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Hewlett-Packard
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 as 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. Rights for non-DOD U.S. Government Departments 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 B3093-97000, July 1996

6830x Emulation and Analysis

The HP 64798 6830x emulators replace the microprocessor in your embedded microprocessor system, also called the *target system*, so that you can control execution and view or modify processor and target system resources.

The emulator requires an *emulation analyzer* that captures 64 channels of emulation processor bus cycle information synchronously with the processor's clock signal. The HP 64703 Emulation Bus Analyzer meets this requirement.

The HP 64703 Emulation Bus Analyzer also has an an *external analyzer* that captures up to 16 channels of data external to the emulator. You can also use the HP 64704 or HP 64794 Emulation Bus Analyzer which has 80 channels; however, these analyzers do not have external analysis channels.

With the Emulator, You Can ...

- Plug into 6830x target systems.
- Download programs into emulation memory or target system RAM.
- Display or modify the contents of processor registers and memory resources.
- Run programs, set up software breakpoints, step through programs, and reset the emulation processor. For information about your emulator clock speed, refer to the *6830x Installation/Service/Terminal Interface User's Guide*.

With the Analyzer, You Can ...

- Trigger the analyzer when a particular bus cycle state is captured. States are stored relative to the trigger state.
- Qualify which states get stored in the trace.
- Prestore certain states that occur before each normal store state.
- Trigger the analyzer after a sequence of up to 8 events have occurred.
- Capture data on signals of interest in the target system with the external analyzer.

- Cause emulator execution to break when the analyzer finds its trigger condition.

With the HP 64700 Card Cage, You Can ...

- Use the RS-422 capability of the serial port and an RS-422 interface card on the host computer (HP 98659 for the HP 9000 Series 300) to provide upload/download rates of up to 230.4K baud.
- Easily access and use the emulator over a Local Area Network by connecting it to the LAN.
- Easily upgrade HP 64700 firmware by downloading to flash memory.

With Multiple HP 64700s, You Can ...

- Start and stop up to 16 emulators at the same time (up to 32 if modifications are made).
- Use the analyzer in one HP 64700 to arm (activate) the analyzers in other HP 64700 card cages or to cause emulator execution in other HP 64700 card cages to break.
- Use the HP 64700's BNC connector to trigger an external instrument (for example, a logic analyzer or oscilloscope) when the analyzer finds its trigger condition, or you can allow an external instrument to arm the analyzer or break emulator execution.

With the Graphical User Interface, You Can ...

- Use the emulator and analyzer under an X Window System that supports OSF/Motif interfaces.
- Enter commands using pull-down or pop-up menus.
- Enter, recall, and edit commands using the command line pushbuttons.
- Enter file names, recalled commands, recalled values, etc., using dialog boxes.
- Set breakpoints by pointing the mouse cursor on a line in the mnemonic memory display and clicking.
- Create action keys for commonly used commands or command files.

With the Softkey Interface, You Can ...

- Use the emulator and analyzer with a terminal or terminal emulator to execute terminal interface commands.
- Quickly enter commands using softkeys, command recall, and command editing.

In This Book

This book documents the Graphical User Interface and the Softkey Interface when used with the HP 64798 6830x emulators and the HP 64703/4 analyzer. It is organized into five parts whose chapters are described below.

Part 1. Quick Start Guide

Chapter 1 presents an overview of emulation and analysis and quickly shows you how to use the emulator and analyzer.

Part 2. User's Guide

Chapter 2 tells where to find information about plugging the emulator into a target system.

Chapter 3 shows you how to start and exit the HP 64700 interfaces.

Chapter 4 shows you how to enter commands.

Chapter 5 shows you how to configure the emulator.

Chapter 6 shows you how to use the emulator.

Chapter 7 shows you how to use the analyzer.

Chapter 8 shows you how to use the Software Performance Measurement Tool (SPMT) with the analyzer.

Chapter 9 shows you how to use the external state analyzer.

Chapter 10 shows you how to make coordinated measurements.

Chapter 11 shows you how to change X resource settings.

Part 3. Reference

Chapter 12 describes emulator/analyzer interface commands.

Chapter 13 lists the error messages that can occur.

Part 4. Concept Guide

Chapter 14 contains conceptual information on various topics.

Part 5. Installation Guide

Chapter 15 outlines the installation of the Graphical User Interface.

Chapter 16 shows you how to install or update emulator firmware. Follow these instructions if you have ordered the HP 64782 emulator and the HP 64700 Card Cage separately.

Contents

Part 1

Quick Start Guide

Part 1

A one-glance overview of the product and a few task instructions to help you get comfortable.

1

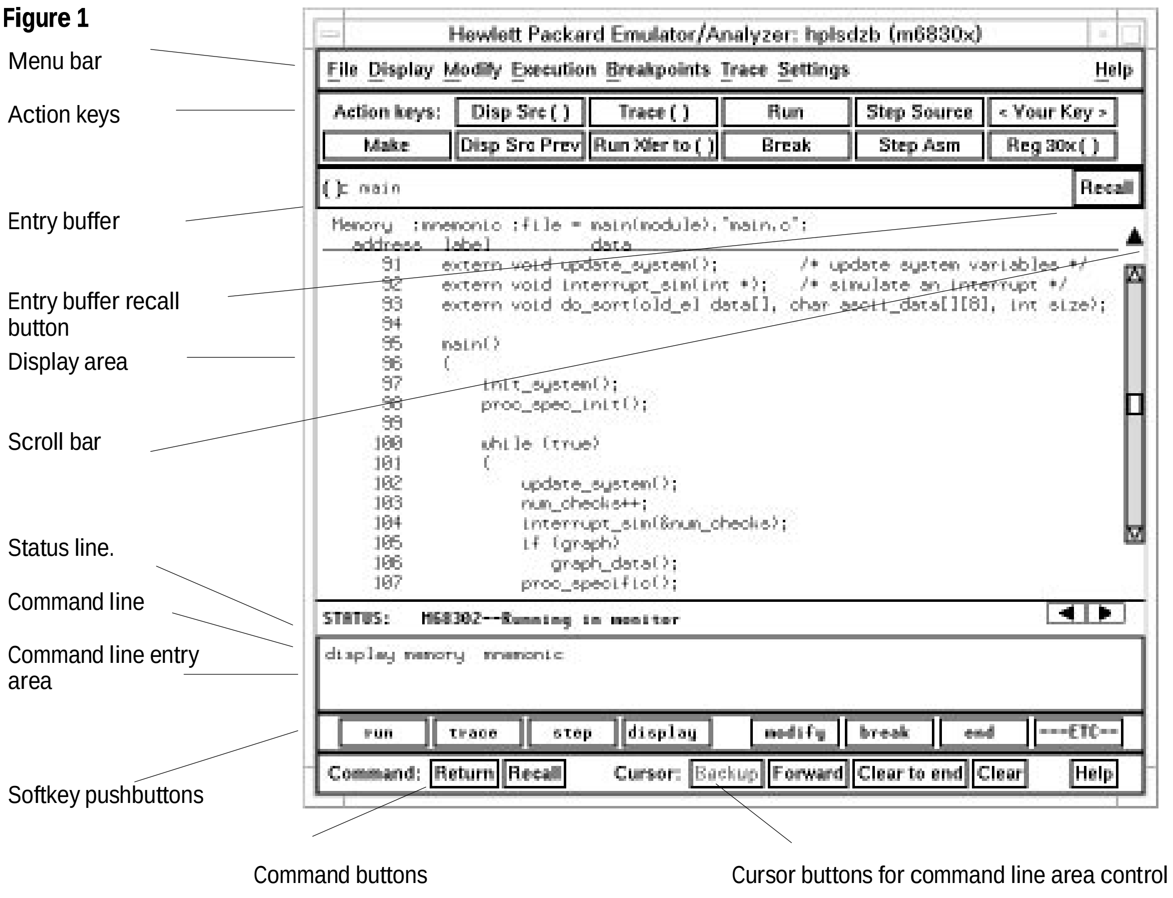
Getting Started

The Emulator/Analyzer Interface — At a Glance

When an X Window System that supports OSF/Motif interfaces is running on the host computer, the emulator/analyzer interface is the Graphical User Interface which provides pull-down and pop-up menus, point and click setting of breakpoints, cut and paste, online help, customizable action keys and pop-up recall buffers, etc.

The Graphical User Interface

Figure 1


Hewlett Packard Emulator/Analyzer: hplsdzb (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x( )
Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help
Menu bar
Action keys
Entry buffer
Entry buffer recall button
Display area
Scroll bar
Status line.
Command line
Command line entry area
Softkey pushbuttons
Command buttons
Cursor buttons for command line area control

```
( ): main
```

```
Memory  :mnemonic :file = main(module),"main.c":
   address  label           data
       91    extern void update_system();       /* update system variables */
       92    extern void interrupt_sim(int *);   /* simulate an interrupt */
       93    extern void do_sort(old_e) data[], char ascii_data[][8], int size);
       94
       95    main()
       96    (
       97        init_system();
       98        proc_spec_init();
       99
      100        while (true)
      101        (
      102            update_system();
      103            num_checks++;
      104            interrupt_sim(&num_checks);
      105            if (graph)
      106               graph_data();
      107            proc_specific();
```

```
STATUS:    M68302--Running in monitor
```

```
display memory  mnemonic
```

Menu Bar. Provides pull-down menus from which you select commands. When menu items are not applicable, they appear half-bright and do not respond to mouse clicks.

Action Keys. User-defined pushbuttons. You can label these pushbuttons and define the action to be performed.

Entry Buffer. Wherever you see "( )" in a pull-down menu, the contents of the entry buffer are used in that command. You can type values into the entry buffer, or you can cut and paste values into the entry buffer from the display area or from the command line entry area. You can also set up action keys to use the contents of the entry buffer.

Entry Buffer Recall Button. Allows you to recall entry buffer values that have been predefined or used in previous commands. When you click on the entry buffer Recall button, a dialog box appears that allows you to select values.

Display Area. Can show memory, data values, analyzer traces, registers, breakpoints, status, simulated I/O, global symbols, local symbols, pod commands (the emulator's underlying Terminal Interface), error log, or display log.

Whenever the mouse pointer changes from an arrow to a hand, you can press and hold the *select* mouse button to access pop-up menus.

Scroll Bar. A "sticky slider" that allows navigation in the display area. Click on the upper and lower arrows to scroll to the top (home) and bottom (end) of the window. Click on the inner arrows to scroll one line. Drag the slider handle up or down to cause continuous scrolling. Click between the inner arrows and the slider handle to page up or page down.

Status Line. Displays the emulator and analyzer status. Also, when error and status messages occur, they are displayed on the status line in addition to being saved in the error log. You can press and hold the *select* mouse button to access the Status Line pop-up menu.

Command Line. The command line area is similar to the command line in the Softkey Interface; however, the graphical interface lets you use the mouse to enter and edit commands.

- Command line entry area. Allows you to enter commands from the command line.

- Softkey pushbuttons. Clicking on these pushbuttons, or pressing softkeys, places the command in the command line entry area. You can press and hold the *select* mouse button to access the Command Line pop-up menu.
- Command buttons (includes command recall button). The command Return button is the same as pressing the carriage return key — it sends the command in the command line entry area to the emulator/analyzer.

  The command Recall button allows you to recall previous or predefined commands. When you click on the command Recall button, a dialog box appears that allows you to select a command.
- Cursor buttons for command line area control. Allow you to move the cursor in the command line entry area forward or backward, clear to the end of the command line, or clear the whole command line entry area.

You can choose not to display the command line area by turning it off. For the most common emulator/analyzer operations, the pull-down menus, pop-up menus, and action keys provide all the control you need. Choosing menu items that require use of the command line will automatically turn the command line back on.

Graphical User Interface Conventions

Choosing Menu Commands

This chapter uses a shorthand notation for indicating that you should choose a particular menu item. For example, the following instruction

Choose File→Load→Configuration

means to first display the File pull-down menu, then display the Load cascade menu, then select the Configuration item from the Load cascade menu.

Based on this explanation, the general rule for interpreting this notation can be stated as follows:

- The leftmost item in bold is the pull-down menu label.
- If there are more than two items, then cascade menus are involved and all items between the first and last item have cascade menus attached.
- The last item on the right is the actual menu choice to be made.

Mouse Button and Keyboard Bindings

Because the Graphical User Interface runs on different kinds of computers, which may have different conventions for mouse buttons and key names, the Graphical User Interface supports different bindings and the customization of bindings.

This manual refers to the mouse buttons using general (or "generic") terms. The following table describes the generic mouse button names and shows the default mouse button bindings.

Mouse Button Bindings and Descriptions

Generic Button Name	**Bindings for HP 9000**	**Bindings for Sun SPARCsystem (SunOS or Solaris)**	**Description**
paste	left	left	Paste from the display area to the entry buffer.
command paste	middle[1]	middle[1]	Paste from the entry buffer to the command line text entry area.
select	right	right	Click selects first item in pop-up menus. Press and hold displays menus.
command select	left	right	Displays pull-down menus.
pushbutton select	left	left	Actuates pushbuttons outside of the display area.

[1]Middle button on three-button mouse. Both buttons on two-button mouse.

The following table shows the default keyboard bindings.

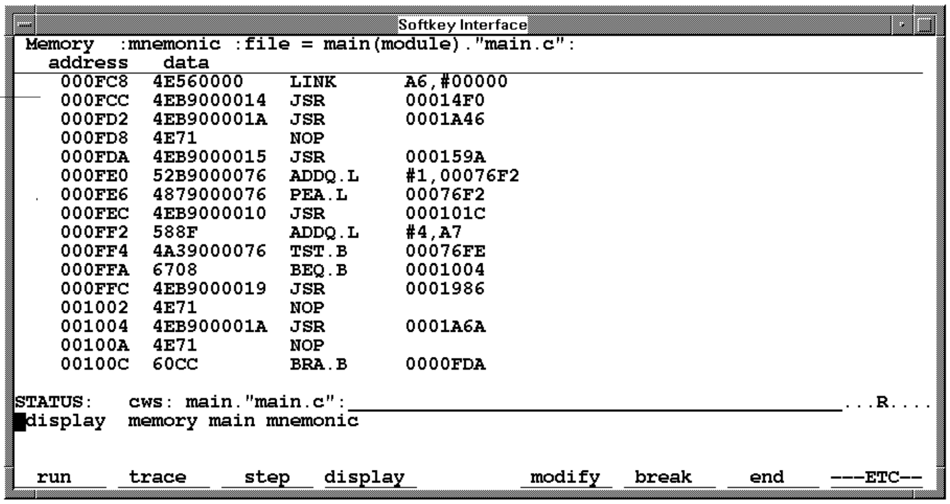
Keyboard Key Bindings

Generic Key Name	HP 9000	Sun SPARCsystem (SunOS or Solaris)
menu select	extend char	extend char
insert	insert char	insert char
delete	delete char	delete char
left-arrow	left arrow	left arrow
right-arrow	right arrow	right arrow
up-arrow	up arrow	up arrow
down-arrow	down arrow	down arrow
escape	escape	escape
TAB	TAB	TAB

The Softkey Interface

The emulator/analyzer interface can also be the Softkey Interface which is provided for several types of terminals, terminal emulators, and bitmapped displays. When using the Softkey Interface, commands are entered from the keyboard.

Figure 2



```
Memory  :mnemonic :file = main(module)."main.c":
  address   data
    000FC8  4E560000    LINK      A6,#00000
    000FCC  4EB9000014  JSR       00014F0
    000FD2  4EB900001A  JSR       0001A46
    000FD8  4E71        NOP
    000FDA  4EB9000015  JSR       000159A
    000FE0  52B9000076  ADDQ.L    #1,00076F2
    000FE6  4879000076  PEA.L     00076F2
    000FEC  4EB9000010  JSR       000101C
    000FF2  588F        ADDQ.L    #4,A7
    000FF4  4A39000076  TST.B     00076FE
    000FFA  6708        BEQ.B     0001004
    000FFC  4EB9000019  JSR       0001986
    001002  4E71        NOP
    001004  4EB900001A  JSR       0001A6A
    00100A  4E71        NOP
    00100C  60CC        BRA.B     0000FDA

STATUS:   cws: main."main.c":________________________________________...R....
display  memory main mnemonic

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

Display area

Can show memory, data values, analyzer traces, registers, breakpoints, status, simulated I/O, global symbols, local symbols, pod commands (the emulator's underlying Terminal Interface), error log, or display log. You can use the UP ARROW, DOWN ARROW, PAGE UP, and PAGE DOWN cursor keys to scroll or page up or down the information in the active window.

Status line

Displays the emulator and analyzer status. Also, when error and status messages occur, they are displayed on the status line in addition to being saved in the error log.

Command line

Commands are entered on the command line by pressing softkeys (or by typing them in) and executed by pressing the Return key. The Tab and Shift-Tab keys allow you to move the cursor on the command line forward or backward. The Clear line key (or CTRL-e) clears from the cursor position to the end of the line. The CTRL-u key clears the whole command line.

Softkey Interface Conventions

Example Softkey Interface 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 UNIX prompt. Commands which follow the "$" are entered at the UNIX prompt.
<RETURN>	The carriage return key.

The Getting Started Tutorial

This tutorial gives you step-by-step instructions on how to perform a few basic tasks using the emulator/analyzer interface. The tutorial examples presented in this chapter make the following assumptions:

- The HP 64798 emulator and HP 64703/4 analyzer are installed into the HP 64700 Card Cage, the HP 64700 is connected to the host computer, and the Graphical User Interface software has been installed as outlined in the "Installation" chapter.
- The emulator is operating out-of-circuit (that is, not plugged into a target system).

The Demonstration Program

The demonstration program used in this chapter is a simple environmental control system. The program controls the temperature and humidity of a room requiring accurate environmental control.

Step 1: Start the demo

A demo program and its associated files are provided with the Graphical User Interface in the directory hp64798.

1 **Change to the demo directory.**

```
$ cd /usr/hp64000/demo/debug_env/hp64798
```

Refer to the README file for more information on the demo program.

2 **Check that "/usr/hp64000/bin" and "." are in your PATH environment variable. To see the value of PATH:**

```
$ echo $PATH
```

3 **If the Graphical User Interface software is installed on a different type of computer than the computer you are using, edit the "platformScheme" resource setting in the "Xdefaults.emul" file.**

For example, if the Graphical User Interface will be run on a HP 9000 computer and displayed on a Sun SPARCsystem computer, change the platform scheme to "SunOS".

4 **Start the emulator/analyzer demo.**

```
$ Startemul em6830x
```

This script starts the emulator/analyzer interface (with a customized set of action keys), loads a configuration file for the demo program, and then loads the demo program.

The <logical_emul_name> in the command above is the logical emulator name given in the HP 64700 emulator device table file (/usr/hp64000/etc/64700tab.net). You can enter the LAN name address instead of the logical emulator name.

Step 2: Display the program in memory

1 If the symbol "main" is not already in the entry buffer, move the mouse pointer to the entry buffer (notice the flashing I-beam cursor) and type in "main".

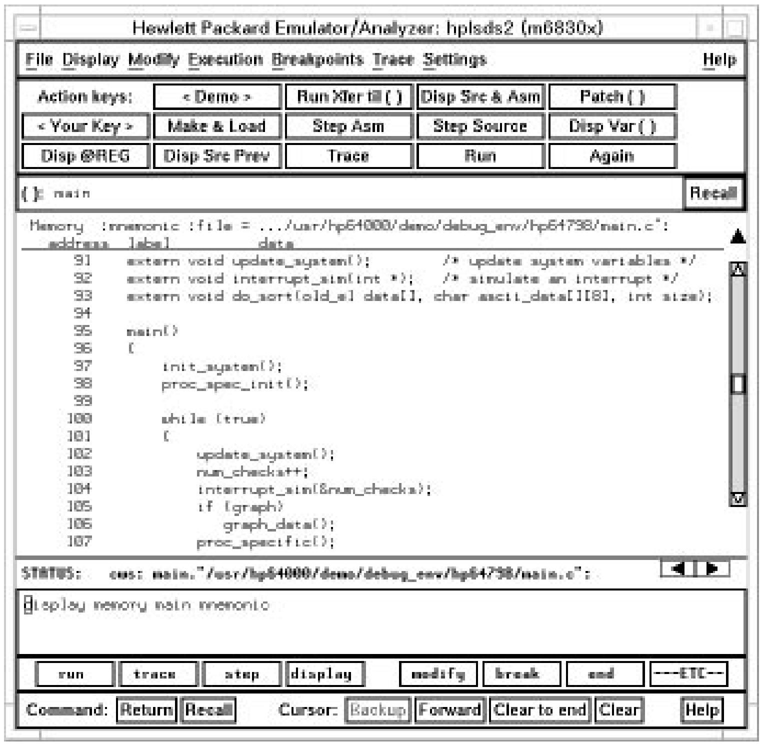
2 Choose **Display→Memory→Mnemonic ()**.

Or, using the command line, enter:

```
display memory main mnemonic
```

Figure 3

Entry buffer



```
( ): main
```

```
Memory  :mnemonic :file = .../usr/hp64000/demo/debug_env/hp64798/main.c":
  address  label          data
     91    extern void update_system();        /* update system variables */
     92    extern void interrupt_sim(int *);    /* simulate an interrupt */
     93    extern void do_sort(old_el data[], char ascii_data[][8], int size);
     94
     95    main()
     96    {
     97        init_system();
     98        proc_spec_init();
     99
    100        while (true)
    101        {
    102            update_system();
    103            num_checks++;
    104            interrupt_sim(&num_checks);
    105            if (graph)
    106               graph_data();
    107            proc_specific();
```

```
STATUS:    cws: main."/usr/hp64000/demo/debug_env/hp64798/main.c":
```

```
display memory main mnemonic
```

The default display mode settings cause source lines and symbols to appear in displays where appropriate. You can use symbols when specifying expressions. The global symbol "main" is used in the command above to specify the starting address of the memory to be displayed.

Step 3: Run from the transfer address

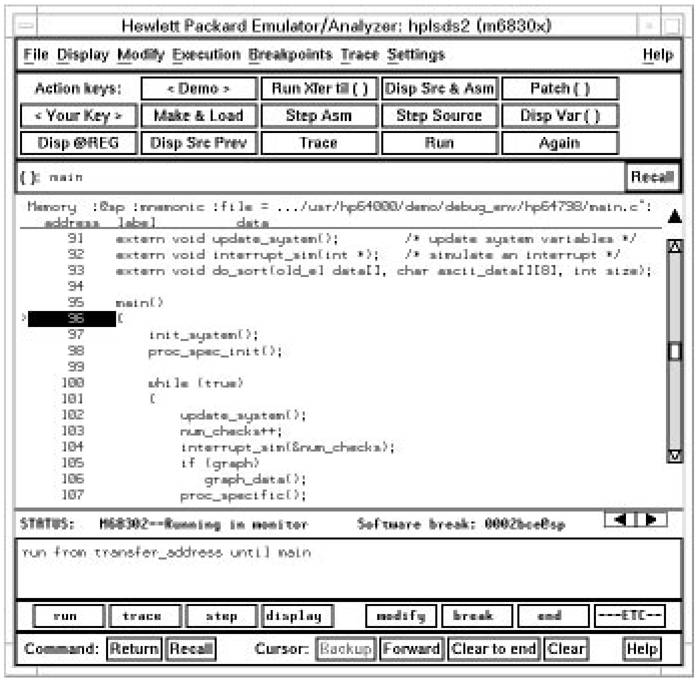
The transfer address is the entry address defined by the software development tools and included with the program's symbol information.

- Click on the **Run Xfer til ()** action key.

Or, using the command line, enter:

run from transfer_address until `main`

Figure 4


Hewlett Packard Emulator/Analyzer: hplsds2 (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: < Demo > Run Xfer til ( ) Disp Src & Asm Patch ( )
< Your Key > Make & Load Step Asm Step Source Disp Var ( )
Disp @REG Disp Src Prev Trace Run Again
Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
( ): main

Memory  :@sp :mnemonic :file = .../usr/hp64000/demo/debug_env/hp64798/main.c":
  address  label         data
      91   extern void update_system();        /* update system variables */
      92   extern void interrupt_sim(int *);   /* simulate an interrupt */
      93   extern void do_sort(old_el data[], char ascii_data[][8], int size);
      94
      95   main()
>     96   {
      97       init_system();
      98       proc_spec_init();
      99
     100       while (true)
     101       {
     102           update_system();
     103           num_checks++;
     104           interrupt_sim(&num_checks);
     105           if (graph)
     106              graph_data();
     107           proc_specific();

STATUS:    M68302--Running in monitor      Software break: 0002bce@sp

run from transfer_address until main
```

Notice the message "Software break: <address>" is displayed on the status line and that the emulator is "Running in monitor" (you may have to click the *select* mouse button to remove temporary messages from the status line). When you run until an address, a breakpoint is set at the address before the program is run.

Notice the highlighted bar on the screen; it shows the current program counter.

Step 4: Step high-level source lines

You can step through the program by high-level source lines. The emulator executes as many instructions as are associated with the high-level program source lines.

1 **To step a source line from the current program counter, click on the Step Source action key.**

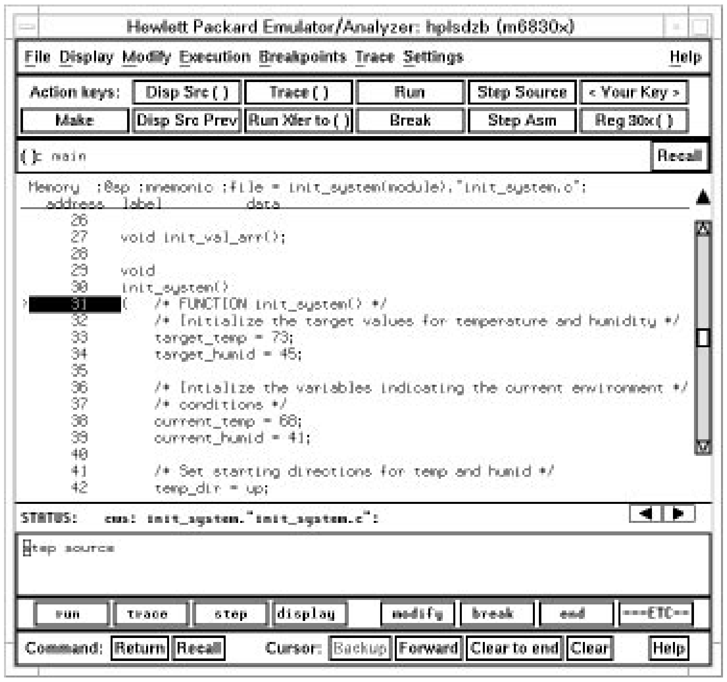
Or, using the command line, enter:

```
step source
```

Notice that the highlighted bar (the current program counter) moves to the next high-level source line.

2 **Step into the "init_system" function by continuing to step source lines, either by clicking on the Step Source action key, by clicking on the Again action key which repeats the previous command, or by entering the step source command on the command line.**

Figure 5


Hewlett Packard Emulator/Analyzer: hplsdzb (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x( )
Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
( ): main

Memory  :@sp :mnemonic :file = init_system(module)."init_system.c":
  address  label         data
      26
      27    void init_val_arr();
      28
      29    void
      30    init_system()
>     31    {    /* FUNCTION init_system() */
      32         /* Initialize the target values for temperature and humidity */
      33         target_temp = 73;
      34         target_humid = 45;
      35
      36         /* Intialize the variables indicating the current environment */
      37         /* conditions */
      38         current_temp = 68;
      39         current_humid = 41;
      40
      41         /* Set starting directions for temp and humid */
      42         temp_dir = up;

STATUS:    cms: init_system."init_system.c":

step source
```

Step 5: Display the previous mnemonic display

- Click on the **Disp Src Prev** action key.

Or, using the command line, enter:

```
display memory mnemonic previous_display
```

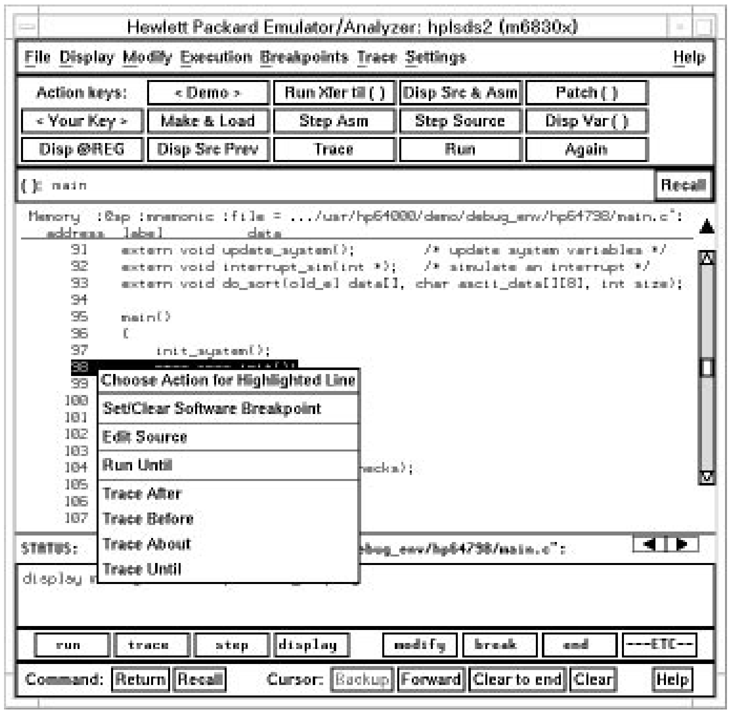
This command is useful, for example, when you have stepped into a function that you do not wish to look at—you can display the previous mnemonic display and run until the source line that follows the function call.

Step 6: Run until an address

When displaying memory in mnemonic format, a selection in the pop-up menu lets you run from the current program counter address until a specific source line.

- Position the mouse pointer over the line "proc_spec_init();", press and hold the *select* mouse button, and choose **Run Until** from the pop-up menu. This screen shows the command line turned on.

Figure 6


Hewlett Packard Emulator/Analyzer: hplsds2 (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: < Demo > Run Xfer til ( ) Disp Src & Asm Patch ( )
< Your Key > Make & Load Step Asm Step Source Disp Var ( )
Disp @REG Disp Src Prev Trace Run Again
Recall
Choose Action for Highlighted Line
Set/Clear Software Breakpoint
Edit Source
Run Until
Trace After
Trace Before
Trace About
Trace Until
STATUS:
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
( ): main
Memory  :@sp :mnemonic :file = .../usr/hp64000/demo/debug_env/hp64798/main.c":
  address  label         data
      91    extern void update_system();         /* update system variables */
      92    extern void interrupt_sim(int *);    /* simulate an interrupt */
      93    extern void do_sort(old_e) data[], char ascii_data[][8], int size);
      94
      95    main()
      96    {
      97        init_system();
      98
      99
     100
     101
     102
     103
     104                                   acks);
     105
     106
     107
STATUS:                                  ebug_env/hp64798/main.c":
display m
```

Or, using the command line, enter:

```
run until main."main.c": line 98
```

After the command has executed, notice the highlighted bar indicates the program counter has moved to the specified source line.

Step 7: Display data values

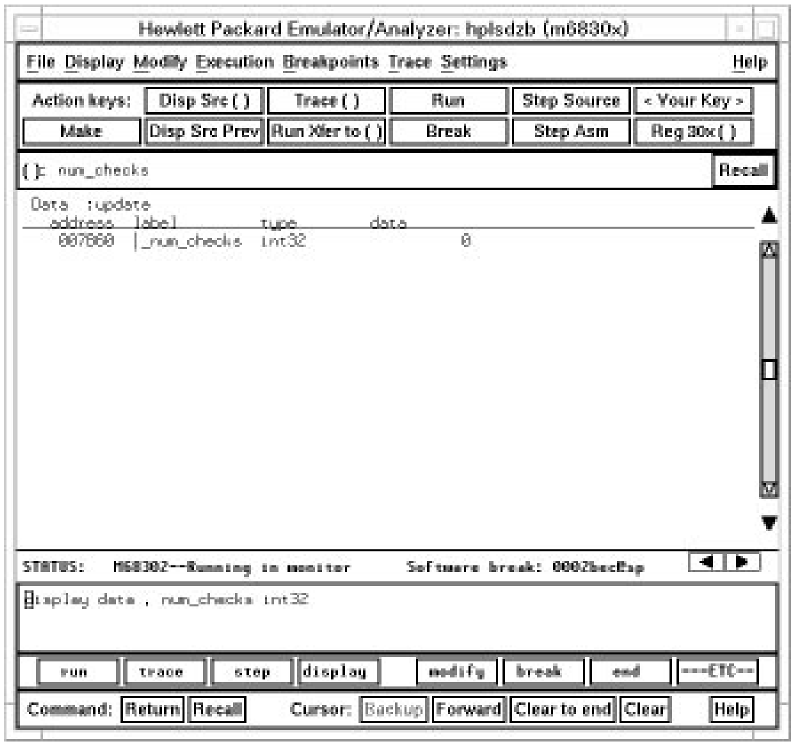
1 Position the mouse pointer over "num_checks" in the source line that reads "num_checks++;" and click the *paste* mouse button (notice "num_checks" is cut and pasted into the entry buffer).

2 Choose **Display→Data Values**

Or, using the command line, enter:

display data , `num_checks` ***int32***

Figure 7


Hewlett Packard Emulator/Analyzer: hplsdzb (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x( )
( ): num_checks
Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
Data   :update
  address  label          type       data
   007860  |_num_checks  int32                 0
```

```
STATUS:   M68302--Running in monitor      Software break: 0002becBsp
```

```
display data , num_checks int32
```

The "num_checks" variable is added to the data values display and its value is displayed as a 32-bit integer.

Step 8: Display registers

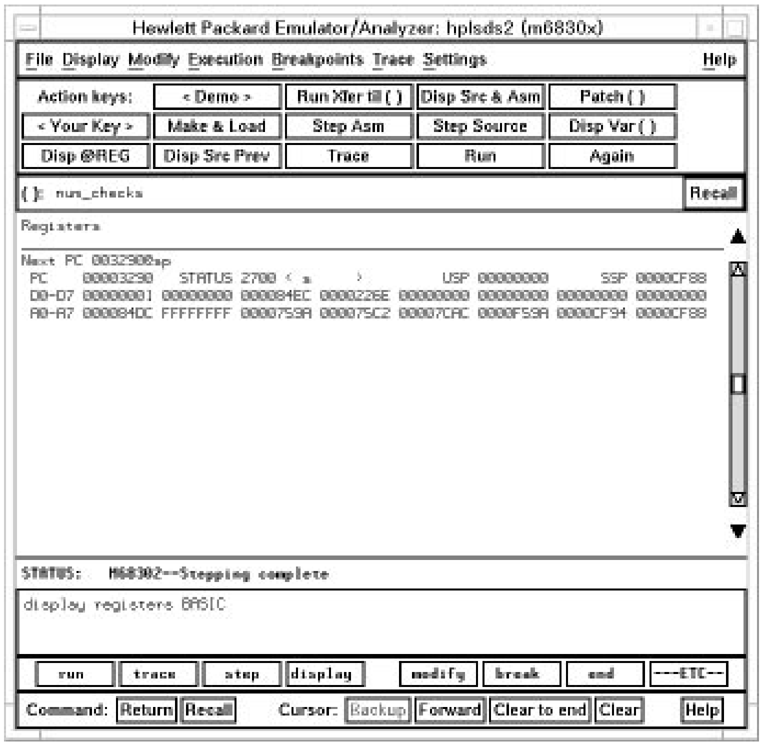
You can display the contents of the processor registers.

- Choose **Display→Registers→BASIC**.

Or, using the command line, enter:

`display registers`

Figure 8


Hewlett Packard Emulator/Analyzer: hplsds2 (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: < Demo > Run Xfer til ( ) Disp Src & Asm Patch ( )
< Your Key > Make & Load Step Asm Step Source Disp Var ( )
Disp @REG Disp Src Prev Trace Run Again
( ): num_checks Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
Registers
Next PC 0032900sp
 PC     00003290   STATUS 2700 < s      >          USP 00000000      SSP 0000CF88
 D0-D7 00000001 00000000 000084EC 0000226E 00000000 00000000 00000000 00000000
 A0-A7 000084DC FFFFFFFF 0000759A 000075C2 00007CAC 0000F59A 0000CF94 0000CF88

STATUS:   M68302--Stepping complete

display registers BASIC
```

Step 9: Step assembly-level instructions

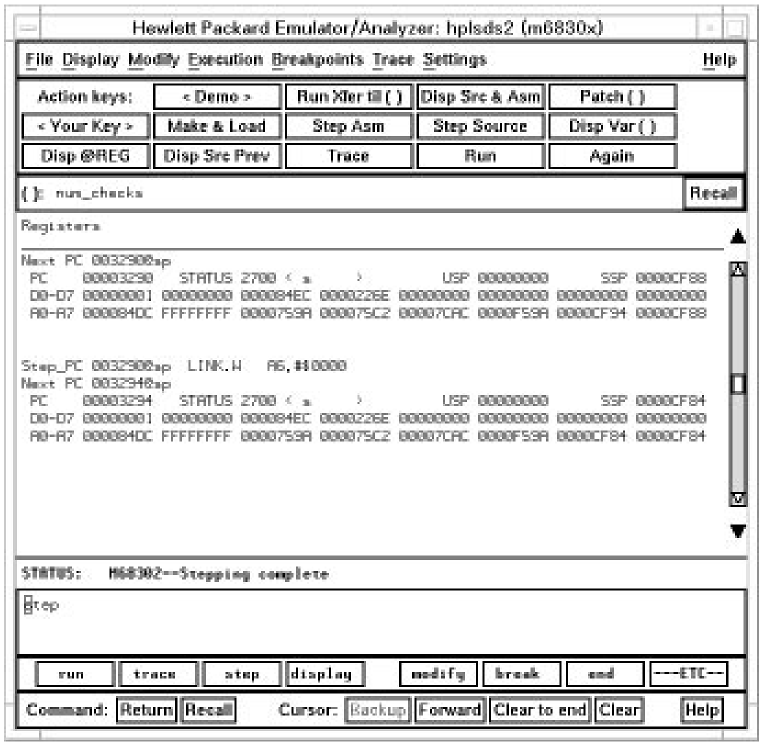
You can step through the program one instruction at a time.

- To step one instruction from the current program counter, click on the **Step Asm** action key.

Or, using the command line, enter:

`step`

Figure 9


Hewlett Packard Emulator/Analyzer: hplsds2 (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: < Demo > Run Xfer til ( ) Disp Src & Asm Patch ( )
< Your Key > Make & Load Step Asm Step Source Disp Var ( )
Disp @REG Disp Src Prev Trace Run Again
( ): num_checks
Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
Registers
Next PC 0032900@sp
 PC     00003290   STATUS 2700 < s      >         USP 00000000       SSP 0000CF88
 D0-D7 00000001 00000000 000084EC 0000226E 00000000 00000000 00000000 00000000
 A0-A7 000084DC FFFFFFFF 0000759A 000075C2 00007CAC 0000F59A 0000CF94 0000CF88


Step_PC 0032900@sp  LINK.W   A6,#0000
Next PC 0032940@sp
 PC     00003294   STATUS 2700 < s      >         USP 00000000       SSP 0000CF84
 D0-D7 00000001 00000000 000084EC 0000226E 00000000 00000000 00000000 00000000
 A0-A7 000084DC FFFFFFFF 0000759A 000075C2 00007CAC 0000F59A 0000CF84 0000CF84
```

```
STATUS:    M68302--Stepping complete
```

```
step
```

Notice, when registers are displayed, stepping causes the assembly language instruction just executed to be displayed.

Step 10: Trace the program

When the analyzer traces program execution, it looks at the data on the emulation processor’s bus and control signals at each clock cycle. The information seen at a particular clock cycle is called a state.

When one of these states matches the "trigger state" you specify, the analyzer stores states in trace memory. When trace memory is filled, the trace is said to be "complete."

1 Click on the **Recall** button to the right of the entry buffer.

A selection dialog box appears. You can select from entry buffer values that have been entered previously or that have been predefined.

2 Click on "main" in the selection dialog box, and click the "OK" pushbutton.

Notice that the value "main" has been returned to the entry buffer.

3 To trigger on the address "main" and store states that occur after the trigger, choose **Trace→After ()**.

Or, using the command line, enter:

```
trace after main
```

Notice the message "Emulation trace started" appears on the status line. This shows that the analyzer has begun to look for the trigger state which is the address "main" on the processor’s address bus.

4 Run the emulator demo program from its transfer address by choosing **Execution→Run→from Transfer Address**.

Or, using the command line, enter:

```
run from transfer_address
```

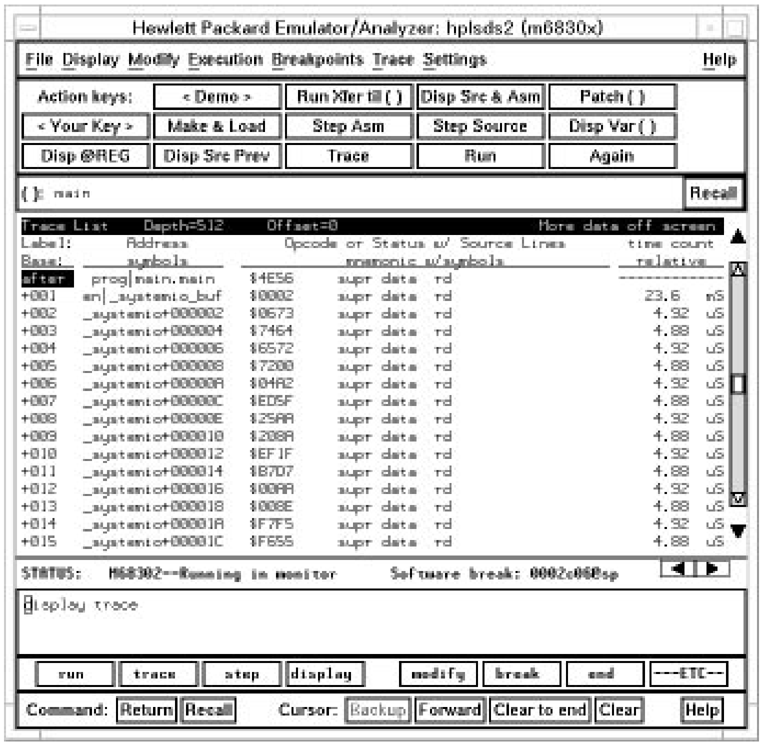
Notice that now the message on the status line is "Emulation trace complete". This shows the trigger state has been found and the analyzer trace memory has been filled.

5 To view the captured states, choose **Display→Trace**.

Or, using the command line, enter:

`display trace`

Figure 10



```
Trace List      Depth=512      Offset=0                     More data off screen
Label:        Address          Opcode or Status w/ Source Lines          time count
Base:         symbols                   mnemonic w/symbols                relative
after     prog|main.main      $4E56      supr data  rd                 ------------
+001    mn|_systemio_buf      $0002      supr data  rd                     23.6   mS
+002    _systemio+000002      $0673      supr data  rd                     4.92   uS
+003    _systemio+000004      $7464      supr data  rd                     4.88   uS
+004    _systemio+000006      $6572      supr data  rd                     4.92   uS
+005    _systemio+000008      $7200      supr data  rd                     4.88   uS
+006    _systemio+00000A      $04A2      supr data  rd                     4.92   uS
+007    _systemio+00000C      $ED5F      supr data  rd                     4.88   uS
+008    _systemio+00000E      $25AA      supr data  rd                     4.92   uS
+009    _systemio+000010      $208A      supr data  rd                     4.88   uS
+010    _systemio+000012      $EF1F      supr data  rd                     4.92   uS
+011    _systemio+000014      $B7D7      supr data  rd                     4.88   uS
+012    _systemio+000016      $00AA      supr data  rd                     4.92   uS
+013    _systemio+000018      $008E      supr data  rd                     4.88   uS
+014    _systemio+00001A      $F7F5      supr data  rd                     4.92   uS
+015    _systemio+00001C      $F655      supr data  rd                     4.88   uS

STATUS:   M68302--Running in monitor      Software break: 0002c060sp

display trace
```

The default display mode settings cause source lines and symbols to appear in the trace list.

Captured states are numbered in the left-hand column of the trace list. Line 0 always contains the state that caused the analyzer to trigger.

Other columns contain address information, data values, opcode or status information, and time count information.

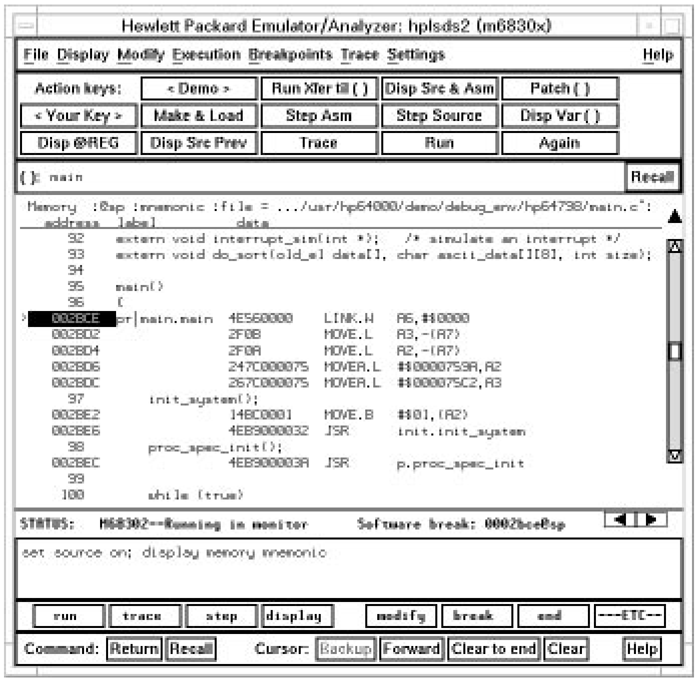
Step 11: Patch assembly language code

The Patch () action key lets you patch code in your program.

1 With "main" still in the entry buffer, click on the **Run Xfer til ()** action key.

2 To display memory with assembly-level instructions intermixed with the high-level source lines, click on the **Disp Src & Asm** action key.

Figure 11


Hewlett Packard Emulator/Analyzer: hplsds2 (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: < Demo > Run Xfer til ( ) Disp Src & Asm Patch ( )
< Your Key > Make & Load Step Asm Step Source Disp Var ( )
Disp @REG Disp Src Prev Trace Run Again
Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
( ): main
Memory  :@sp :mnemonic :file = .../usr/hp64000/demo/debug_env/hp64738/main.c":
   address  label          data
       92    extern void interrupt_sim(int *);   /* simulate an interrupt */
       93    extern void do_sort(old_el data[], char ascii_data[][8], int size);
       94
       95    main()
       96    {
>   002BCE  pr|main.main   4E560000    LINK.W   A6,#0000
    002BD2                 2F0B        MOVE.L   A3,-(A7)
    002BD4                 2F0A        MOVE.L   A2,-(A7)
    002BD6                 247C000075  MOVEA.L  #0000759A,A2
    002BDC                 267C000075  MOVEA.L  #000075C2,A3
       97        init_system();
    002BE2                 14BC0001    MOVE.B   #01,(A2)
    002BE6                 4EB9000032  JSR      init.init_system
       98        proc_spec_init();
    002BEC                 4EB900003A  JSR      p.proc_spec_init
       99
      100        while (true)

STATUS:   M68302--Running in monitor      Software break: 0002bce@sp

set source on; display memory mnemonic
```

3 Click on the address for main in the source display to enter the address into the entry buffer.

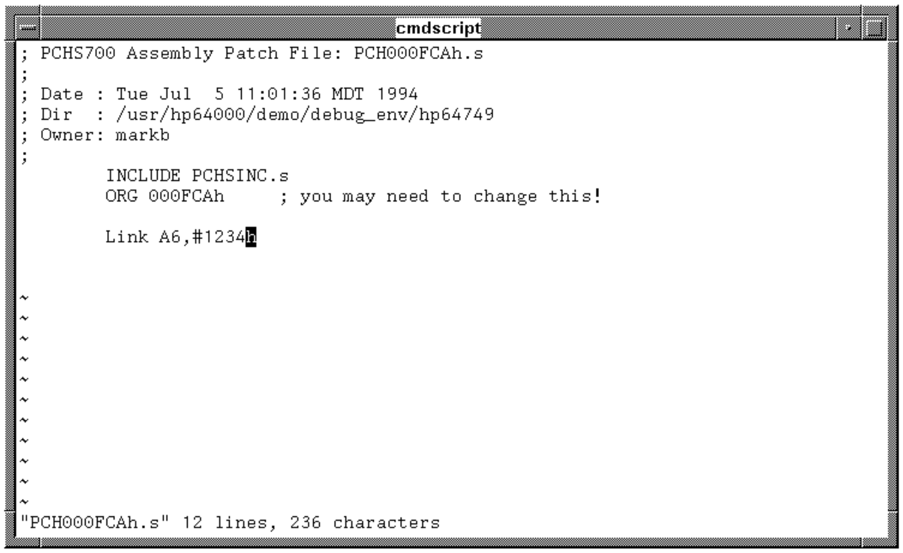
4 Click on the **Patch ()** action key.

A window appears and the vi editor is started. Notice that the address for "main" appears in the ORG statement.

5 Add the line:

```
LINK A6,#1234h
```

Figure 12



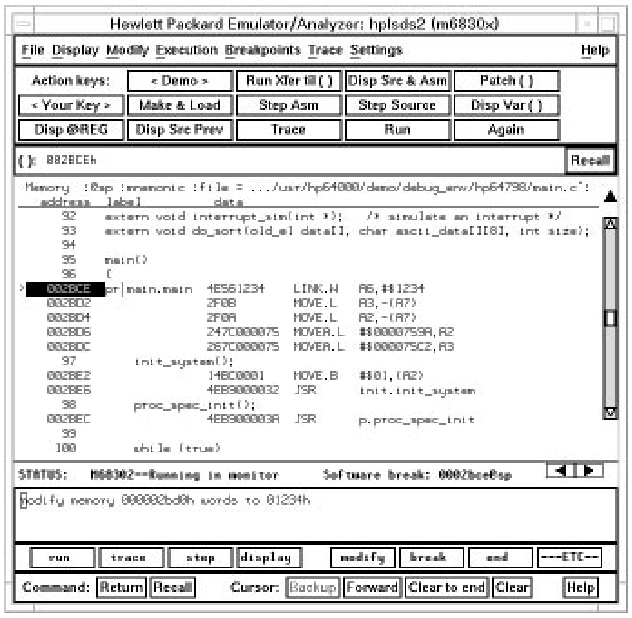
```
; PCHS700 Assembly Patch File: PCH000FCAh.s
;
; Date : Tue Jul  5 11:01:36 MDT 1994
; Dir  : /usr/hp64000/demo/debug_env/hp64749
; Owner: markb
;
        INCLUDE PCHSINC.s
        ORG 000FCAh      ; you may need to change this!

        Link A6,#1234h


~
~
~
~
~
~
~
~
~
~
~
"PCH000FCAh.s" 12 lines, 236 characters
```

6 Exit out of the editor, saving your changes.

The file you just edited is assembled, and the patch main menu appears.

7 Type "a" and press <RETURN> to apply the patch.

Figure 13



```
( ): 002BCEh

Memory  :@sp :mnemonic :file = .../usr/hp64000/demo/debug_env/hp64738/main.c":
   address  label          data
      92    extern void interrupt_sim(int *);   /* simulate an interrupt */
      93    extern void do_sort(old_el data[], char ascii_data[][8], int size);
      94
      95    main()
      96    {
> 002BCE  pr|main.main  4E561234    LINK.W    A6,#$1234
  002BD2                2F0B        MOVE.L    A3,-(A7)
  002BD4                2F0A        MOVE.L    A2,-(A7)
  002BD6                247C000075  MOVEA.L   #$0000759A,A2
  002BDC                267C000075  MOVEA.L   #$000075C2,A3
      97        init_system();
  002BE2                14BC0001    MOVE.B    #$01,(A2)
  002BE6                4EB9000032  JSR       init.init_system
      98        proc_spec_init();
  002BEC                4EB900003A  JSR       p.proc_spec_init
      99
     100        while (true)

STATUS:   M68302--Running in monitor       Software break: 0002bce@sp

modify memory 000002bd0h words to 01234h
```

Notice in the emulator/analyzer interface that the instruction at address "main" has changed.

8 Click on the **Patch ()** action key again.

A window running the vi editor again appears, allowing you to modify the patch code that was just created.

9 Modify the line you added previously to:

```
LINK A6,#0
```

10 Exit out of the editor, saving your changes.

The file you just edited is assembled, and the patch main menu appears.

11 Type "a" and press <RETURN> to apply the patch.

Notice in the emulator/analyzer interface that the instruction at address "main" has been changed back to what it was originally.

When patching a single address, make sure the new instruction takes up the same number of bytes as the old instruction; otherwise, you may inadvertently modify code that follows.

12 Type "main+4 thru main+15" in the entry buffer.

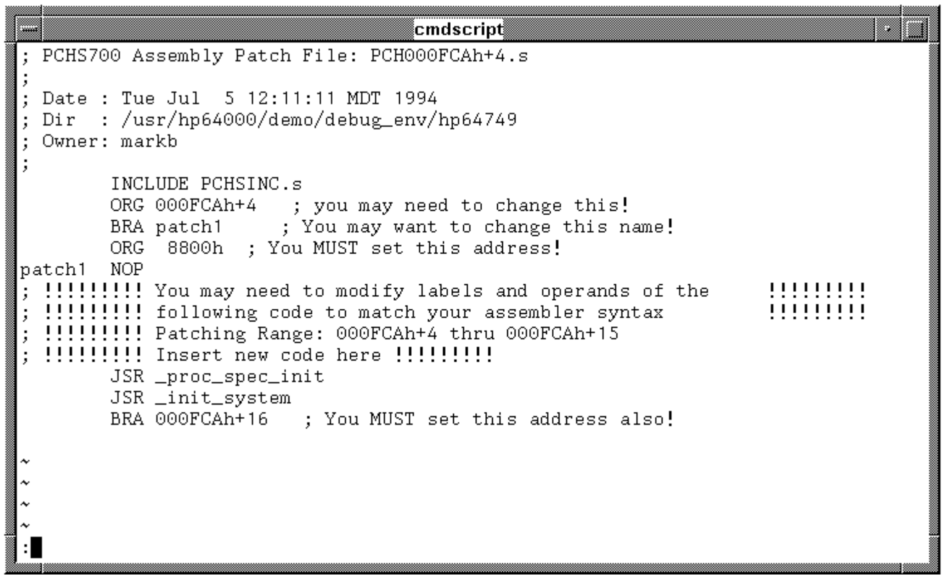
By entering an address range in the entry buffer (that is, <address> thru <address>) before clicking on the Patch () action key, you can modify a patch template file which allows you to insert as much or as little code as you wish.

If you make a mistake while editing the patch, then save the changes, when you enter the file again to fix the mistake, a "duplicate symbol" error will appear on the INCLUDE PCHSINC.s" line when the patch is reassembled. Although this message is displayed, it will not affect the new patch assembly.

13 Click on the **Patch ()** action key again.

A window running the vi editor again appears. Suppose you want to patch the demo program so that the proc_spec_init() function is called before the init_system() function. Suppose also that there is memory available at address 8800H. Edit the patch template file as shown below.

Figure 14



```
; PCHS700 Assembly Patch File: PCH000FCAh+4.s
;
; Date : Tue Jul  5 12:11:11 MDT 1994
; Dir  : /usr/hp64000/demo/debug_env/hp64749
; Owner: markb
;
        INCLUDE PCHSINC.s
        ORG 000FCAh+4   ; you may need to change this!
        BRA patch1     ; You may want to change this name!
        ORG  8800h  ; You MUST set this address!
patch1  NOP
; !!!!!!!!! You may need to modify labels and operands of the     !!!!!!!!!
; !!!!!!!!! following code to match your assembler syntax        !!!!!!!!!
; !!!!!!!!! Patching Range: 000FCAh+4 thru 000FCAh+15
; !!!!!!!!! Insert new code here !!!!!!!!!
        JSR _proc_spec_init
        JSR _init_system
        BRA 000FCAh+16   ; You MUST set this address also!

~
~
~
~
:
```

Notice that symbols can be used in the patch file.

14 Exit out of the editor, saving your changes.

The file you just edited is assembled, and the patch main menu appears.

15 Type "a" and press <RETURN> to apply the patch.

You can step through the program to view execution of the patch.

Step 12: Exit the emulator/analyzer interface

- To exit the emulator/analyzer interface and release the emulator, choose **File→Exit→Released**.

  Or, using the command line, enter:

```
end release_system
```

Part 2

User’s Guide

Part 2

A complete set of task instructions and problem-solving guidelines, with a few basic concepts.

2

Plugging into a Target System

Plugging the Emulator into a Target System

For information about plugging the emulator into a target system, refer to the *6830x Installation/Service/Terminal Interface User's Guide*. That manual describes connecting these emulators to a target system using a 144-pin TQFP, a PQFP (plastic quad flat pack) cable.

- M68302—Model HP 64798C
- M68LC302—Model HP 64798F
- M68306—Model HP 64798H

CAUTION

Possible Damage to the Emulator Probe. The emulator contains devices that are susceptible to damage by static discharge. Therefore, precautionary measures should be taken before handling the emulator probe to avoid damaging the internal components of the emulator by static electricity.

3

Starting and Exiting HP 64700 Interfaces

Starting and Exiting HP 64700 Interfaces

You can use several types of interfaces to the same emulator at the same time to give yourself different views into the target system.

The strength of the emulator/analyzer interface is that it lets you perform the real-time analysis measurements that are helpful when integrating hardware and software.

The Software Performance Analyzer interface (which is also a separate product) lets you make measurements that can help you improve the performance of your software.

These interfaces can operate at the same time with the same emulator. When you perform an action in one of the interfaces, it is reflected in the other interfaces.

Up to 10 interface windows may be started for the same emulator. Only one SPA window is allowed, but you can start multiple emulator/analyzer interface windows.

The tasks associated with starting and exiting HP 64700 interfaces are grouped into the following sections:

- Starting the emulator/analyzer interface.
- Opening other HP 64700 interface windows.
- Exiting HP 64700 interfaces.

Starting the Emulator/Analyzer Interface

Before starting the emulator/analyzer interface, the emulator and interface software must have already been installed as described in the "Installation" chapter.

This section describes how to:

- Start the interface.
- Start the interface using the default configuration.
- Run a command file on interface startup.
- Display the status of emulators defined in the 64700tab.net file.
- Unlock an interface that was left locked by another user.

To start the emulator/analyzer interface

- Use the **emul700 <emul_name>** command.

If /usr/hp64000/bin is specified in your PATH environment variable (as shown in the "Installation" chapter), you can start the interface with the **emul700 <emul_name>** command. The "emul_name" is the logical emulator name given in the HP 64700 emulator device table (/usr/hp64000/etc/64700tab.net). It may also be the LAN address.

If you are running a window system on your host computer (for example, the X Window System), you can run the interface in a maximum of 10 windows. This capability provides you with several views into the emulation system. For example, you can display memory in one window, registers in another, an analyzer trace in a third, and data in the fourth.

Examples

To start the emulator/analyzer interface for the 6830x emulator:

```
$ emul700 em6830x
```

The "em6830x" in the command above is the logical emulator name given in the HP 64700 emulator device table file (/usr/hp64000/etc/64700tab.net).

```
# Blank  lines  and  the  rest of each line after a '#' character are ignored.
# The information in each line must be in the specified order, with  one  line
# for  each  HP series 64700 emulator.  Use blanks or tabs to separate fields.
#
#--------+------------+-----------+-------------------------------------------
# Channel|  Logical   | Processor | Remainder of Information for the Channel
#  Type  |   Name     |   Type    | (IP address for LAN connections)
#--------+------------+-----------+-------------------------------------------
#  lan:      em6830x    m6830x     21.17.9.143
serial:      em6830x    m6830x     myhost /dev/emcom23 OFF 9600 NONE XON 2 8
```

If you're currently running the X Window System, the Graphical User Interface starts; otherwise, the Softkey Interface starts.

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 UNIX prompt. Error messages are described in the "Error Messages" chapter.

To start the interface using the default configuration

- Use the **emul700 -d <emul_name>** command.

In the **emul700 -d <emul_name>** command, the -d option starts the default configuration. The -d option is ignored if the interface is already running in another window or on another terminal.

To run a command file on interface startup

- Use the **emul700 -c <cmd_file> <emul_name>** command.

You can cause command files to be run upon starting the interface by using the -c <cmd_file> option of the **emul700** command.

Refer to the "Using Command Files" section in the "Entering Commands" chapter for information on creating command files.

Examples

To start the emulator/analyzer interface and run the "startup" command file:

```
$ emul700 -c startup em6830x
```

To display the status of emulators

- Use the **emul700 -l** or **emul700 -lv** command.

The -l option of the **emul700** command lists the status of all emulators defined in the 64700tab and 64700tab.net files. If a logical emulator name is included in the command, just the status of that emulator is listed.

You can also use the -v option with the -l option for a verbose listing of the status information.

Examples

To list, verbosely, the status of the emulator whose logical name is "em6830x":

```
$ emul700 -lv em6830x
```

The information may be similar to:

```
em6830x - m6830x running; user = guest
    description:      M6830x emulation, 512K bytes emul mem
    user interfaces:  xdebug, xemul, xperf, skemul, sktiming
    device channel:   /dev/emcom23
```

Or, the information may be similar to:

```
em6830x - m6830x running; user = guest@myhost
    description:      M6830x emulation, 512K bytes emul mem
    user interfaces:  xdebug, xemul, xperf, skemul, sktiming
    internet address: 21.17.9.143
```

To unlock an interface that was left locked by another user

- Use the **emul700 -U <emul_name>** command.

The -U option of the **emul700** command may be used to unlock the emulators whose logical names are specified. This command will fail if there currently is a session in progress.

Examples

To unlock the emulator whose logical name is "em6830x":

```
$ emul700 -U em6830x
```

Opening Other HP 64700 Interface Windows

The **File→Emul700** menu lets you open additional emulator/analyzer interface windows or other HP 64700 interface windows if those products have been installed (for example, the software performance analyzer (SPA) interface).

This section shows you how to:

- Open additional emulator/analyzer interface windows.
- Open the software performance analyzer (SPA) interface window.

To open additional emulator/analyzer windows

- To open additional Graphical User Interface windows, choose **File→Emul700→Graphic Windows→ Emulator/Analyzer**.

Or, using the command line, enter:

```
emul700 <emul_name>
```

- To open additional conventional Softkey Interface windows, choose **File→Emul700→ Terminal Windows→Emulator/Analyzer** .

Or, using the command line, enter:

```
emul700 <emul_name>
```

You can open additional Graphical User Interface windows, or terminal emulation windows containing the Softkey Interface. When you open an additional window, the status line will show that this session is joining a session already in progress, and the event log is displayed.

You can enter commands in any window. When you enter commands in different windows, the command entered in the first window must complete before the command entered in the second window can start. The status lines and the event log displays are updated in all windows.

To open the software performance analyzer (SPA) interface window

- Choose **File→Emul700→Graphic Windows→Performance Analyzer.**

Or, using the command line, enter:

```
emul700 -u xperf <emul_name>
```

For information on how to use the software performance analyzer, refer to the *Software Performance Analyzer User's Guide*.

Exiting HP 64700 Interfaces

There are several options available when exiting the HP 64700 interfaces. You can simply close one of the open interface windows, or you can exit the debug session by closing all the open windows. When exiting the debug session, you can lock the emulator so that you can continue later, or you can release the emulation system so that others may use it. This section describes how to:

- Close an interface window.
- Exit a debug/emulation session.

To close an interface window

- **In the interface window you wish to close, choose File→Exit→Window.**

Or, using the command line, enter:

```
end
```

All other interface windows remain open, and the emulation session continues, unless the window closed is the only one open for the emulation session. In that case, closing the window ends the emulation session, but locks the emulator so that other users cannot access it.

To exit a debug/emulation session

- To exit the interface, save your configuration to a temporary file, and lock the emulator so that it cannot be accessed by other users, choose **File→Exit→Locked**.

  Or, using the command line, enter:

```
end locked
```

- To exit the interface and release the emulator for access by other users, choose **File→Exit→Released**.

  Or, using the command line, enter:

```
end release_system
```

If you exit the interface locked, the interface saves the current configuration to a temporary file and locks the emulator to prevent other users from accessing it. When you again start the interface with the **emul700** command, the temporary file is reloaded, and therefore, you return to the configuration you were using when you quit the interface locked.

Also saved when you exit the interface locked are the contents of the entry buffer and command recall buffer. These recall buffer values will be present when you restart the interface.

In contrast, if you end released, you must have saved the current configuration to a configuration file (if the configuration has changed), or the changes will be lost.

4

Entering Commands

Entering Commands

When an X Window System that supports OSF/Motif interfaces is running on the host computer, the emulator/analyzer interface is the Graphical User Interface which provides pull-down and pop-up menus, point and click setting of breakpoints, cut and paste, online help, customizable action keys and pop-up recall buffers, etc.

The emulator/analyzer interface also provides the Softkey Interface for several types of terminals, terminal emulators, and bitmapped displays. When using the Softkey Interface, commands are entered from the keyboard.

When using the Graphical User Interface, the *command line* portion of the interface gives you the option of entering commands in the same manner as they are entered in the Softkey Interface. If you are using the Softkey Interface, you can only enter commands from the keyboard using the command line.

The menu commands in the Graphical User Interface are a subset of the commands available when using the command line. While you have a great deal of capability in the menu commands, you have even more in the command line.

This chapter shows you how to enter commands in each type of emulator/analyzer interface. The tasks associated with entering commands are grouped into the following sections:

- Using menus, the entry buffer, and action keys.
- Using the command line with the mouse.
- Using the command line with the keyboard.
- Using command files.
- Using pod commands.
- Forwarding commands to other HP 64700 interfaces.

Using Menus, the Entry Buffer, and Action Keys

This section describes the tasks you perform when using the Graphical User Interface to enter commands. This section describes how to:

- Choose a pull-down menu item using the mouse.
- Choose a pull-down menu item using the keyboard.
- Use the pop-up menus.
- Use the entry buffer.
- Copy and paste to the entry buffer.
- Use action keys.
- Use dialog boxes.
- Access help information.

To choose a pull-down menu item using the mouse (method 1)

1 **Position the mouse pointer over the name of the menu on the menu bar.**

2 **Press and hold the *command select* mouse button to display the menu.**

3 **While continuing to hold down the mouse button, move the mouse pointer to the desired menu item. If the menu item has a cascade menu (identified by an arrow on the right edge of the menu button), then continue to hold the mouse button down and move the mouse pointer toward the arrow on the right edge of the menu. The cascade menu will display. Repeat this step for the cascade menu until you find the desired menu item.**

4 Release the mouse button to select the menu choice.

If you decide not to select a menu item, simply continue to hold the mouse button down, move the mouse pointer off of the menu, and release the mouse button.

Some menu items have an ellipsis ("...") as part of the menu label. An ellipsis indicates that the menu item will display a dialog or message box when the menu item is chosen.

To choose a pull-down menu item using the mouse (method 2)

1 Position the mouse pointer over the menu name on the menu bar.

2 Click the *command select* mouse button to display the menu.

3 Move the mouse pointer to the desired menu item. If the menu item has a cascade menu (identified by an arrow on the right edge of the menu button), then repeat the previous step and then this step until you find the desired item.

4 Click the mouse button to select the item.

If you decide not to select a menu item, simply move the mouse pointer off of the menu and click the mouse button.

Some menu items have an ellipsis ("...") as part of the menu label. An ellipsis indicates that the menu item will display a dialog or other box when the menu item is chosen.

To choose a pull-down menu item using the keyboard

- To initially display a pull-down menu, press and hold the *menu select* key (for example, the "Extend char" key on an HP 9000 keyboard) and then type the underlined character in the menu label on the menu bar. (For example, "f" for "File". Type the character in lowercase only.)
- To move right to another pull-down menu after having initially displayed a menu, press the *right-arrow* key.
- To move left to another pull-down menu after having initially displayed a menu, press the *left-arrow* key.

- To move down one menu item within a menu, press the *down-arrow* key.
- To move up one menu item within a menu, press the *up-arrow* key.
- To choose a menu item, type the character in the menu item label that is underlined. Or, move to the menu item using the arrow keys and then press the <*RETURN*> key on the keyboard.
- To cancel a displayed menu, press the *Escape* key.

The interface supports keyboard mnemonics and the use of the arrow keys to move within or between menus. For each menu or menu item, the underlined character in the menu or menu item label is the keyboard mnemonic character. Notice the keyboard mnemonic is not always the first character of the label. If a menu item has a cascade menu attached to it, then typing the keyboard mnemonic displays the cascade menu.

Some menu items have an ellipsis ("...") as part of the menu label. An ellipsis indicates that the menu item will display a dialog or other box when the menu item is chosen.

Dialog boxes support the use of the keyboard as well. To direct keyboard input to a dialog box, you must position the mouse pointer somewhere inside the boundaries of the dialog box. That is because the interface *keyboard focus policy* is set to *pointer*. That just means that the window containing the mouse pointer receives the keyboard input.

In addition to keyboard mnemonics, you can also specify keyboard accelerators which are keyboard shortcuts for selected menu items. Refer to the "Setting X Resources" chapter and the "Softkey.Input" scheme file for more information about setting the X resources that control defining keyboard accelerators.

To choose pop-up menu items

1 Move the mouse pointer to the area whose pop-up menu you wish to access. (If a pop-up menu is available, the mouse pointer changes from an arrow to a hand.)

2 Press and hold the *select* mouse button.

3 After the pop-up menu appears (while continuing to hold down the mouse button), move the mouse pointer to the desired menu item.

4 Release the mouse button to select the menu choice.

If you decide not to select a menu item, simply continue to hold the mouse button down, move the mouse pointer off of the menu, and release the mouse button.

To place values into the entry buffer using the keyboard

1 Position the mouse pointer within the text entry area. (An "I-beam" cursor will appear.)

2 Enter the text using the keyboard.

To clear the entry buffer text area from beginning until end, press the <Ctrl>u key combination.

To copy and paste to the entry buffer

- To copy and paste a discrete text string as determined by the interface, position the mouse pointer over the text to copy and click the *paste* mouse button.
- To specify the exact text to copy to the entry buffer: press and hold the *paste* mouse button; drag the mouse pointer to highlight the text to copy-and-paste; release the *paste* mouse button.

You can copy-and-paste from the display area, the status line, and from the command line entry area.

When you position the pointer and click the mouse button, the interface expands the highlight to include the most complete text string it considers to be discrete. Discrete here means that the interface will stop expanding the highlight in a given direction when it discovers a delimiting character not determined to be part of the string. A common delimiter would, of course, be a space.

When you press and hold the mouse button and drag the pointer to highlight text, the interface copies all highlighted text to the entry buffer when you release the mouse button.

Because the interface displays absolute addresses as hex values, any copied and pasted string that can be interpreted as a hexadecimal value (that is, the string contains only numbers 0 through 9 and characters "a" through "f") automatically has an "h" appended.

Copy and paste and the Entry Buffer

If you have multiple Graphical User Interface windows open, a copy-and-paste action in any window causes the text to appear in all entry buffers in all windows. That is because although there are a number of entry buffers being displayed, there is actually only one entry buffer and it is common to all windows. That means you can copy a symbol or an address from one window and then use it in another window.

On a memory display or trace display, a symbol may not be completely displayed because there are too many characters to fit into the width limit for a particular column of the display. To make a symbol usable for copy-and-paste, you can scroll the screen left or right to display all, or at least more, of the characters from the symbol. The interface displays absolute addresses as hex values.

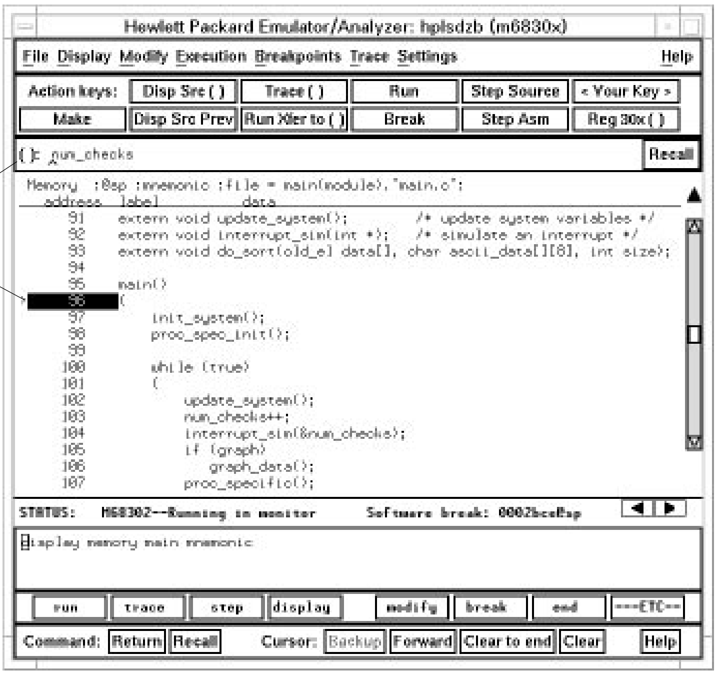
Text pasted into the entry buffer replaces that which is currently there. You cannot use paste to append text to existing text already in the entry buffer.

See "To copy-and-paste from the entry buffer to the command line entry area" for information about pasting the contents of the entry buffer into the command line entry area.

Example

To paste the symbol "num_checks" into the entry buffer from the interface display area, position the mouse pointer over the symbol and then click the paste mouse button.

A mouse click causes the interface to expand the highlight to include the symbol "num_checks" and paste the symbol into the entry buffer.


Hewlett Packard Emulator/Analyzer: hplsdzb (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x ( )
Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
( ): num_checks

Memory  :@sp :mnemonic :file = main(module)."main.c":
  address  label          data
      91     extern void update_system();        /* update system variables */
      92     extern void interrupt_sim(int *);   /* simulate an interrupt */
      93     extern void do_sort(old_el data[], char ascii_data[][8], int size);
      94
      95     main()
      96     {
      97         init_system();
      98         proc_spec_init();
      99
     100         while (true)
     101         {
     102             update_system();
     103             num_checks++;
     104             interrupt_sim(&num_checks);
     105             if (graph)
     106                graph_data();
     107             proc_specific();

STATUS:   M68302--Running in monitor      Software break: 0002bce@sp

display memory main mnemonic
```

To recall entry buffer values

- Position the mouse pointer over the **Recall** button just to the right of the entry buffer text area, click the mouse button to bring up the Entry Buffer Recall dialog box, and then choose a string from that dialog box.

The Entry Buffer Recall dialog box contains a list of entries gained during the emulation session as well as any predefined entries present at interface startup.

If you exit the emulation/analysis session with the interface "locked", recall buffer values are saved and will be present when you restart the interface.

You can predefine entries for the Entry Buffer Recall dialog box and define the maximum number of entries by setting X resources (refer to the "Setting X Resources" chapter).

See the following "To use dialog boxes" section for information about using dialog boxes.

To use the entry buffer

1. Place information into the entry buffer (see the previous "To place values into the entry buffer using the keyboard", "To copy-and-paste to the entry buffer", or "To recall entry buffer values" task descriptions).
2. Choose the menu item, or click the action key, that uses the contents of the entry buffer (that is, the menu item or action key that contains "()").

To copy and paste from the entry buffer to the command line entry area

1 Place text to be pasted into the command line in the entry buffer text area.

You may do this:

- Copying the text from the display area using the copy and paste feature.
- Entering the text directly by typing it into the entry buffer text area.
- Choosing the text from the entry buffer recall dialog box.

2 Position the mouse pointer within the command line text entry area.

3 If necessary, reposition the cursor to the location where you want to paste the text.

4 If necessary, choose the insert or replace mode for the command entry area.

5 Click the *command paste* mouse button to paste the text in the command line entry area at the current cursor position.

The entire contents of the entry buffer are pasted into the command line at the current cursor position.

Although a paste from the display area to the entry buffer affects all displayed entry buffers in all open windows, a paste from the entry buffer to the command line only affects the command line of the window in which you are currently working.

See "To copy and paste to the entry buffer" for information about pasting information from the display into the entry buffer.

To use the action keys

1 If the action key uses the contents of the entry buffer, place the desired information in the entry buffer.

2 Position the mouse pointer over the action key and click the action key.

Action keys are user-definable pushbuttons that perform interface or system functions. Action keys can use information from the entry buffer — this makes it possible to create action keys that are more general and flexible.

Several action keys are predefined when you first start the Graphical User Interface. You can use the predefined action keys, but you'll really appreciate action keys when you define and use your own.

Action keys are defined by setting an X resource. Refer to the chapter "Setting X Resources" for more information about creating action keys.

To use dialog boxes

1 **Click on an item in the dialog box list to copy the item to the text entry area.**

2 **Edit the item in the text entry area (if desired).**

3 **Click on the "OK" pushbutton to make the selection and close the dialog box, click on the "Apply" pushbutton to make the selection and leave the dialog box open, or click on the "Cancel" pushbutton to cancel the selection and close the dialog box.**

The graphical interface uses a number of dialog boxes for selection and recall as described in the following:

Directory Selection	Selects the working directory. You can change to a previously accessed directory, a predefined directory, or specify a new directory.
File Selection	From the working directory, lets you select an existing file name or specify a new file name.
Entry Buffer Recall	Lets you recall a previously used entry buffer text string, a predefined entry buffer text string, or a newly entered entry buffer string, to the entry buffer text area.
Command Recall	Lets you recall a previously executed command, a predefined command, or a newly entered command, to the command line.

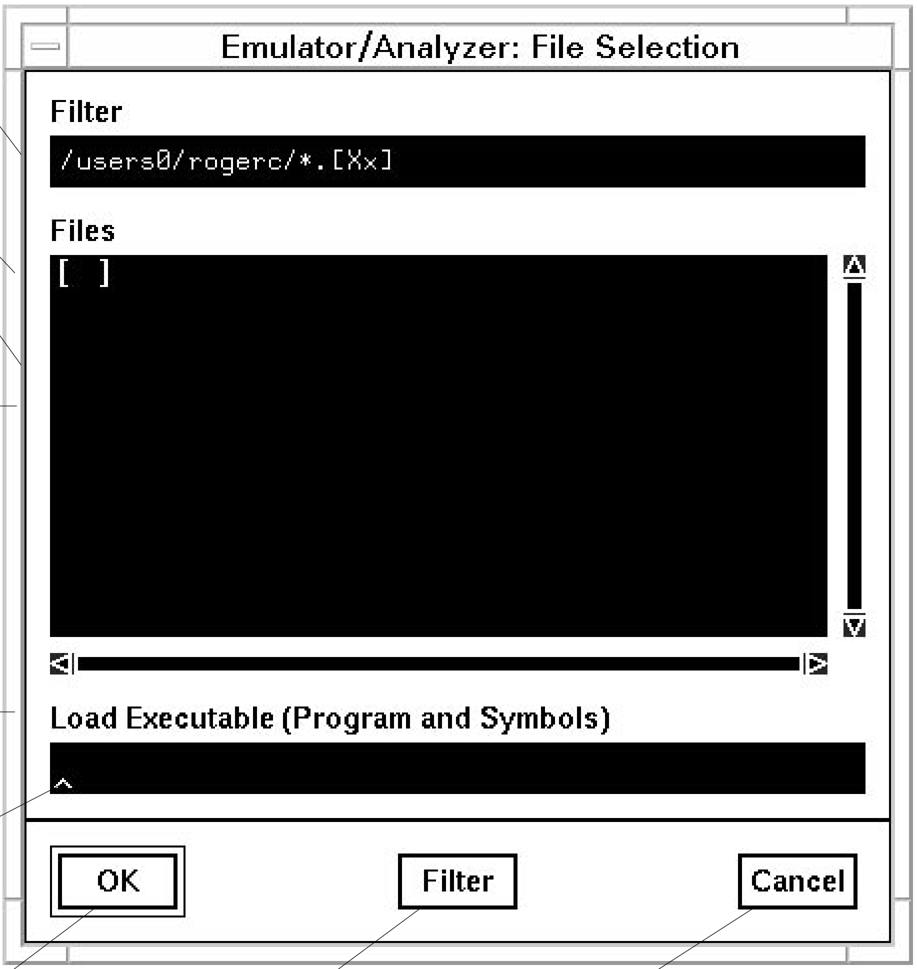
The dialog boxes share some common properties:

- Most dialog boxes can be left on the screen between uses.
- Dialog boxes can be moved around the screen and do not have to be positioned over the graphical interface window.
- If you iconify the interface window, all dialog boxes are iconified along with the main window.

Except for the File Selection dialog box, predefined entries for each dialog box (and the maximum number of entries) are set via X resources (refer to the "Setting X Resources" chapter).

Examples

To use the File Selection dialog box:


Emulator/Analyzer: File Selection
Filter
/users0/rogerc/*.[Xx]
Files
[ ]
Load Executable (Program and Symbols)
OK
Filter
Cancel

The file filter selects specific files.

A list of filter-matching files from the current directory.

A list of files previously accessed during the emulation session.

A single click on a file name from either list highlights the file name and copies it to the text area. A double click chooses the file and closes the dialog box.

Label informs you what kind of file selection you are performing.

Text entry area. Text is either copied here from the recall list, or entered directly.

Clicking this button chooses the file name displayed in the text entry area and closes the dialog box.

Entering a new file filter and clicking this button causes a list of files matching the new filter to be read from the directory.

Clicking this button cancels the file selection operation and closes the dialog box.

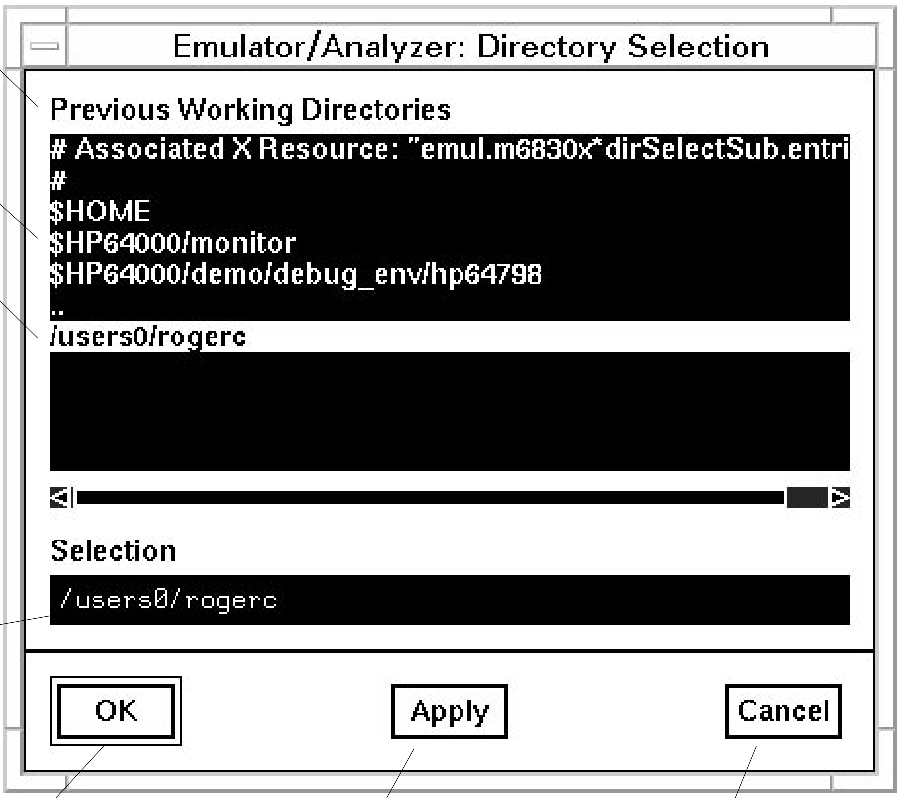
To use the Directory Selection dialog box:

Label informs you of the type of list displayed.

A list of predefined or previously accessed directories.

A single click on a directory name from the list highlights the name and copies it to the text area. A double click chooses the directory and closes the dialog box.

Text entry area. Directory name is either copied here from the recall list, or entered directly.



Emulator/Analyzer: Directory Selection
Previous Working Directories
Selection
OK
Apply
Cancel

```
# Associated X Resource: "emul.m6830x*dirSelectSub.entri
#
$HOME
$HP64000/monitor
$HP64000/demo/debug_env/hp64798
..
/users0/rogerc
```

```
/users0/rogerc
```

Clicking this button chooses the directory displayed in the text entry area and closes the dialog box.

Clicking this button chooses the directory displayed in the text entry area, but keeps the dialog box on the screen instead of closing it.

Clicking this button cancels the directory selection operation and closes the dialog box.

To access help information

1 Display the Help Index by choosing **Help→General Topic...** or **Help→Command Line...**.

2 Choose a topic of interest from the Help Index.

The Help Index lists topics covering operation of the interface as well other information about the interface. When you choose a topic from the Help Index, the interface displays a window containing the help information. You may leave the window on the screen while you continue using the interface.

Using the Command Line with the Mouse

When using the Graphical User Interface, the *command line* portion of the interface gives you the option of entering commands in the same manner as they are entered in the Softkey Interface. Additionally, the graphical interface makes the softkey labels pushbuttons so commands may be entered using the mouse.

If you are using the Softkey Interface, using the command line with the keyboard is the only way to enter commands.

This section describes how to:

- Turn the command line off/on.
- Enter commands.
- Edit commands.
- Recall commands.
- Display the help window.

To turn the command line on or off

- To turn the command line on or off using the pull-down menu, choose **Settings→Command Line**.
- To turn the command line on or off using the status line pop-up menu: position the mouse pointer within the status line area, press and hold the *select* mouse button, and choose **Command Line Off** from the menu.
- To turn the command line off using the command line entry area pop-up menu: position the mouse pointer within the entry area, press and hold the **select** mouse button, and choose **Command Line Off** from the menu.

Turns display of the command line area "on" or "off." On means that the command line is displayed and you can use the softkey label pushbuttons, the command return and recall pushbuttons, and the cursor pushbuttons for command line editing. Off means the command line is not displayed and you use only the pull-down menus and the action keys to control the interface.

The command line area begins just below the status line and continues to the bottom of the emulator/analyzer window. The status line is not part of the command line and continues to be displayed whether the command line is on or off.

Choosing certain pull-down menu items while the command line is off causes the command line to be turned on. That is because the menu item chosen requires some input at the command line that cannot be supplied another way.

To enter a command

1 Build a command using the softkey label pushbuttons by successively positioning the mouse pointer on a pushbutton and clicking the *pushbutton select* mouse button until a complete command is formed.

2 Execute the completed command by clicking the *Return* pushbutton (found near the bottom of the command line in the "Command" group).

Or:

Execute the completed command using the Command Line entry area pop-up menu: Position the mouse pointer in the command line entry area; press and hold the *select* mouse button until the Command Line pop-up menu appears; then, choose the *Execute Command* menu item.

You may need to combine pushbutton and keyboard entry to form a complete command.

A complete command is a string of softkey labels and text entered with the keyboard. You know a command is complete when Return pushbutton is not halfbright. The interface does not check or act on a command, however, until the command is executed. (In contrast, commands resulting from pull-down menu choices and action keys are supplied with the needed carriage return as part of the command.)

To edit the command line using the command line pushbuttons

- To clear the command line, click the *Clear* pushbutton.
- To clear the command line from the cursor position to the end of the line, click the *Clear to end* pushbutton.
- To move to the right one command word or token, click the *Forward* pushbutton.
- To move to the left one command word or token, click the *Backup* pushbutton.
- To insert characters at the cursor position, press the **insert key** to change to insertion mode, and then type the characters to be inserted.
- To delete characters to the left of the cursor position, press the <*BACKSPACE*> key.

When the cursor arrives at the beginning of a command word or token, the softkey labels change to display the possible choices at that level of the command.

When moving by words left or right, the Forward pushbutton becomes half-bright and unresponsive when the cursor reaches the end of the command string. Similarly, the Backup pushbutton becomes half-bright and unresponsive when the cursor reaches the beginning of the command.

See "To edit the command line using the mouse and the command line pop-up menu" and "To edit the command line using the keyboard" for information about additional editing operations you can perform.

To edit the command line using the command line pop-up menu

- To clear the command line: position the mouse pointer within the Command Line entry area; press and hold the *select* mouse button until the Command Line pop-up menu appears; choose **Clear Entire Line** from the menu.
- To clear the command line from the cursor position to the end of the line: position the mouse pointer at the place where you want the clear-to-end to start; press and hold the *select* mouse button until the

Command Line pop-up menu appears; choose **Clear to End of Line** from the menu.

- To position the cursor and insert characters at the cursor location: position the mouse pointer in a non-text area of the command line entry area; press and hold the *select* mouse button to display the Command Line pop-up menu; choose **Position Cursor, Insert Mode** from the menu; type the characters to be inserted.
- To replace characters at the current cursor location: position the mouse pointer in a non-text area of the command line entry area; press and hold the *select* mouse button to display the Command Line pop-up menu; choose **Position Cursor, Replace Mode** from the menu; type the characters to be inserted.
- To position the cursor and replace characters at the cursor location: position the mouse pointer in a non-text area of the command line entry area; press and hold the *select* mouse button to display the Command Line pop-up menu; choose **Position Cursor, Replace Mode** from the menu; type the characters to be inserted.

When the cursor arrives at the beginning of a command word or token, the softkey labels change to display the possible choices at that level of the command.

See "To edit the command line using the mouse and the command line pushbuttons" and "To edit the command line using the keyboard" for information about additional editing operations you can perform.

To recall commands

1. Click the pushbutton labeled *Recall* in the Command Line to display the dialog box.
2. Choose a command from the buffer list. (You can also enter a command directly into the text entry area of the dialog box.)

Because all command entry methods in the interface — pull-down menus, action keys, and command line entries — are echoed to the command line entry area, the contents of the Command Recall dialog box is not restricted to just commands entered directly into the command line entry area.

The Command Recall dialog box contains a list of interface commands executed during the session as well as any predefined commands present at interface startup.

If you exit the emulation/analysis session with the interface "locked", commands in the recall buffer are saved and will be present when you restart the interface.

You can predefine entries for the Command Recall dialog box and define the maximum number of entries by setting X resources (refer to the "Setting X Resources" chapter).

See "To use dialog boxes" for information about using dialog boxes.

To get help about the command line

- **To display the help topic explaining the operation of the command line, press the *Help* pushbutton located near the bottom-right corner of the Command Line area.**

Using the Command Line with the Keyboard

When using the command line with the keyboard, you enter commands by pressing softkeys whose labels appear at the bottom of the screen. Softkeys provide for quick command entry, and minimize the possibility of errors.

The command line also provides command completion. You can type the first few characters of a command (enough to uniquely identify the command) and then press <Tab>. The interface completes the command word for you.

Entering commands with the keyboard is easy. However, the interface provides other features that make entering commands even easier. For example, you can:

- Enter multiple commands on one line.
- Recall commands.
- Edit commands.
- Access online help information.

To enter multiple commands on one command line

- **Separate the commands with semicolons (;).**

  More than one command may be entered in a single command line if the commands are separated by semicolons (;).

Examples

To reset the emulator and break into the monitor:

```
reset ; break
```

To recall commands

- **Press <CTRL>r or <CTRL>b.**

The most recent 20 commands you enter are stored in a buffer and may be recalled by pressing <CTRL>r. Pressing <CTRL>b cycles forward through the recall buffer.

Examples

For example, to recall and execute the command prior to the last command:

```
<CTRL>r <CTRL>r
```

To edit commands

- **Use the following keys:**

<Left arrow> <Right arrow>	Move the cursor single spaces to the left or right.
<Tab> <Shift> <Tab>	Move the cursor to the next or previous word on the command line.
<Insert char>	Enters the insert editing mode and allows characters or command options to be inserted at the cursor location.
<Back space>	Deletes the character to the left of the cursor.
<Delete char>	Deletes the character to the right of the cursor.
<Clear line>	Deletes the characters from the cursor to the end of the line.
<CTRL>u	Erases the command line.

To access online help information

- **Use the help or ? commands.**

To access the command line's online 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 display information on that topic.

Examples

To display information on the system commands:

```
help system_commands
```

Or:

```
? system_commands
```

The help information is scrolled on to the screen. If there is more than a screen full of information, you will have to press the space bar to see the next screen full, or the <RETURN> key to see the next line, just as you do with the UNIX 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 command line.

Using Command Files

You can execute a series of commands that have been stored in a command file. You can create command files by logging commands while using the interface or by using an editor on your host computer.

Once you create a command file, you can execute the file in the emulation environment by typing the name of the file on the command line and pressing <RETURN>.

Command files execute until an end-of-file is found or until a syntax error occurs. You can stop a command file by pressing <CTRL>c or the <Break> key.

This section shows you how to:

- Start logging commands to a command file.
- Stop logging commands to a command file.
- Playback (execute) a command file.

Nesting Command Files

You can nest a maximum of eight levels of command files. Nesting command files means one command file calls another.

Comments in Command Files

Text that follows a pound sign (#), up to the end of the line, is interpreted as a comment.

Using the wait Command

When editing command files, you can insert **wait** commands to pause execution of the command file at certain points.

If you press <CTRL>c to stop execution of a command file while the "wait" command is being executed from the command file, the <CTRL>c will terminate the "wait" command, but will not terminate command file execution. To do this, press <CTRL>c again.

Use the **wait measurement_complete** command after changing the trace depth. By doing this, when you copy or display the trace after changing the trace depth, the new trace states will be available. Otherwise the new states won't be available.

Passing Parameters

Command files provide a convenient method for passing parameters by using a parameter declaration line preceding the commands in the command file. When the command file is called, the system will prompt you for current values of the formal parameters listed.

Parameters are defined as:

Passed Parameters These are ASCII strings passed to a command file. Any continuous set of ASCII characters can be passed. Spaces separate the parameters.

Formal Parameters These are symbols preceded by an ampersand (&), which are the variables of the command file.

The ASCII string passed (passed parameter) will be substituted for the formal parameter when the command file is executed.

The only way to pass a parameter containing a space is to enclose the parameter in double quotes (") or single quotes ('). Thus, to pass the parameter HP 9000 to a command file, you can use either "HP 9000" or 'HP 9000'.

The special parameter &ArG_lEfT gets set to all the remaining parameters specified when the command file was invoked. This lets you use variable size parameter lists. If no parameters are left, &ArG_lEfT gets set to NULL.

Consider the command file example (named CMDFILE) shown below:

```
PARMS &ADDR &VALUE1
#
# modify a location or list of locations in memory
# and display the result
#
modify memory &ADDR words to &VALUE1 &ArG_lEfT
```

```
display memory &ADDR blocked words
```

When you execute CMDFILE, you will be prompted with:

```
Define command file parameter [&ADDR]
```

To pass the parameter, enter the address of the first memory location to be modified. You will then be prompted for &VALUE1. If you enter, for example, "0,-1,20, 0ffffh, 4+5*4", the first parameter "0,-1,20," is passed to &VALUE1 and the remaining parameters "0ffffh," and "4+5*4" are passed to &ArG_lEfT.

You can also pass the parameters when you invoke the command file (for example, CMDFILE 1000h 0,-1,20, 0ffffh, 4+5*4).

Other Things to Know About Command Files

You should know the following about using command files:

1 Command files may contain shell variables. Only those shell variables beginning with "$" followed by an identifier will be supported. An identifier is a sequence of letters, digits or underscores beginning with a letter or underscore. The identifier may be enclosed by braces "{ }" or entered directly following the "$" symbol. Braces are required when the identifier is followed by a letter, a digit or an underscore that is not interpreted as part of its name.

For example, assume a directory named /users/softkeys and the shell variable "S". The value of "S" is "soft". By specifying the directory as /users/${S}keys the correct result is obtained. However, if you attempt to specify the directory as /users/$Skeys, the Softkey Interface looks for the value of the variable "Skeys". This is not the operators intended result. You may not get the intended result unless Skeys is already defined to be "softkeys".

You can examine the current values of all shell variables defined in your environment with the command "env".

2 Positional shell variables, such as $1, $2, and so on, are not supported. Neither are special shell variables, such as $@, $*, and so on, supported.

3 You can continue command file lines. This is done by avoiding the line feed with a backslash (\). A line terminated by "\" is concatenated with any following lines until a line that does not contain a backslash is found. A line constructed in this manner is recognized and executed as one single command line. If the last line in a command file is terminated by "\", it appears on the command line but is not executed. Normally, the line feed is recognized as the command terminator. The UNIX environment recognizes three quoting characters for shell commands which are double quotes ("), single quotes ('), and the backslash symbol (\).

For example, the following three lines are treated as a single shell command. The two hidden line feeds are ignored because they are inside the two single quotes ('):

```
!awk '/$/ { blanks++ }
END { print blanks }
' an_unix_file
```

To start logging commands to a command file

- Choose **File→Log→Record** and use the dialog box to select a command file name.
- Using the command line, enter the **log_commands to <file>** command.

To stop logging commands to a command file

- Choose **File→Log→Stop**.

Or, using the command line, enter:

```
log_commands off
```

To playback (execute) a command file

- Choose **File→Log→Playback** ,and use the dialog box to select the name of the command file you wish to execute.

Or, using the command line, enter the name of the command file and press <RETURN>.

If you enter the name of the command file in the command line and the interface cannot find the command file in the current directory, it searches the directories specified in the HP64KPATH environment variable.

To interrupt playback of a command file, press the <CTRL>c key combination. (The mouse pointer must be within the interface window.)

If you press <CTRL>c to stop execution of a command file while the "wait" command is being executed from the command file, the <CTRL>c will terminate the "wait" command, but will not terminate command file execution. To do this, press <CTRL>c again.

Using Pod Commands

Pod commands are Terminal Interface commands. The Terminal Interface is the low-level interface that resides in the firmware of the emulator.

A pod command used in the Graphical User Interface bypasses the interface and goes directly to the emulator. Because some pod commands can cause the interface to become out-of-sync with the emulator, or even cause the interface to terminate abnormally, they must be used with care.

For example, if you change configuration items, the actual state of the emulator will no longer match the internal record the interface keeps about the state of the emulator.

Issuing certain communications-related commands can prevent the interface from communicating with the emulator and cause abnormal termination of the interface.

However, it is sometimes necessary to use pod commands. For example, you must use a pod command to execute the emulator's *performance verification (pv)* routine. Performance verification is an internal self-test procedure for the emulator.

Remember that pod commands can cause trouble for the high-level interface if they are used indiscriminately.

This section shows you how to:

- Display the pod commands screen.
- Use pod commands.

To display the pod commands screen

- Choose **Display→Pod Commands**.

The pod commands screen displays the results of pod (Terminal Interface) commands. To set the interface to use pod commands, choose **Settings→Pod Command Keyboard**.

To use pod commands

- To begin using pod commands, choose **Settings→Pod Command Keyboard**.
- To end using pod commands, click the **suspend** pushbutton softkey.

The **Settings→Pod Command Keyboard** command displays the pod commands screen and activates the keyboard for entering pod command on the command line.

Examples

To see a list of pod command categories available, choose **Settings→Pod Command Keyboard**, and on the command line, type: **help**.

To see a list of pod commands that control the emulator, type: **help emul**.

To see details of the emulator "r" (run user code) command, type: **help r**.

Forwarding Commands to Other HP 64700 Interfaces

To allow the emulator/analyzer interface to run concurrently with other HP 64700 interfaces like the software performance analyzer, a background "daemon" process is necessary to coordinate actions in the interfaces.

This background process also allows commands to be forwarded from one interface to another. Commands are forwarded using the forward command available in the command line. The general syntax is:

```
forward <interface_name> "<command_string>"
```

This section shows you how to:

- Forward commands to the software performance analyzer.

To forward commands to the software performance analyzer

- Enter the **forward perf "<command string>"** command using the command line.

Examples

To send the "profile" command to the software performance analyzer:

```
forward perf "profile"
```

5

Configuring the Emulator

Configuring the Emulator

This chapter describes how to configure the emulator. You must map memory whenever you use the emulator. When you plug the emulator into a target system, you must configure the emulator so that it operates correctly in the target system. The configuration tasks are grouped into the following sections:

- Using the configuration interface.
- Modifying the general items and monitor setup.
- Reconfiguring the emulator copy of the SIM registers.
- Mapping memory.
- Setting the debug/trace options.
- Setting Simulated I/O
- Verifying the emulator configuration.

The simulated I/O feature and configuration questions are described in the *Simulated I/O User's Guide*.

The external analyzer configuration options are described in the "Using the External State Analyzer" chapter.

The interactive measurement configuration options are described in the "Making Coordinated Measurements" chapter.

Using the Configuration Interface

This section shows you how to modify, store, and load configurations using the emulator configuration interface.

This section shows you how to:

- Start the configuration interface.
- Modify a configuration section.
- Apply configuration changes to the emulator.
- Display information if the "apply" didn’t work.
- Store configuration changes to a file.
- Change the configuration directory context.
- Display the configuration context.
- Access help topics.
- Access context sensitive (f1) help.
- Exit the configuration interface.
- Load an existing configuration file.

This section describes emulator configuration in general. The remaining sections in this chapter describe the specific configuration options for your emulator.

To start the configuration interface

- Choose **Modify→Emulator Config...** from the emulator/analyzer interface pull-down menu.

Or, using the command line, enter:

```
modify configuration
```

The configuration interface top-level dialog box (see the following example) is displayed.

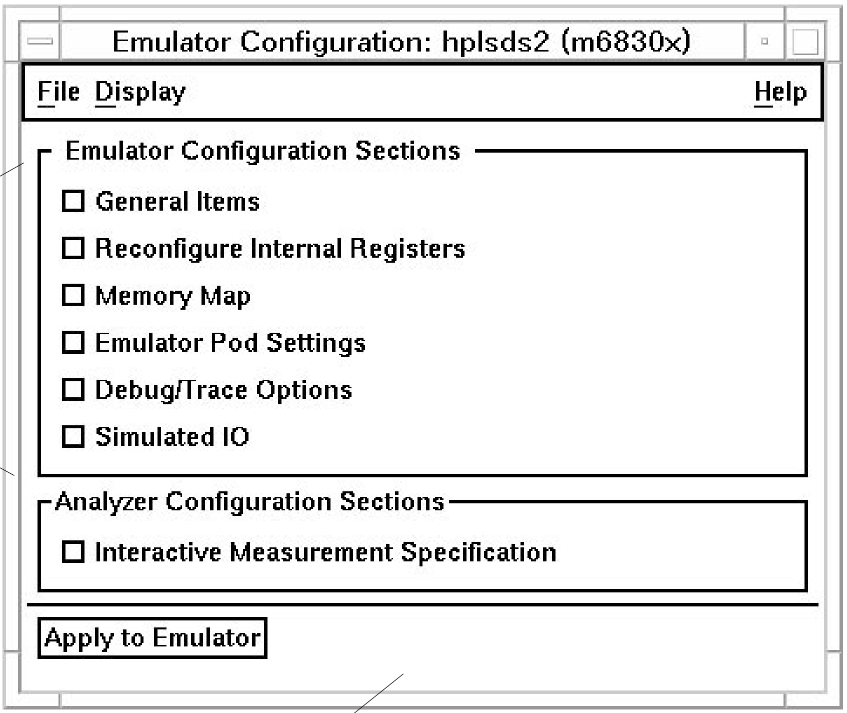
The configuration sections that are presented depend on the hardware and the features of your particular emulator.

The configuration interface may be left running while you are using the emulator/analyzer interface.

If you're using the Softkey Interface from a terminal or terminal emulation window, you don't get a dialog box from which to choose configuration sections; however, you have access to the same configuration options through a series of configuration questions.

Examples

The 6830x emulator configuration interface top-level dialog box is shown below.


Emulator Configuration: hplsds2 (m6830x)
File Display
Help
Emulator Configuration Sections
General Items
Reconfigure Internal Registers
Memory Map
Emulator Pod Settings
Debug/Trace Options
Simulated IO
Analyzer Configuration Sections
Interactive Measurement Specification
Apply to Emulator

The menu bar.

Clicking on one of these lines selects a particular configuration section.

Clicking this button loads configuration changes into the emulator.

This portion of the dialog box displays configuration status information.

To modify a configuration section

1 Start the emulator configuration interface.

2 Click on a section name in the configuration interface top-level dialog box.

3 Use the section dialog box to make changes to the configuration.

As soon as you change a configuration option, the change is recorded (as seen by the "Changes Not Loaded" message in the top level dialog).

If you're using the Softkey Interface:

The configuration questions in the "General Items" section are the first to be asked.

To access the questions in the "Reconfigure Internal Registers" section, answer "yes" to the "Reconfigure internal registers?" question.

To access the questions in the "Memory Map" section, answer "yes" to the "Modify memory configuration?" question.

To access the questions in the "Simulated IO" section, answer "yes" to the "Modify Simulated IO?" question.

To access the questions in the "Debug/Trace Options" section, answer "yes" to the "Modify debug/trace options?" question.

4 Apply the configuration changes to the emulator.

To apply configuration changes to the emulator

- **Click the "Apply to Emulator" button in the top-level dialog box.**

Loads the configuration changes into the emulator. Status text to the right shows whether the load was successful.

You can apply configuration changes to the emulator at any time (even while section dialog boxes are open). This lets you verify changes without closing section dialog boxes.

The "Apply to Emulator" button does not store configuration changes to a file.

When you exit the configuration interface and there are configuration changes that have not been stored, you are asked whether you want to store the changes, exit without storing, or cancel the exit.

If apply to emulator fails

Choose Display→Failed Apply Info from the pull-down menu in the top-level configuration interface window.

A window containing the following information about the failed configuration is opened:

- Chip select information from the emsim (emulator) resister set.
- Bus interface port information from the emsim (emulator) resister set.
- The expanded memory map.
- Reset mode configuration information.
- A complete list of the configuration inconsistencies. This list is not limited to 16 messages as is the **Display→Configuration Info→Diagnostics** command.

This information is shown in the same format as output from the various **Display→Configuration Info→** commands.

Because the old configuration is reloaded when an apply to emulator fails, the information displayed in this window is different from the information displayed by the **Display→Configuration Info→** commands (which display information about the configuration currently loaded).

Refer to the "Verifying the Emulator Configuration" section later in this chapter for details on these types of configuration information displays.

To store configuration changes to a file

- Choose **File→Store...** from the pull-down menu in the top-level configuration interface window, and use the file selection dialog box to name the configuration file.
- If you're using the Softkey Interface, the last configuration question, "Configuration file name?", lets you name the file to which configuration information is stored. If you don't enter a name, configuration information is saved to a temporary file (which is deleted when you exit the interface and release the emulation system).

When modifying a configuration using the graphical interface, you can store your answers at any time.

Configuration information is saved in a file with the extension ".EA". This file is the "source", ASCII format copy of the file. (The interface will create a temporary file with the extension ".EB" which is the "binary" or loadable copy of the file.)

CAUTION Do not modify configurations by editing the ".EA" files. Use the configuration interface to modify and save configurations.

For more information on how to use dialog boxes, refer to the "To use dialog boxes" description in the "Using Menus, the Entry Buffer, and Action Keys" section of the "Entering Commands" chapter.

To change the configuration directory context

- **Choose File→Directory... from the pull-down menu in the top-level configuration interface window, and use the directory selection dialog box to specify the new directory.**

The directory context specifies the directory to which configuration files are stored and from which they are loaded.

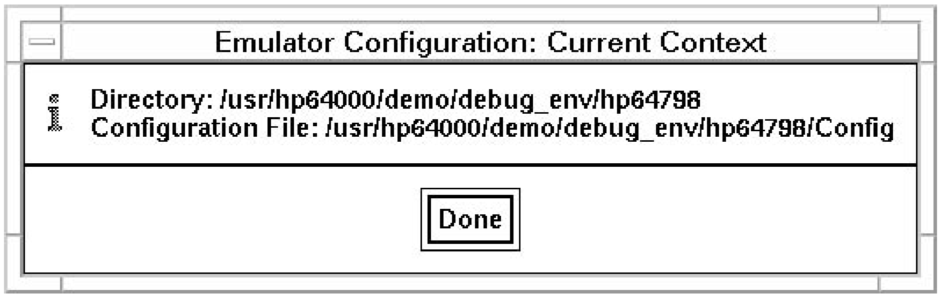
For more information on how to use dialog boxes, refer to the "To use dialog boxes" description in the "Using Menus, the Entry Buffer, and Action Keys" section of the "Entering Commands" chapter.

To display the configuration context

- **Choose Display→Context... from the pull-down menu in the top-level configuration interface window.**

The current directory context and the current configuration files are displayed in a window. Click the "Done" pushbutton when you wish to close the window.

Figure 15


Emulator Configuration: Current Context

Directory: /usr/hp64000/demo/debug_env/hp64798
Configuration File: /usr/hp64000/demo/debug_env/hp64798/Config

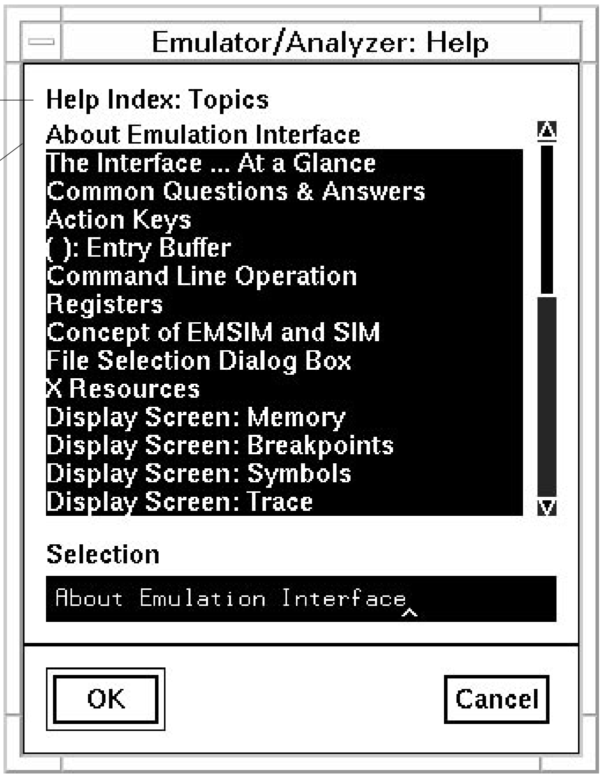
To access help topics

- Choose **Help→General Topic...** from the pull-down menu in the top-level configuration interface window, click on a topic in the selection dialog box, and click the "OK" button.

Figure 16

Displays help on main items.

Displays help about the specific topic.



To access context sensitive (f1) help

- Place the mouse pointer over the item you're interested in, and press the f1 key.
- Choose **Help→On Item...** from the pull-down menu in the top-level configuration interface window. Notice that the mouse pointer changes from an arrow to a question mark. Move the question mark mouse pointer over the item you're interested in, and click any mouse button.

The configuration interface provides context sensitive help in the top level dialog box and throughout the configuration section dialog boxes.

To exit the configuration interface

- Choose **File→Exit...** from the pull-down menu in the top-level configuration interface window (or type <CTRL>x).

If configuration changes have not been stored to a file, a confirmation dialog box appears, giving you the options of: storing, exiting without storing, or canceling the exit.

To load an existing configuration file

- In the emulator/analyzer interface, choose **File→Load→Emulator Config...** from the pull-down menu, and use the file selection dialog box to specify the configuration file to be loaded.

Or, using the command line, enter:

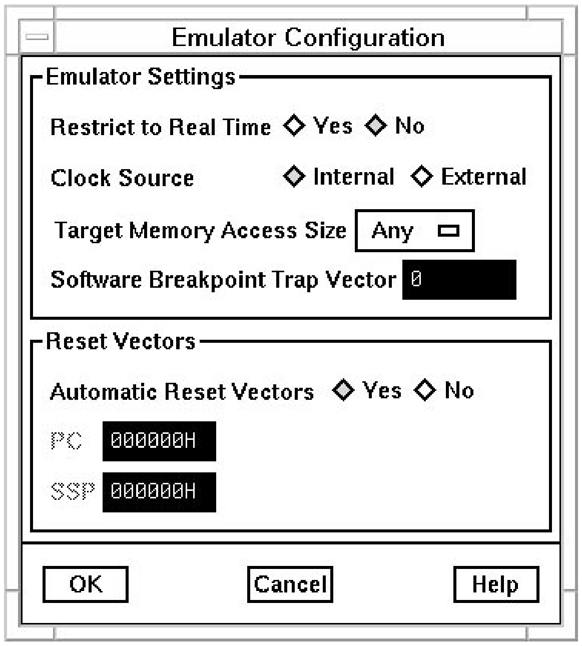
```
load configuration <FILE>
```

This command loads previously created and stored configuration files. You cannot load a configuration while the configuration interface is running.

Modifying the General Items and Monitor Setup

To modify the general configuration items, first start the configuration interface and access the "General Items/Monitor Setup" configuration section (refer to the previous "Using the Configuration Interface" section).

Figure 17



Emulator Configuration

Emulator Settings

Restrict to Real Time ◇ Yes ◇ No

Clock Source ◇ Internal ◇ External

Target Memory Access Size: Any

Software Breakpoint Trap Vector: 0

Reset Vectors

Automatic Reset Vectors ◇ Yes ◇ No

PC: 000000H

SSP: 000000H

To access information about the configuration items, use the online help button or press f1 to show context-sensitive help on an individual item.

When Restricting the Emulator to Real-time Runs

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

The default configuration does not restrict the emulator to real-time runs. Therefore, the emulator might make temporary breaks into the monitor to complete certain commands. However, you may wish to restrict runs to real time to prevent temporary breaks that might cause target system problems.

When runs are restricted to real time and the emulator is running the user program, all commands that cause a break (except ***reset***, ***break***, ***run***, and ***step*** are refused.

The following commands are not allowed when runs are restricted to real time and the emulator is running the user program:

- Display/modify registers.
- Display/modify target system memory.
- Load/store target system memory.
- Modify SIM registers.
- Display emulator SIM configuration info.

If you want to enter one of these commands, you must first make an explicit break into the monitor using the break command.

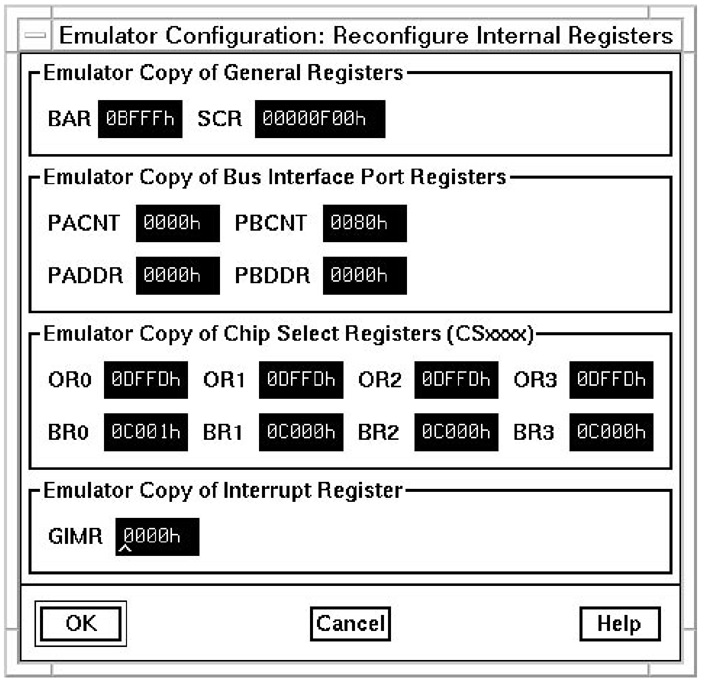
Because the emulator contains dual-port emulation memory, commands that access emulation memory are allowed while runs are restricted to real time.

When the restriction to real-time runs is turned off, all commands, regardless of whether or not they require a break to the emulation monitor, are accepted by the emulator.

Reconfiguring the Emulator Copy of the SIM Registers

To reconfigure the emulator copy of the SIM registers, first start the configuration interface and access the "Reconfigure Internal Registers" configuration section (refer to the previous "Using the Configuration Interface" section).

Figure 18


Emulator Configuration: Reconfigure Internal Registers

Emulator Copy of General Registers

BAR 0BFFFh SCR 00000F00h

Emulator Copy of Bus Interface Port Registers

PACNT 0000h PBCNT 0080h

PADDR 0000h PBDDR 0000h

Emulator Copy of Chip Select Registers (CSxxxx)

OR0 0DFFDh OR1 0DFFDh OR2 0DFFDh OR3 0DFFDh

BR0 0C001h BR1 0C000h BR2 0C000h BR3 0C000h

Emulator Copy of Interrupt Register

GIMR 0000h

To access information about the configuration items, use the online help button or press f1 to show context-sensitive help on an individual item.

To define values for the emulator copy of the SIM registers

- **Click on the register field and enter the desired value. Then apply the changes to the emulator.**

Refer to the "Using the EMSIM Registers" section in the "Using the Emulator" chapter for information on how these registers are used. Refer to the "Concepts" chapter for conceptual information about these registers.

Mapping Memory

Because the emulator can use target system memory or emulation memory (or both), it is necessary to map ranges of memory so that the emulator knows where to direct its accesses.

Up to 8 ranges of memory can be mapped, and the resolution of mapped ranges is 256 bytes (that is, the memory ranges must begin on 256-byte boundaries and must be at least 256 bytes in length).

Emulation memory is made available to the mapper in 64-Kbyte blocks. When you map an address range to emulation memory, at least one block is assigned to the range. When a block of emulation memory is assigned to a range, it is no longer available, even though part of the block may be unused.

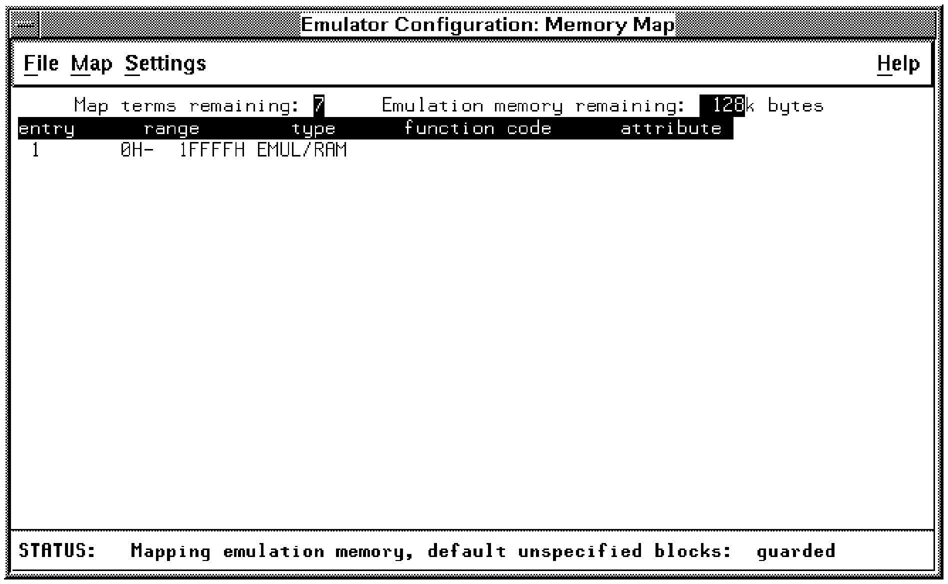
Direct memory access (DMA) to emulation memory is not permitted.

You should map all memory ranges used by your programs before loading programs into memory.

In order to map memory, you must first start the configuration interface and access the "Memory Map" configuration section (refer to the previous "Using the Configuration Interface" section).

When you select the "Memory Map" configuration option, the following window appears:

Figure 19



```
    Map terms remaining: 7     Emulation memory remaining:  128k bytes
entry       range         type       function code       attribute
 1       0H-   1FFFFH EMUL/RAM
```

```
STATUS:   Mapping emulation memory, default unspecified blocks:  guarded
```

This section shows you how to:

- Add memory map entries.
- Modify memory map entries.
- Delete memory map entries.
- Characterize unmapped ranges.
- Map memory ranges that use function codes.

To add memory map entries

- Choose **Map→Add New Entry** from the pull-down menu in the memory map window.
- Press and hold the *select* mouse button and choose **Add New Entry** from the pop-up menu.
- Using the command line (**Settings→Command Line**), enter the address range, memory type, and possibly a blk1-blk8 attribute for emulation memory ranges.

You can characterize memory ranges as emulation RAM, emulation ROM, target system RAM, target system ROM, or as guarded memory.

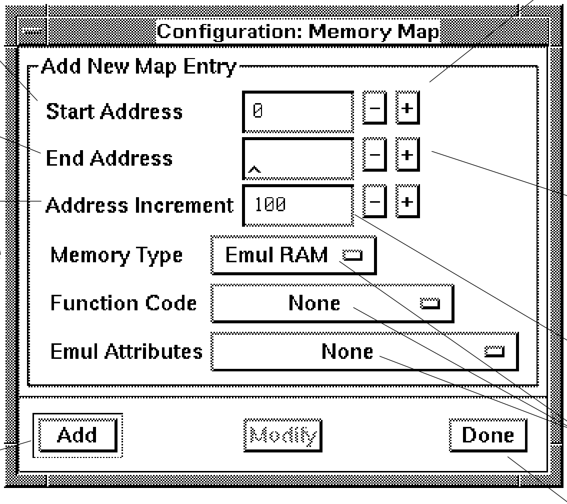
Guarded memory accesses will cause emulator execution to break into the monitor program.

Writes to locations characterized as ROM will cause emulator execution to break into the monitor program if the "Break processor on write to ROM" trace/debug configuration option is enabled.

Writes to emulation ROM will be inhibited. Writes by user code to target system memory locations mapped as ROM or guarded memory will result in a break to the monitor but are not inhibited (that is, the write still occurs).

The first two methods of mapping memory ranges give you the following dialog box.

Figure 20



Configuration: Memory Map

Add New Map Entry

Start Address: 0

End Address:

Address Increment: 100

Memory Type: Emul RAM

Function Code: None

Emul Attributes: None

The starting address of the range to be added.

The ending address of the range to be added.

Specifies the increment value for the "+" and "-" buttons of the start and end address fields.

Adds the defined range to the memory map.

Subtract or add the address increment value. The end address is changed by the same amount, thereby moving the block of memory.

Change only the end address, thereby changing the size of the block of memory.

Multiply or divide the increment value by 2.

These buttons may be held down to repeat the action.

Closes the dialog box.

Examples

Consider the following section summary from the linker load map output listing.

```
SECTION SUMMARY
---------------

SECTION    ATTRIBUTE                 START      END        LENGTH     ALIGN

           ABSOLUTE DATA             00000000   0000002F   00000030   0 (BYTE)
0          NORMAL                    00000030   00000030   00000000   2 (WORD)
env        NORMAL CODE               00000400   00000FC0   00000BC1   2 (WORD)
prog       NORMAL CODE               00000FC2   00001A89   00000AC8   2 (WORD)
const      NORMAL ROM                00001A8A   00001ACF   00000046   2 (WORD)
lib        NORMAL CODE               00001AD0   00002663   00000B94   2 (WORD)
libc       NORMAL CODE               00002664   00004881   0000221E   2 (WORD)
libm                                 00004882   00004882   00000000   0 (BYTE)
mon        NORMAL CODE               00004882   000049CB   0000014A   2 (WORD)
envdata    NORMAL DATA               00007000   00007155   00000156   4 (LONG)
data       NORMAL DATA               00007156   00007721   000005CC   2 (WORD)
idata                                00007722   00007722   00000000   0 (BYTE)
udata                                00007722   00007722   00000000   0 (BYTE)
libdata    NORMAL DATA               00007724   00007727   00000004   4 (LONG)
libcdata   NORMAL DATA               00007728   00008153   00000A2C   2 (WORD)
mondata    NORMAL DATA               00008154   00008177   00000024   2 (WORD)
stack      NORMAL DATA               0000B000   00012FFF   00008000   4 (LONG)
heap       NORMAL DATA               00013000   00016FFD   00003FFE   4 (LONG)
```

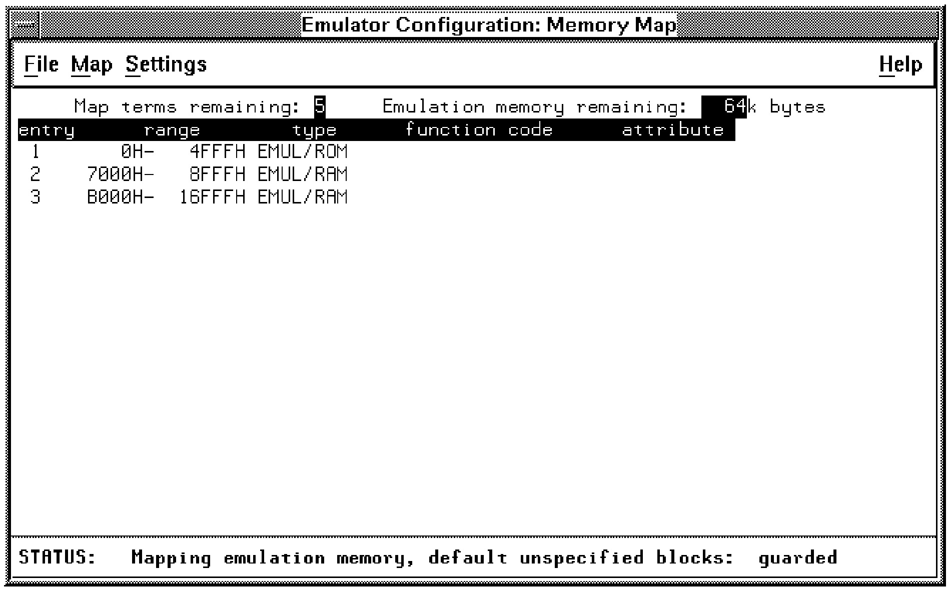
Notice the ABSOLUTE DATA, CODE, and ROM sections occupy locations 0 through 49CBH. Because the contents of these sections will eventually reside in target system ROM, this area should be characterized as ROM when mapped. This will prevent these locations from being written over accidentally. If breaks on writes to ROM are enabled, instructions that attempt to write to these locations will cause emulator execution to break into the monitor.

Also, notice the DATA sections occupy locations 7000H through 8177H and 0B000H through 16FFDH. Since these sections are written to, they should be characterized as RAM when mapped.

Using the command line (choose **Settings→Command Line** from the pull-down menu in the memory map window), enter the following commands to map memory for the above program.

```
delete all
<addr> 0 thru 4fffh emulation rom
7000h thru 8fffh emulation ram
0b000h thru 16fffh emulation ram
```

The resulting memory mapper screen is shown below.



```
    Map terms remaining: 5     Emulation memory remaining:   64k bytes
entry      range         type       function code      attribute
 1       0H-   4FFFH EMUL/ROM
 2    7000H-   8FFFH EMUL/RAM
 3    B000H-  16FFFH EMUL/RAM

STATUS:   Mapping emulation memory, default unspecified blocks:  guarded
```

To exit out of the memory mapper, enter:

```
end
```

To modify memory map entries

- Choose **Map→Modify Entry** from the pull-down menu in the memory map window and select the entry number from the cascade menu.
- Position the mouse pointer over the entry you wish to modify, press and hold the *select* mouse button and choose **Modify Entry** from the pop-up menu.

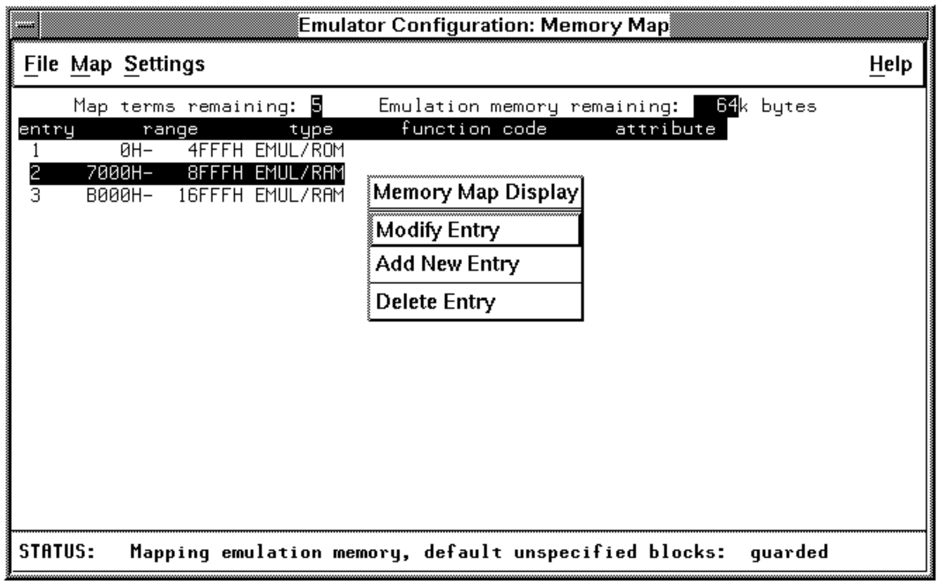
These commands open the same dialog box that is used for adding memory map entries, except it lets you modify the current settings for the entry.

In order to modify an entry when using the command line, you must delete the entry and add a new entry.

Examples

To modify a memory map entry using the pop-up menu:

Click and hold the mouse select button to bring up the menu. Then choose the Modify Entry item to modify the highlighted memory map entry.



```
     Map terms remaining: 5     Emulation memory remaining:   64k bytes
entry      range        type       function code      attribute
 1       0H-    4FFFH EMUL/ROM
 2    7000H-    8FFFH EMUL/RAM
 3    B000H-   16FFFH EMUL/RAM
```

```
STATUS:   Mapping emulation memory, default unspecified blocks:  guarded
```

Use the dialog box to modify the entry, and click the "Modify" button to add the modified range to the memory map.

To delete memory map entries

- Choose **Map→Delete Entry** from the pull-down menu in the memory map window and select the entry number from the cascade menu.
- Position the mouse pointer over the entry you wish to delete, press and hold the *select* mouse button and choose **Delete Entry** from the pop-up menu.

Or, using the command line, enter:

```
delete <ENTRY#>
```

Note that programs should be reloaded after deleting mapper terms. The memory mapper may re-assign blocks of emulation memory after the insertion or deletion of mapper terms.

To characterize unmapped ranges

- Choose **Map→Default Memory Type** from the pull-down menu in the memory map window and select the memory type from the cascade menu.

Or, using the command line, enter:

```
 default <memory_type>
```

Unmapped memory ranges are treated as target system RAM by default. Unmapped memory ranges cannot be characterized as emulation memory.

To map memory ranges that use function codes

- **Specify function codes with address ranges when mapping memory.**

The function code can be:

- None
- supervisor
- supervisor program
- supervisor data
- user
- user program
- user data
- program
- data

Function code information lets you further characterize memory blocks as supervisor, user, supervisor program, supervisor data, user program, or user data space. When you specify function codes with mapper ranges, the 6830x function code outputs (FC0, FC1, FC2) are decoded to select particular blocks of memory. Function codes let you overlay address ranges. When you specify function codes as part of the address, the emulator memory mapper knows that overlaid blocks are different memory regions and will define them separately.

If you specify a function code when mapping a range of memory, you must include the function code when referring to locations in that range. If you don't include the function code, an "ambiguous address" error message is displayed.

If you use different function codes, it's possible to map address ranges that overlap. When address ranges with different function codes overlap, you must load a separately linked module for the space associated with each function code. The modules are linked separately because linker errors occur when address ranges overlap.

When address ranges are mapped with different function codes, and there are no overlapping ranges, your program modules may exist in one absolute file. However, you have to use multiple ***load*** commands—one for each function code specifier. This is necessary to load the various sections of the absolute file into the appropriate function code qualified memory ranges.

When you do this, be sure that all address ranges not mapped (that is, the "other" memory mapper term) are mapped as target RAM. When "other" is mapped as guarded, guarded memory access errors (from the attempt to load the absolute file sections that are outside the specified function code range) can prevent the absolute file sections that are inside the specified function range from being loaded.

Examples

Suppose you're developing a system with the following characteristics:

- Input port at 100 hex.
- Output port at 400 hex.
- Supervisor program from 1000 through 1fff hex.
- Supervisor data from 2000 through 2fff hex.
- User program from 3000 through 3fff hex.
- User data from 3000 through 3fff hex.

The last two terms have address ranges that overlap. You can use function codes to cause these terms to be mapped to different blocks of memory.

Suppose also that the only things that exist in your target system at this time are the input and output ports and some control logic; no memory is available. You can reflect this by mapping the I/O ports to target system memory space and the rest of memory to emulation memory space by entering the following mapper commands using the command line (**Settings→Command Line**):

```
0h thru 0fffh target ram
1000h thru 1fffh supervisor program emulation rom
2000h thru 2fffh supervisor data emulation ram
3000h thru 3fffh user program emulation ram
3000h thru 3fffh user data emulation ram
```

After the configuration is saved, display memory at 1000H by entering the following command (using the command line):

```
display memory 1000h blocked bytes
```

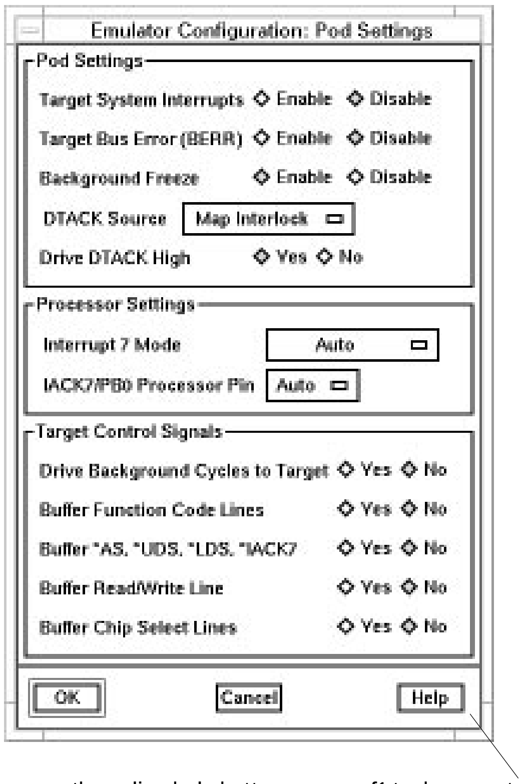
Notice that an "ambiguous address" error occurs because the "sp" function code was not included with the address. The following command should have been entered instead:

```
display memory fcode sp 1000h blocked bytes
```

Modifying the Emulator Pod Settings

To modify the emulator pod settings, you must start the configuration interface and access the "Debug/Trace Options" configuration section (refer to the "Using the Configuration Interface" section).

Figure 21



Emulator Configuration: Pod Settings

Pod Settings

Target System Interrupts ◇ Enable ◇ Disable

Target Bus Error (BERR) ◇ Enable ◇ Disable

Background Freeze ◇ Enable ◇ Disable

DTACK Source: Map Interlock

Drive DTACK High ◇ Yes ◇ No

Processor Settings

Interrupt 7 Mode: Auto

IACK7/PB0 Processor Pin: Auto

Target Control Signals

Drive Background Cycles to Target ◇ Yes ◇ No

Buffer Function Code Lines ◇ Yes ◇ No

Buffer *AS, *UDS, *LDS, *IACK7 ◇ Yes ◇ No

Buffer Read/Write Line ◇ Yes ◇ No

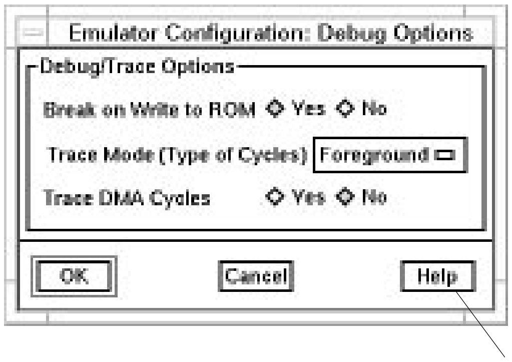
Buffer Chip Select Lines ◇ Yes ◇ No

To access information about the configuration items, use the online help button or press f1 to show context-sensitive help on an individual item.

Setting the Debug/Trace Options

To set the debug/trace options, you must start the configuration interface and access the "Debug/Trace Options" configuration section (refer to the "Using the Configuration Interface" section).

Figure 22


Emulator Configuration: Debug Options
Debug/Trace Options
Break on Write to ROM ◇ Yes ◇ No
Trace Mode (Type of Cycles) Foreground
Trace DMA Cycles ◇ Yes ◇ No
OK
Cancel
Help

To access information about the configuration items, use the online help button or press f1 to show context-sensitive help on an individual item.

To configure breaks on writes to ROM

When breaks on writes to ROM are enabled, the emulator will break into the emulation monitor whenever the user program attempts to write to a memory region mapped as ROM. The emulator will prevent the processor from actually writing to memory mapped as emulation ROM; however, it cannot prevent writes to target system RAM locations which are mapped as ROM, even though the write to ROM break is enabled.

When breaks on writes to ROM are disabled, 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.

To configure the trace mode

"Background" specifies that the analyzer trace only background cycles. This is rarely a useful setting for user program debugging.

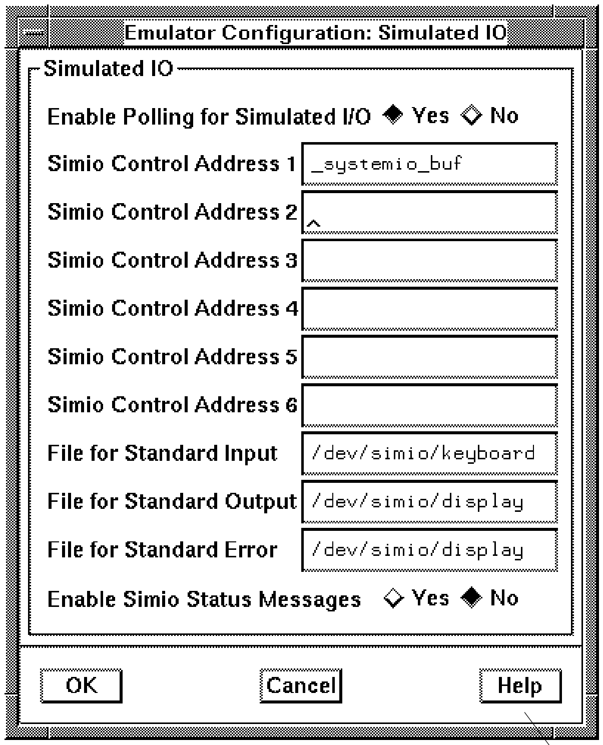
"Both" specifies that the analyzer trace both foreground and background cycles. You may wish to specify this option so that all emulation processor cycles may be viewed in the trace display.

"Foreground" specifies that the analyzer trace only foreground cycles.

Setting Simulated I/O

To set the debug/trace options, you must start the configuration interface and access the "Simulated I/O " configuration section (refer to the "Using the Configuration Interface" section).

Figure 23



Enable Polling for Simulated I/O [✓ Yes] No
Simio Control Address 1: _systemio_buf
Simio Control Address 2:
Simio Control Address 3:
Simio Control Address 4:
Simio Control Address 5:
Simio Control Address 6:
File for Standard Input: /dev/simio/keyboard
File for Standard Output: /dev/simio/display
File for Standard Error: /dev/simio/display
Enable Simio Status Messages Yes [✓ No]

To access information about the configuration items, use the online help button or press f1 to show context-sensitive help on an individual item.

Verifying the Emulator Configuration

The 6830x emulator lets you display information about emulator configuration and processor SIM programming. You can also display information about inconsistencies found in the emulator configuration.

This section shows you how to:

- Display information about chip selects.
- Display information about bus interface ports.
- Display information about the memory map.
- Display information about the reset mode configuration.
- Display assembly language instructions for setting up the SIM.
- Check for configuration inconsistencies.

To display information about chip selects

- Choose **Display→Configuration Info→Chip Selects (SIM)** or **Display→Configuration Info→Chip Selects (Emulator SIM)** from either the configuration interface the emulator/analyzer interface pull-down menu.

Or, using the command line, enter:

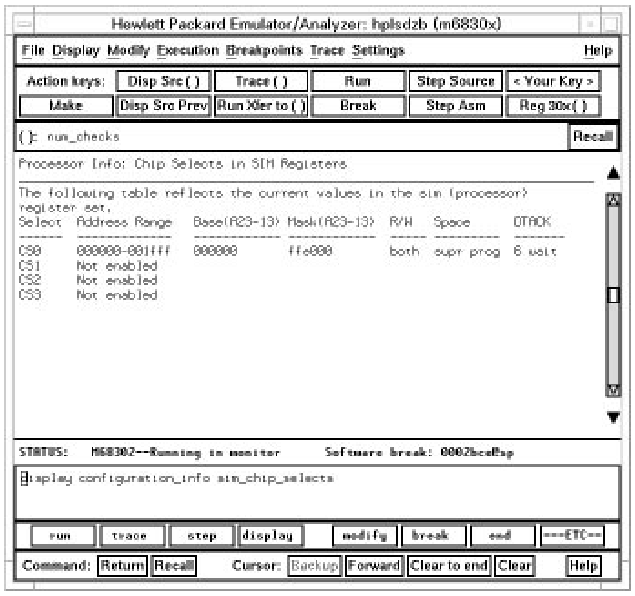
```
display configuration_info sim_chip_selects or
display configuration_info emsim_chip_selects
```

These commands let you display chip select information from the sim (processor) register set or the emsim (emulator) register set.

The resulting display shows how the chip select is assigned, the base address, the block size, and other information from the option register.

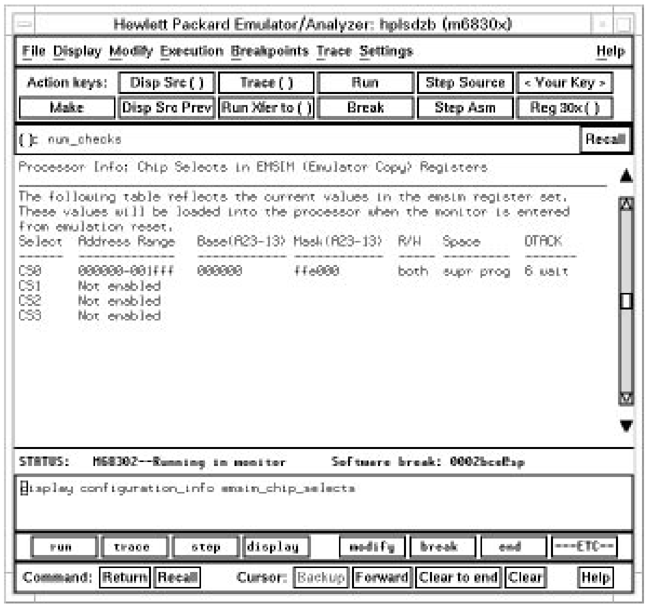
Examples

To display information about chip selects from the sim (processor) register set, choose **Display→Configuration Info→Chip Selects (SIM)** from either the configuration interface the emulator/analyzer interface pull-down menu.


Hewlett Packard Emulator/Analyzer: hplsdzb (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x( )
( ): num_checks
Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
Processor Info: Chip Selects in SIM Registers
-----------------------------------------------------------------
The following table reflects the current values in the sim (processor)
register set.
Select  Address Range   Base(A23-13) Mask(A23-13)  R/W   Space      DTACK
------  -------------   ------------ ------------  ----- ---------  -------
CS0     000000-001fff   000000       ffe000        both  supr prog  6 wait
CS1     Not enabled
CS2     Not enabled
CS3     Not enabled

STATUS:    M68302--Running in monitor       Software break: 0002bce0sp

display configuration_info sim_chip_selects
```

To display information about chip selects from the emsim (emulator) register set, choose **Display→Configuration Info→Chip Selects (Emulator SIM)** from either the configuration interface the emulator/analyzer interface pull-down menu.


Hewlett Packard Emulator/Analyzer: hplsdzb (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x( )
( ): num_checks Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
Processor Info: Chip Selects in EMSIM (Emulator Copy) Registers
________________________________________________________________
The following table reflects the current values in the emsim register set.
These values will be loaded into the processor when the monitor is entered
from emulation reset.
Select  Address Range   Base(A23-13) Mask(A23-13)  R/W   Space      DTACK
------  -------------   ------------ ------------  ----- ---------  -------
CS0     000000-001fff   000000       ffe000        both  supr prog  6 wait
CS1     Not enabled
CS2     Not enabled
CS3     Not enabled

STATUS:   M68302--Running in monitor       Software break: 0002bce8sp

display configuration_info emsim_chip_selects
```

To display information about bus interface ports

- Choose **Display→Configuration Info→Bus Interface Port (SIM)** or **Display→Configuration Info→Bus Interface Port (Emulator SIM)** (where Port is your port of interest) from either the configuration interface the emulator/analyzer interface pull-down menu.

  Or, using the command line, enter:

```
display configuration_info bus_interface_port or
display configuration_info embus_interface_port
```

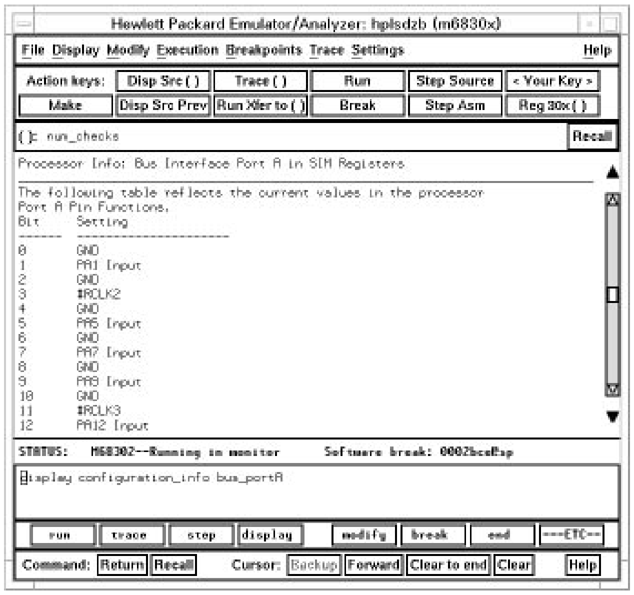
where port is your port of interest.

These commands let you display chip select information from the sim (processor) register set or the emsim (emulator) register set for your port of interest.

The resulting display shows the pin assignments for the port you have chosen.

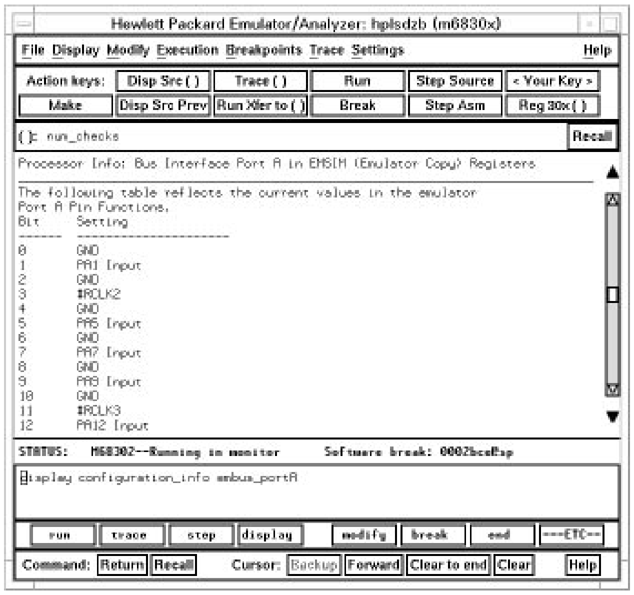
Examples

To display information about bus interface port A from the sim (processor) register set, choose **Display→Configuration Info→Bus Interface Port A (SIM)** from either the configuration interface the emulator/analyzer interface pull-down menu.


Hewlett Packard Emulator/Analyzer: hplsdzb (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x( )
( ): num_checks Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
Processor Info: Bus Interface Port A in SIM Registers

The following table reflects the current values in the processor
Port A Pin Functions.
Bit     Setting
------  ----------------------
0       GND
1       PA1 Input
2       GND
3       #RCLK2
4       GND
5       PA5 Input
6       GND
7       PA7 Input
8       GND
9       PA9 Input
10      GND
11      #RCLK3
12      PA12 Input

STATUS:   M68302--Running in monitor      Software break: 0002bca@sp

display configuration_info bus_portA
```

To display information about bus interface port A from the emsim (emulator) register set, choose **Display→Configuration Info→Bus Interface Port A (Emulator SIM)** from either the configuration interface the emulator/analyzer interface pull-down menu.


Hewlett Packard Emulator/Analyzer: hplsdzb (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x ( )
( ): num_checks Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
Processor Info: Bus Interface Port A in EMSIM (Emulator Copy) Registers

The following table reflects the current values in the emulator
Port A Pin Functions.
Bit      Setting
------   ---------------------
0        GND
1        PA1 Input
2        GND
3        #RCLK2
4        GND
5        PA5 Input
6        GND
7        PA7 Input
8        GND
9        PA9 Input
10       GND
11       #RCLK3
12       PA12 Input

STATUS:   M68302--Running in monitor      Software break: 0002bca8sp

display configuration_info embus_portA
```

To display information about the memory map

- Choose **Display→Configuration Info→Memory Map** from either the configuration interface the emulator/analyzer interface pull-down menu.

  Or, using the command line, enter:

  display configuration_info memory_map

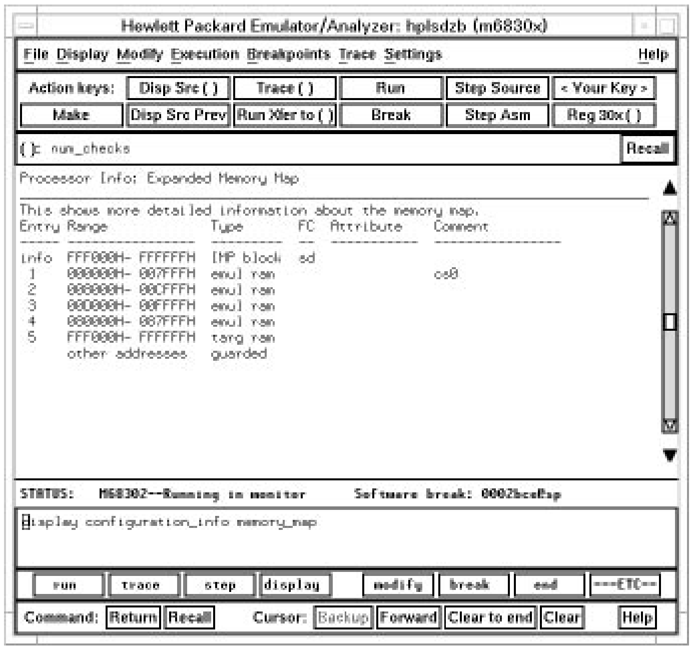
When in the memory map section of the emulator configuration, the ranges of memory that have been mapped are displayed.

The memory map configuration information shows detailed information about the memory map and how actual mapper resources are allocated due to the current programming of the chip selects in the EMSIM register sets.

When the emulator automatically expands the memory map to assign actual mapper resources, it looks at ranges that have been mapped during emulator configuration.

Examples

To display information about the memory map and its correlation with RAM, choose **Display→Configuration Info→Memory Map** from either the configuration interface the emulator/analyzer interface pull-down menu.


Hewlett Packard Emulator/Analyzer: hplsdzb (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x( )
( ): num_checks
Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
Processor Info: Expanded Memory Map

This shows more detailed information about the memory map.
Entry Range              Type       FC  Attribute    Comment
----- ---------------    ---------  --  -----------  ---------------
info  FFF000H- FFFFFFH   IMP block  sd
 1    000000H- 007FFFH   emul ram                    cs0
 2    008000H- 00CFFFH   emul ram
 3    00D000H- 00FFFFH   emul ram
 4    080000H- 087FFFH   emul ram
 5    FFF000H- FFFFFFH   targ ram
      other addresses    guarded

STATUS:   M68302--Running in monitor      Software break: 0002bce@sp

display configuration_info memory_map
```

Notice the entry labeled "info". Ranges with this label do not take up mapper resources; they just show information about the processor's address space. Another "info" entry will be listed if the internal RAM is enabled with the EMRAMBAR register.

To display information about the reset mode configuration

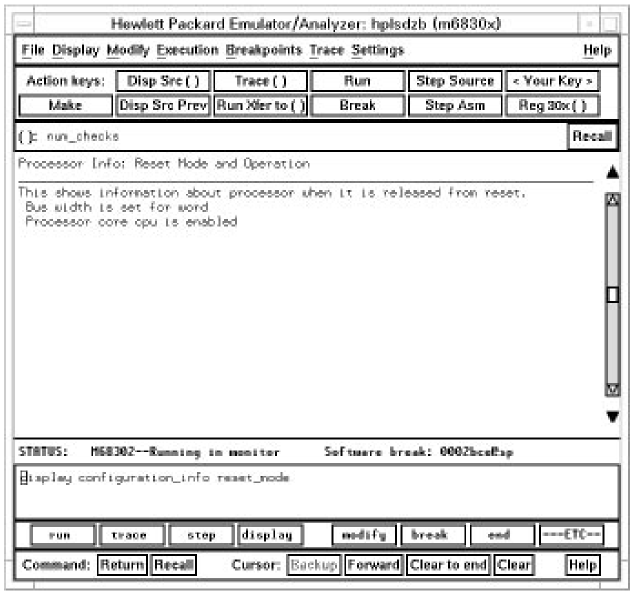
- Choose **Display→Configuration Info→Reset Mode Value** from either the configuration interface the emulator/analyzer interface pull-down menu.

  Or, using the command line, enter:

```
display configuration_info reset_mode
```

Examples

To display information about the reset mode configuration, choose **Display→Configuration Info→Reset Mode Value** from either the configuration interface or the emulator/analyzer interface pull-down menu.


Hewlett Packard Emulator/Analyzer: hplsdzb (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x( )
( ): num_checks Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
Processor Info: Reset Mode and Operation

This shows information about processor when it is released from reset.
 Bus width is set for word
 Processor core cpu is enabled
```

```
STATUS:   M68302--Running in monitor      Software break: 0002bce@sp
```

```
display configuration_info reset_mode
```

To display assembly language instructions for setting up the SIM

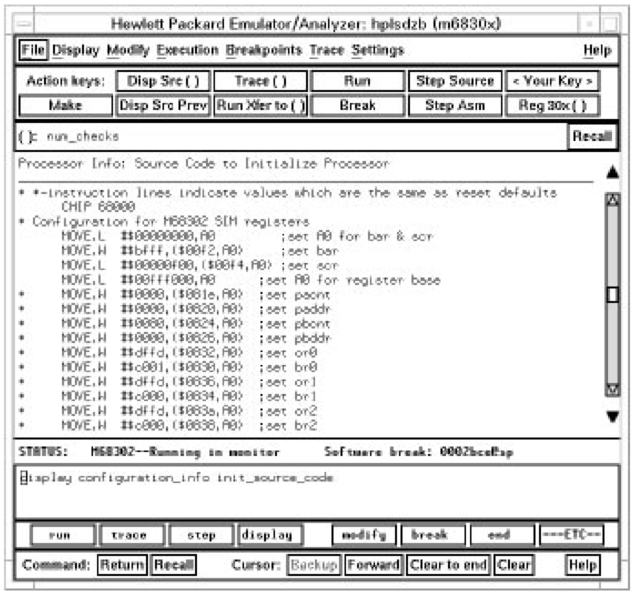
- Choose **Display→Configuration Info→Initialization Source Code** from either the configuration interface the emulator/analyzer interface pull-down menu.

Or, using the command line, enter:

display configuration_info init_source_code

This command displays the assembly language program that will initialize the processor as defined by the current EMSIM register contents.

Examples


Hewlett Packard Emulator/Analyzer: hplsdzb (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x ( )
( ): num_checks
Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
Processor Info: Source Code to Initialize Processor
* *-instruction lines indicate values which are the same as reset defaults
     CHIP 68000
* Configuration for M68302 SIM registers
     MOVE.L  #$00000000,A0         ;set A0 for bar & scr
     MOVE.W  #$bfff,($00f2,A0)     ;set bar
     MOVE.L  #$00000f00,($00f4,A0) ;set scr
     MOVE.L  #$00fff000,A0        ;set A0 for register base
*    MOVE.W  #$0000,($081e,A0)    ;set pacnt
*    MOVE.W  #$0000,($0820,A0)    ;set paddr
*    MOVE.W  #$0000,($0824,A0)    ;set pbcnt
*    MOVE.W  #$0000,($0826,A0)    ;set pbddr
*    MOVE.W  #$dffd,($0832,A0)    ;set or0
*    MOVE.W  #$c001,($0830,A0)    ;set br0
*    MOVE.W  #$dffd,($0836,A0)    ;set or1
*    MOVE.W  #$c000,($0834,A0)    ;set br1
*    MOVE.W  #$dffd,($083a,A0)    ;set or2
*    MOVE.W  #$c000,($0838,A0)    ;set br2

STATUS:   M68302--Running in monitor      Software break: 0002bca8sp

display configuration_info init_source_code
```

To check for configuration inconsistencies

- **Choose Display→Configuration Info→Diagnostics from the configuration interface or the emulator interface pull-down menu.**

Or, using the command line, enter:

```
display configuration_info diagnostics
```

This command:

- Checks for inconsistencies between the mapper and the EMSIM registers.
- Checks for inconsistencies between the reset mode configuration value and the EMSIM registers.
- Compares corresponding values in the SIM and EMSIM register sets.

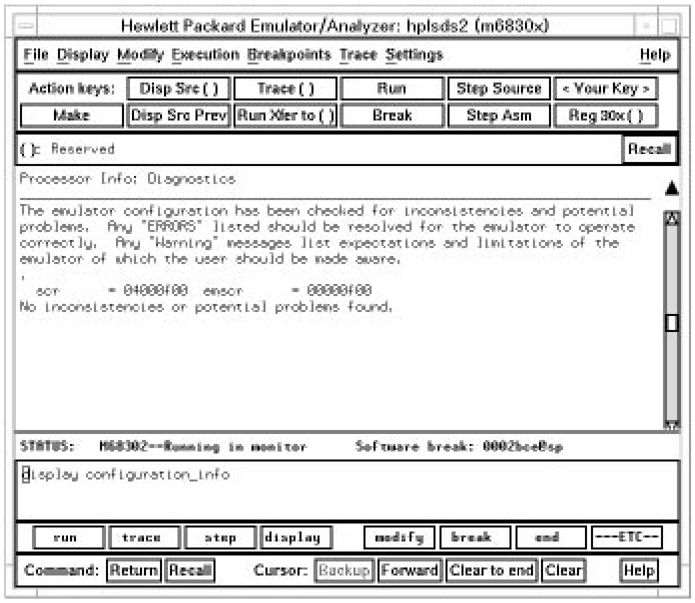
This command identifies errors that result from inconsistencies between related configuration values. These errors should be resolved in order for the emulator to operate correctly.

This command also provides status and warning messages about expectations and limitations of the emulator of which you should be aware.

If no messages are returned, no inconsistencies are found in the emulator configuration.

Examples

To check for inconsistencies between the configuration and the EMSIM registers, choose **Display→Configuration Info→Diagnostics** from either the configuration interface the emulator/analyzer interface pull-down menu.


Hewlett Packard Emulator/Analyzer: hplsds2 (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x( )
( ): Reserved
Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
Processor Info: Diagnostics
________________________________________________________________________
The emulator configuration has been checked for inconsistencies and potential
problems.  Any "ERRORS" listed should be resolved for the emulator to operate
correctly.  Any "Warning" messages list expectations and limitations of the
emulator of which the user should be made aware.
,
  scr      = 04000f00  emscr      = 00000f00
No inconsistencies or potential problems found.
```

```
STATUS:   M68302--Running in monitor      Software break: 0002bce@sp
```

```
display configuration_info
```

6

Using the Emulator

Using the Emulator

This chapter describes general tasks you may wish to perform while using the emulator. These tasks are grouped into the following sections:

- Using the emulation copy of the SIM (emsim) registers.
- Loading and storing absolute files.
- Using symbols.
- Using context commands.
- Executing user programs (starting, stopping, stepping, and resetting the emulator).
- Using execution breakpoints.
- Displaying and modifying registers.
- Displaying and modifying memory.
- Displaying data values.
- Changing the interface settings.
- Using system commands.
- Using Simulated I/O.
- Using Basis Branch Analysis.

Using the EMSIM Registers

The 6830x processors contain a System Integration Block (SIB) which integrates various peripherals with with the M68000 core. Programming certain parts of the SIB will affect operation of the emulator. These emulator-sensitive parts of the SIB are referred to as the processor's System Integration Module (SIM).

The SIM contains the BAR and SCR registers along with the chip-select, port control and interrupt control registers, plus any clock control registers if present. For the 68302 emulator (64798C) the SIM registers are: BAR, SCR, BR0, OR0, BR1, OR1, BR2, OR2, BR3, OR3, PACNT, PADDR, PBCNT, PBADDR, and GIMR.

The Programming of the SIM registers by the user affects how the emulator must be configured to operate properly. For example, the GIMR determines how interrupt level 7 is detected by the processor. The emulator uses interrupt level 7 to break to the monitor, thus must be configured according to the expectations of the processor. The chip select registers determine the DTACK source for a bus cycle within the range of a chip select. This information is needed for the emulator to properly complete bus cycles.

The EMSIM registers, which are an emulator version of the SIM registers, are used to configure the emulator hardware. The EMSIM registers are usually set to the "after initialization code" values desired for the SIM registers. By default the EMSIM registers contain the "processor reset" SIM values (refer to the appropriate *Motorola MC6830x User's Manual* for specific values.) Therefore, the default programming of the emulator hardware will match the SIM reset values.

If desired, the programming of the emulator hardware (EMSIM registers) can be transferred into the processor SIM registers with the **Modify→SIM Registers →Copy Emulator SIM to Processor SIM** pull-down menu or ***sync_sim_registers to_6830x_from_config*** from the command line. This happens automatically each time a break to the monitor from emulation reset occurs. This ensures that the processor is prepared to properly access memory when a program is downloaded to the emulator.

Alternatively, the emulator hardware can be programmed from the processor's SIM registers with the **Modify→SIM Registers →Copy Processor SIM to Emulator SIM** pull-down menu or ***sync_sim_registers from_6830x_to_config*** from the command line. This is useful if initialization code that configures the processor SIM exists, but you don't know what its values are. In this case, you can use the default configuration, run from reset to execute the initialization code, and use this command to configure the emulator to match the processor SIM.

At any time, you can verify if the SIM and emulator hardware (EMSIM) are programmed the same with the **Display→SIM Register Differences** pull-down menu or ***sync_sim_registers difference*** from the command line. Any differences between the two register sets will be listed.

It should be noted that the emulator hardware is programmed solely from the EMSIM register set and is therefore static with respect to the application program. No attempt is made to update the programming of the emulator hardware by tracking instructions that will program the processor SIM.

This section shows you how to:

- View the SIM register differences.
- Synchronize to the 6830x SIM registers.
- Synchronize to the EMSIM registers.
- Restore default values in the EMSIM registers.

To view the SIM register differences

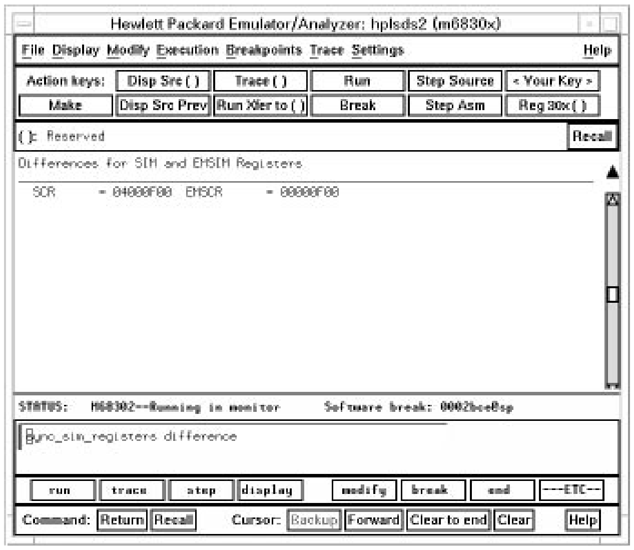
- Choose **Display→SIM Register Differences** from the emulator/analyzer interface pull-down menu.

Or, using the command line, enter:

```
sync_sim_registers difference
```

Examples

To display the SIM register differences:


Hewlett Packard Emulator/Analyzer: hplsds2 (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x( )
( ): Reserved
Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
Differences for SIM and EMSIM Registers

  SCR      = 04000F00  EMSCR      = 00000F00

STATUS:   M68302--Running in monitor      Software break: 0002bce@sp

sync_sim_registers difference
```

To synchronize to the 6830x SIM registers

- Choose **Modify→SIM Registers→Copy Processor SIM to Emulator SIM** from the emulator/analyzer pull-down menu.

Or, using the command line, enter:

```
sync_sim_registers from_6830x_to_config
```

The contents of the 6830x SIM registers are copied to the emulation copy of the SIM registers.

To synchronize to the EMSIM registers

- Choose **Modify→SIM Registers→Copy Emulator SIM to Processor SIM** from the emulator/analyzer pull-down menu.

  Or, using the command line, enter:

```
sync_sim_registers to_6830x_from_config
```

The contents of the emulation copy of the SIM registers are copied to the 6830x SIM registers.

To restore default values in the EMSIM registers

- Choose **Modify→SIM Registers→Default Emulator SIM** from the emulator/analyzer interface pull-down menu.

  Or, using the command line, enter:

```
sync_sim_registers default_config
```

The contents of the EMSIM register set are restored to their power-up values.

Loading and Storing Absolute Files

This section describes the tasks related to loading absolute files into the emulator and storing memory contents into absolute files. This section shows you how to:

- Load absolute files into memory.
- Load absolute files without symbols.
- Store memory contents into absolute files.

To load absolute files

- **Choose File→Load→Executable and use the dialog box to select the absolute file.**

Or, using the command line, enter:

```
load <absolute_file>
```

You can load absolute files into emulation or target system memory. You can load IEEE-695 format absolute files. You can also load HP format absolute files. The store memory command creates HP format absolute files.

If you wish to load only that portion of the absolute file that resides in memory mapped as emulation RAM or ROM, use the command line's ***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 command line's ***load user_mem*** syntax.

If you want both emulation and target memory to be loaded, do not specify emul_mem or user_mem.

Examples

To load the demo program absolute file and the configuration file, enter the following commands:

```
load configuration Config.EA
load ecs.x
```

To load only portions of the absolute file that reside in target system RAM:

```
load user_mem absfile
```

To load only portions of the absolute file that reside in emulation memory:

```
load emul_mem absfile
```

To load absolute files without symbols

- Choose **File→Load→Program Only** and use the dialog box to select the absolute file.

Or, using the command line, enter:

```
load <absolute_file> nosymbols
```

To store memory contents into absolute files

- Using the command line, enter:

```
store memory <expression>
```

You can store emulation or target system memory contents into HP format absolute files on the host computer. Absolute files are stored in the current directory. If no extension is given for the absolute file name, it is given a ".X" extension.

Examples

To store the contents of memory locations 900H through 9FFH to an absolute file on the host computer named "absfile":

```
store memory 900h thru 9ffh to absfile
```

After the command above, a file named "absfile.X" exists in the current directory on the host computer.

Using Symbols

If symbol information is present in the absolute file, it is loaded along with the absolute file (unless you use the nosymbols option). Both global symbols and local program module symbols can be displayed.

Long symbol names can be truncated in the symbols display; however, you can increase the width of the symbols display by starting the interface with more columns (refer to the "Setting X Resources" chapter). This section describes how to:

- Load symbols.
- Display global and local symbols.
- Display a symbol's parent symbol.
- Copy and paste a full symbol name to the entry buffer.

To load symbols

- **Choose File→Load→Symbols Only and use the dialog box to select the absolute file.**

Or, using the command line, enter:

```
load symbols <absolute_file>
```

Symbols are loaded automatically unless you use the nosymbols option when loading absolute files. If you did use the nosymbols option when loading the absolute file, you can load the symbols without loading the absolute file again.

Examples

To load symbols from the demo program:

```
load symbols ecs.x
```

To display global symbols

- Choose **Display→Global Symbols**.

Or, using the command line, enter:

```
display global_symbols
```

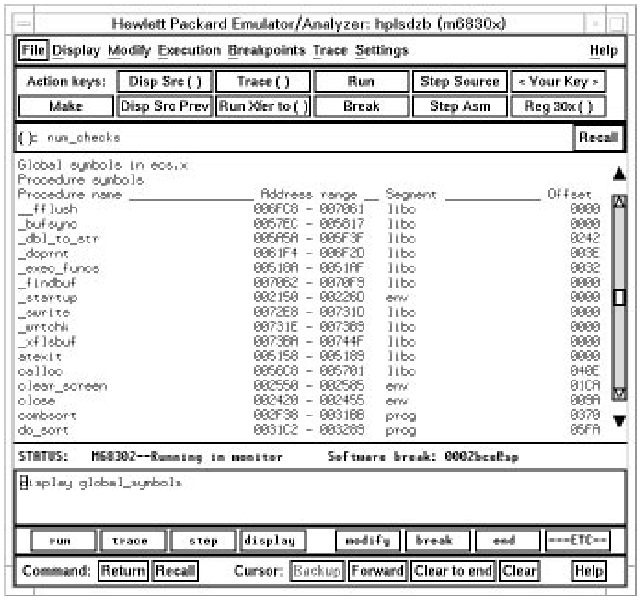
Listed are: address ranges associated with a symbol, the segment the symbol is associated with, and the offset of that symbol within the segment.

If there is more than a screen full of information, you can use the up arrow, down arrow, <NEXT>, or <PREV> keys to scroll the information up or down on the display.

Examples

To display global symbols in the demo program:

display global_symbols


Hewlett Packard Emulator/Analyzer: hplsdzb (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x( )
( ): num_checks
Recall

```
Global symbols in ecs.x
Procedure symbols
Procedure name ________________ Address range __ Segment ____________ Offset
__fflush                        006FC8 - 007061   libc               0000
_bufsync                        0057EC - 005817   libc               0000
_dbl_to_str                     005A5A - 005F3F   libc               0242
_doprnt                         0061F4 - 006F2D   libc               003E
_exec_funcs                     00510A - 0051AF   libc               0032
_findbuf                        007062 - 0070F9   libc               0000
_startup                        002150 - 00226D   env                0000
_swrite                         0072E8 - 00731D   libc               0000
_wrtchk                         00731E - 007389   libc               0000
_xflsbuf                        00738A - 00744F   libc               0000
atexit                          005158 - 005189   libc               0000
calloc                          0056C8 - 005701   libc               040E
clear_screen                    002550 - 002585   env                01CA
close                           002420 - 002455   env                009A
combsort                        002F38 - 0031BB   prog               0370
do_sort                         0031C2 - 003289   prog               05FA

STATUS:    M68302--Running in monitor      Software break: 0002bca@sp

display global_symbols
```

run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

To display local symbols

- When displaying symbols, position the mouse pointer over a symbol on the symbol display screen and click the *select* mouse button.
- When displaying symbols, position the mouse pointer over the symbol, press and hold the *select* mouse button, and choose **Display Local Symbols** from the pop-up menu.
- Position the mouse cursor in the entry buffer and enter the module whose local symbols are to be displayed; then, choose **Display→Local Symbols ()**.

Or, using the command line, enter:

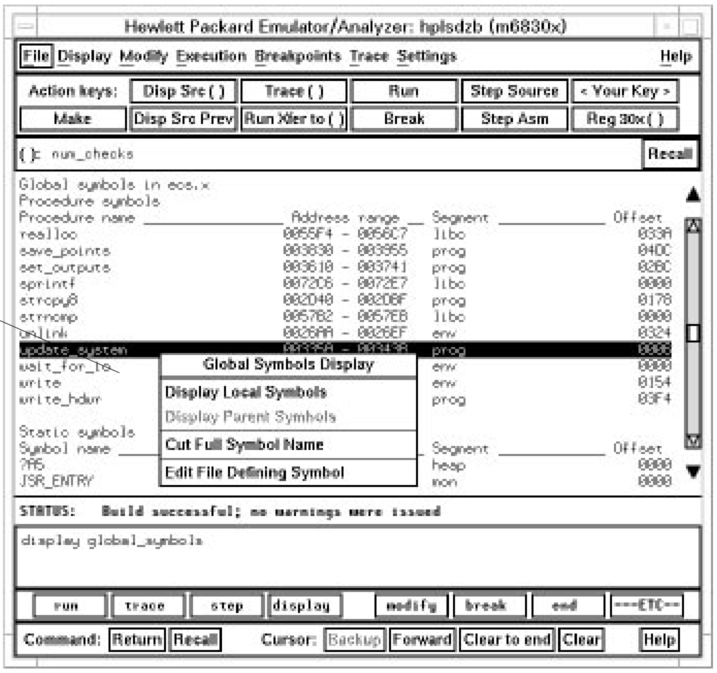
```
display local_symbols_in <module>
```

To display the address ranges associated with the high-level program's source file line numbers, you must display the local symbols in the file.

Examples

To use the Symbols Display pop-up menu:

View the local symbols associated with the highlighted symbol by choosing this menu item.

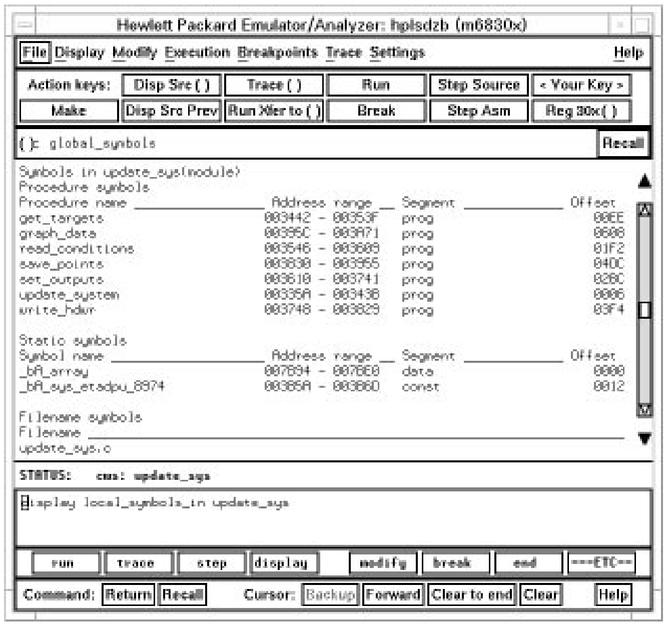

Hewlett Packard Emulator/Analyzer: hplsdzb (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x( )
Recall
Global Symbols Display
Display Local Symbols
Display Parent Symbols
Cut Full Symbol Name
Edit File Defining Symbol
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
( ): num_checks

Global symbols in ecs.x
Procedure symbols
Procedure name ________________ Address range __ Segment ____________ Offset
realloc                         0055F4 - 0056C7   libc               033A
save_points                     003830 - 003955   prog               040C
set_outputs                     003610 - 003741   prog               02BC
sprintf                         0072C6 - 0072E7   libc               0000
strcpy8                         002D40 - 002DBF   prog               0178
strncmp                         0057B2 - 0057EB   libc               0000
unlink                          0026AA - 0026EF   env                0324
update_system                   00339A - 00343A   prog               0000
wait_for_io                                       env                0000
write                                             env                0154
write_hdwr                                        prog               03F4

Static symbols
Symbol name ___                                 _ Segment ____________ Offset
?A6                                               heap               0000
JSR_ENTRY                                         mon                0000

STATUS:   Build successful; no warnings were issued

display global_symbols
```

Using the command line:

To display local symbols in a module:

```
display local_symbols_in update_sys
```


Hewlett Packard Emulator/Analyzer: hplsdzb (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x( )
( ): global_symbols
Recall

```
Symbols in update_sys(module)
Procedure symbols
Procedure name _______________ Address range __ Segment ___________ Offset
get_targets                    003442 - 00353F   prog                  00EE
graph_data                     00395C - 003A71   prog                  0608
read_conditions                003546 - 003609   prog                  01F2
save_points                    003830 - 003955   prog                  04DC
set_outputs                    003610 - 003741   prog                  02BC
update_system                  00335A - 003438   prog                  0006
write_hdwr                     003748 - 003829   prog                  03F4

Static symbols
Symbol name __________________ Address range __ Segment ___________ Offset
_bA_array                      007B94 - 007BE0   data                  0000
_bA_sys_etadpu_8974            00385A - 00386D   const                 0012

Filename symbols
Filename ____________________________________________________________
update_sys.c
```

```
STATUS:    cws: update_sys
```

```
display local_symbols_in update_sys
```

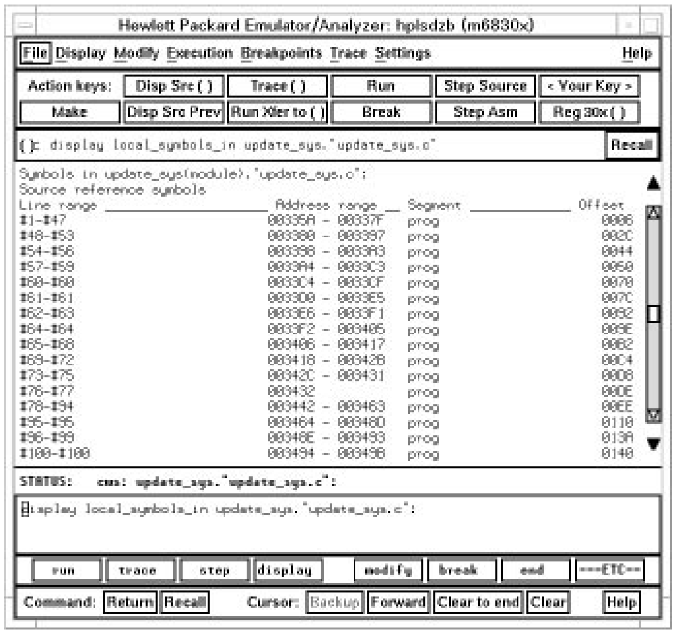
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

To display local symbols in a procedure:

```
display local_symbols_in update_sys.save_points
```

To display address ranges associated with the high-level source line numbers:

`display local_symbols_in`
`update_sys."update_sys.c":`


Hewlett Packard Emulator/Analyzer: hplsdzb (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x( )
Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

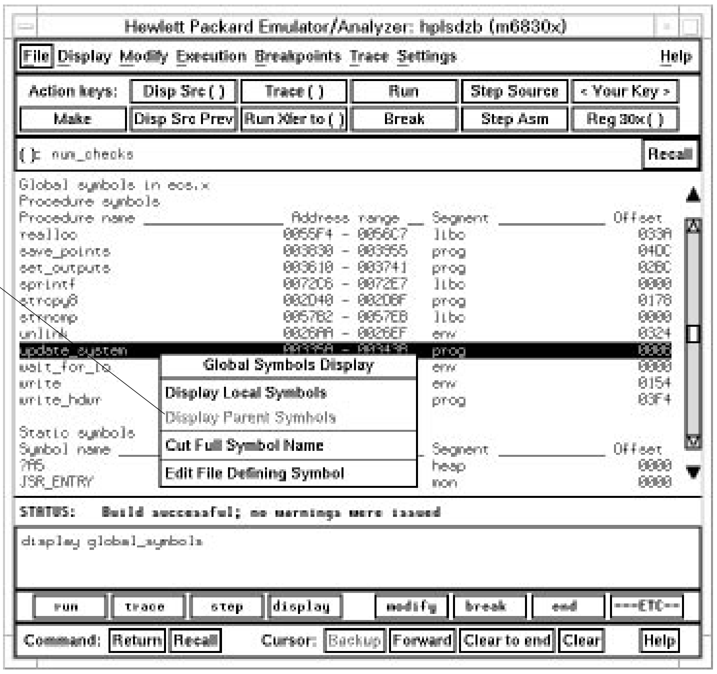
```
( ): display local_symbols_in update_sys."update_sys.c"

Symbols in update_sys(module)."update_sys.c":
Source reference symbols
Line range ____________________ Address range __ Segment ____________ Offset
#1-#47                          00335A - 00337F   prog               0006
#48-#53                         003380 - 003397   prog               002C
#54-#56                         003398 - 0033A3   prog               0044
#57-#59                         0033A4 - 0033C3   prog               0050
#60-#60                         0033C4 - 0033CF   prog               0070
#61-#61                         0033D0 - 0033E5   prog               007C
#62-#63                         0033E6 - 0033F1   prog               0092
#64-#64                         0033F2 - 003405   prog               009E
#65-#68                         003406 - 003417   prog               00B2
#69-#72                         003418 - 00342B   prog               00C4
#73-#75                         00342C - 003431   prog               00D8
#76-#77                         003432            prog               00DE
#78-#94                         003442 - 003463   prog               00EE
#95-#95                         003464 - 00348D   prog               0110
#96-#99                         00348E - 003493   prog               013A
#100-#100                       003494 - 00349B   prog               0140

STATUS:    cws: update_sys."update_sys.c":

display local_symbols_in update_sys."update_sys.c":
```

To display a symbol's parent symbol

- When displaying symbols, position the mouse pointer over the symbol, press and hold the *select* mouse button, and choose **Display Parent Symbols** from the pop-up menu.

Examples

View the parent symbol associated with the highlighted symbol by choosing this menu item.


Hewlett Packard Emulator/Analyzer: hplsdzb (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x( )
( ): num_checks
Recall
Global Symbols Display
Display Local Symbols
Display Parent Symbols
Cut Full Symbol Name
Edit File Defining Symbol
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
Global symbols in ecs.x
Procedure symbols
Procedure name _______________ Address range __ Segment ____________ Offset
realloc                         0055F4 - 0056C7   libc                 033A
save_points                     003830 - 003955   prog                 04DC
set_outputs                     003610 - 003741   prog                 02BC
sprintf                         0072C6 - 0072E7   libc                 0000
strcpy8                         002D40 - 002D8F   prog                 0178
strncmp                         0057B2 - 0057EB   libc                 0000
unlink                          0026AA - 0026EF   env                  0324
update_system                   00335A - 00343A   prog                 0000
wait_for_io                                       env                  0000
write                                             env                  0154
write_hdwr                                        prog                 03F4

Static symbols
Symbol name ____                                  Segment ____________ Offset
?A5                                               heap                 0000
JSR_ENTRY                                         mon                  0000

STATUS:    Build successful; no warnings were issued

display global_symbols
```

To copy and paste a full symbol name to the entry buffer

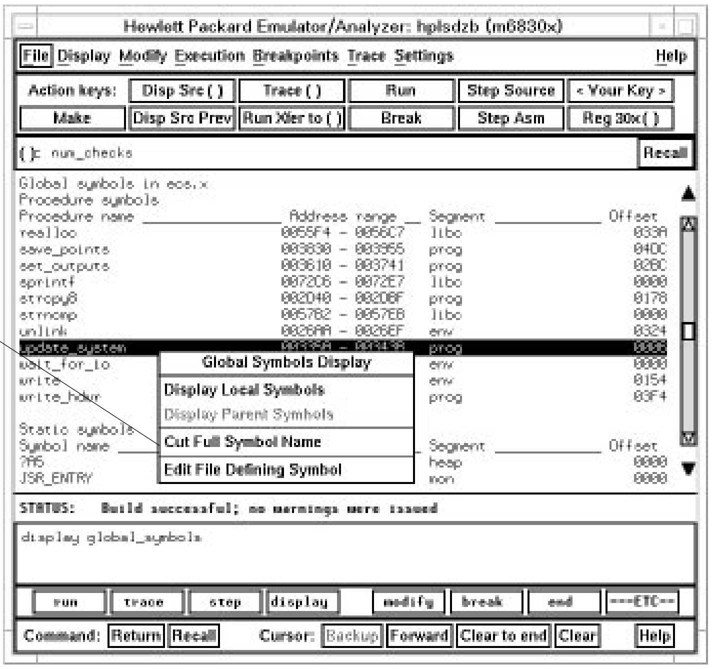
- When displaying symbols, position the mouse pointer over the symbol, press and hold the *select* mouse button, and choose **Cut Full Symbol Name** from the pop-up menu.

Once the full symbol name is in the entry buffer, you can use it with pull-down menu items or paste it to the command line area.

By cutting the full symbol name, you get the complete names of symbols that have been truncated. Also, you are guaranteed of specifying the proper scope of the symbol.

Examples

Copy the full name of the highlighted symbol to the entry buffer by choosing this menu item.



```
( ): num_checks

Global symbols in ecs.x
Procedure symbols
Procedure name _______________ Address range __ Segment ____________ Offset
realloc                         0055F4 - 0056C7   libc                 033A
save_points                     003830 - 003955   prog                 04DC
set_outputs                     003610 - 003741   prog                 02BC
sprintf                         0072C6 - 0072E7   libc                 0000
strcpy8                         002D40 - 002D8F   prog                 0178
strncmp                         005782 - 0057EB   libc                 0000
unlink                          0026AA - 0026EF   env                  0324
update_system                   [illegible]       prog                 0000
wait_for_io                                       env                  0000
write                                             env                  0154
write_hdwr                                        prog                 03F4

Static symbols
Symbol name ____                                  Segment ____________ Offset
?A5                                               heap                 0000
JSR_ENTRY                                         mon                  0000

STATUS:    Build successful; no warnings were issued

display global_symbols
```

Using Context Commands

The commands in this section display and control the directory and symbol contexts for the interface.

Directory context.

The current directory context is the directory accessed by all system references for files—primarily load, store, and copy commands—if no explicit directory is mentioned. Unless you have changed directories since beginning the emulation session, the current directory context is that of the directory from which you started the interface.

Symbol context.

The emulator/analyzer interface and the Symbol Retrieval Utilities (SRU) together support a current working symbol context. The current working symbol represents an enclosing scope for local symbols. If symbols have not been loaded into the interface, you cannot display or change the symbol context.

This section shows you how to:

- Display the current directory and symbol context.
- Change the directory context.
- Change the current working symbol context.

To display the current directory and symbol context

- Choose **Display→Context**.

Or, using the command line, enter:

```
pwd and pws
```

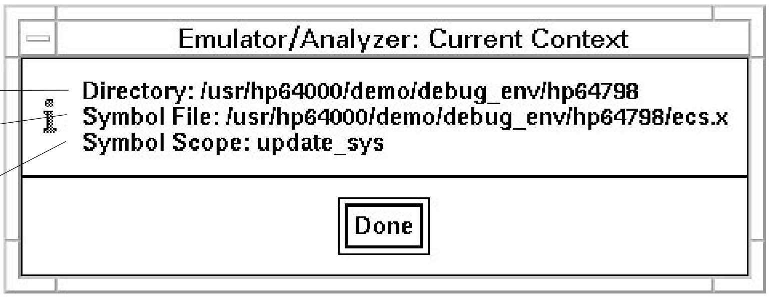
The current directory and working symbol contexts are displayed, and also the name of the last executable file from which symbols were loaded.

Example

Directory context.

Executable from which symbols were last loaded.

Symbol context.



Directory: /usr/hp64000/demo/debug_env/hp64798
Symbol File: /usr/hp64000/demo/debug_env/hp64798/ecs.x
Symbol Scope: update_sys

To change the directory context

- Choose **File→Context→Directory** and use the dialog box to select a new directory.

Or, using the command line, enter:

```
cd <directory>
```

The Directory Selection dialog box contains a list of directories accessed during the emulation session as well as any predefined directories present at interface startup.

You can predefine directories and set the maximum number of entries for the Directory Selection dialog box by setting X resources (see the "Setting X Resources" chapter).

To change the current working symbol context

- Choose **File→Context→Symbols** and use the dialog box to select the new working symbol context.

Or, using the command line, enter:

```
cws <symbol_context>
```

(Because cws is a hidden command and doesn't appear on a softkey label, you have to type it in.)

You can predefine symbol contexts and set the maximum number of entries for the Symbol Scope Selection dialog box by setting X resources (see the "Setting X Resources" chapter).

Displaying local symbols or displaying memory in mnemonic format causes the working symbol context to change as well. The new context will be that of the local symbols or memory locations displayed.

Executing User Programs

You can use the emulator to run programs, break program execution into the monitor, step through the program by high-level source lines or by assembly language instructions, and reset the emulation processor.

When displaying memory in mnemonic format, a highlighted bar shows the current program counter address. When you step, the mnemonic memory display is updated to highlight the new program counter address.

When displaying resisters, the register display is updated to show you the contents of the registers after each step.

You can open multiple interface windows to display memory in mnemonic format and registers at the same time. Both windows are updated after stepping.

Note:
A stack pointer must be initialized prior to execution. If you have set the Reset Vectors option of the Emulation Configuration options to Auto Reset, the stack pointer is automatically reset. If you have not set Reset Vectors to Auto, you must manually reset the pointers through the Emulation Configuration menu prior to each execution.

This section describes how to:

- Start the emulator running the user program.
- Stop (break from) user program execution.
- Step through user programs.
- Reset the emulation processor.

To run programs from the current PC

- Choose **Execution→Run→from PC**.

  Or, using the command line, enter:

```
run
```

When the emulator is executing the user program, the message "Running user program" is displayed on the status line.

To run programs from an address

- Position the mouse pointer in the entry buffer and enter the address you want to run from; then, choose **Execution→Run→from ()**.

  Or, using the command line, enter:

```
run from <address>
```

Examples

To run from address 920H:

```
run from 920h
```

To run programs from the transfer address

- Choose **Execution→Run→from Transfer Address**.

  Or, using the command line, enter:

```
run from transfer_address
```

Most software development tools allow you to specify a starting or entry address for program execution. That address is included with the absolute file's symbolic information and is known by the interface as the *transfer address*.

To run programs from reset

- Choose **Execution→Run→from Reset**.

  Or, using the command line, enter:

```
run from reset
```

The ***run from reset*** command specifies a run from target system reset. It is equivalent to entering a ***reset*** command followed by a ***run*** command. The processor will be reset and then allowed to run.

To run programs until an address

- When displaying memory in mnemonic format, position the mouse pointer over the line that you want to run until; then press and hold the *select* mouse button and choose **Run Until** from the pop-up menu.
- Position the mouse pointer in the entry buffer and enter the address you want to run from; then, choose **Execution→Run→until ()**.

Or, using the command line, enter:

```
run until <address>
```

When you run until an address, a software breakpoint is set at the address and the program is run from the current program counter.

When using the command line, you can combine the various types of run commands; for example, you can run from the transfer address until another address.

Examples

To run from the transfer address until the address of the global symbol main:

```
run from transfer_address until main
```

To stop (break from) user program execution

- Choose **Execution→Break**.

Or, using the command line, enter:

```
break
```

This command generates a break to the monitor.

Software breakpoints and the run until command allow you to stop execution at particular points in the user program.

Examples

To break emulator execution from the user program to the monitor:

```
break
```

To step high-level source lines

- Choose **Execution→Step Source** and select one of the items from the cascade menu.

Or, using the command line, enter:

```
step source
```

When stepping through instructions associated with source lines, execution can remain in a loop and the message "Stepping source line 1; Next PC: <address>" is displayed on the status line. In this situation you can abort the step command by pressing <CTRL>c.

Examples

To step through instructions associated with the high-level source lines at the current program counter:

```
step source
```

To step through instructions associated with high-level source lines at address "main":

```
step source from main
```

To step assembly-level instructions

- **Choose Execution→Step Instruction and select one of the items from the cascade menu.**

Or, using the command line, enter:

```
step
```

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.

Examples

To step one instruction from the current program counter:

```
step
```

To step a number of instructions from the current program counter:

```
step 8
```

To step a number of instructions from a specified address:

```
step 16 from 920h
```

To reset the emulation processor

- Choose **Execution→Reset**.

Or, using the command line, enter:

```
reset
```

The reset command causes the processor to be held in a reset state until a break, run, or step command is entered. A CMB execute signal will also cause the emulator to run if reset.

Using Execution Breakpoints

Breakpoints allow you to stop target program execution at a particular address and transfer control to the emulation monitor. Suppose your system crashes when it executes in a certain area of your program. You can set a breakpoint in your program at a location just before the crash occurs. When the processor executes the breakpoint, the emulator will force a break to the monitor. You can display registers or memory to understand the state of the system before the crash occurs. Then you can step through the program instructions and examine changes in the system registers that lead up to the system crash.

Software breakpoints are implemented in the 6830x emulator by replacing opcodes with TRAP instructions. You can configure the emulator to use one of the 16 different TRAP instructions for software breakpoints. The default emulator configuration specifies that the TRAP #0FH is used for software breakpoints.

In order to successfully set a software breakpoint, the emulator must be able to write to the memory location specified Therefore, software breakpoints cannot be set in target memory while the emulator is reset, and they can never be set in target ROM. (You can, however, copy target ROM to emulation memory by storing the contents of target ROM to an absolute file, re-mapping the range as emualtion RAM, and loading the absolute file.)

When you set a software breakpoint, the emulator replaces the opcode at the address specified with the TRAP instruction. When the emulator detects a read from the appropriate vector table location (TRAP instruction has executed in the user program), execution breaks to the monitor.

If the TRAP was generated by a software breakpoint, a message containing the address of the breakpoint is displayed on the status line, and, if the breakpoint is temporary, the original opcode is resotroed in the user program. If the breakpoint is permanent, it remains active. A subsequent run or step command will execute from the breakpoint address.

If the TRAP was not inserted as the result of a modify software_breakpoints set command (in other words, it is part of the user program), the "Undefined software breakpoint" message is displayed on the status line. To continue with the program execution, you must run or step from the user program's TRAP instruction vector address.

This section shows you how to:

- Set execution breakpoints in RAM.
- Set execution breakpoints in ROM.
- Use temporary and permanent breakpoints.
- Enable and disable execution breakpoints.
- Set a permanent breakpoint.
- Set a ROM breakpoint in RAM.
- Clear execution breakpoints.
- Display status of all execution breakpoints.

CAUTION

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 eecuting code in the area where the breakpoint is being modified, program execution may be unreliable.

To enable execution breakpoints

- Choose **Breakpoints→Enable.**
- Inside the breakpoints list display, press and hold the *select* mouse button and then choose **Enable/Disable Software Breakpoints** from the pop-up menu.

Or, using the command line, enter:

```
modify software_breakpoints enable
```

You must enable breakpoints before you can set, inactivate, or clear any breakpoints.

Once you have enabled breakpoints, you can enter new ones into the breakpoint table. Note that if you enable breakpoints, add several, and then disable them, they all become inactive. If you reenable the breakpoints feature, you must choose **Breakpoints→Set All**, or on the command-line, enter ***modify software_breakpoints*** set if you want to set all the existing breakpoint entries.

To disable an execution breakpoint

- Choose **Breakpoints→Enable** again. The **Breakpoints→Enable** selection is a switch.
- Inside the breakpoints list display, press and hold the *select* mouse button and then choose **Enable/Disable Software Breakpoints** from the pop-up menu.

Or using the command line, enter:

```
modify software_breakpoints disable
```

Sometimes you will want to temporarily disable the execution breakpoints feature without removing the existing breakpoints. Use one of the above commands to do this.

When you disable breakpoints, the emulator replaces the BKPT instructions at all breakpoint locations with the original instructions. It marks the breakpoint table entries as “inactive.” The processor won’t break to monitor when the instructions at inactive locations are executed.

If you later enable breakpoints, the ones in the table are still inactive. To use them, you must set them by choosing **Breakpoints→Set All**, or on the command line, entering the ***modify software_breakpoints set*** command.

To set a permanent breakpoint

- When displaying memory in mnemonic format, position the mouse pointer over the program line where you wish to set the breakpoint and click the *select* mouse button. Or, press and hold the *select* mouse button and choose **Set/Clear Software Breakpoint** from the pop-up menu.
- Place an absolute or symbolic address in the entry buffer; then, choose **Breakpoints→Permanent()**.

Or, using the command line, enter:

```
modify software_breakpoints set <address>
permanent
```

The breakpoints feature must be enabled before individual breakpoints can be set.

When displaying memory in mnemonic format, asterisks (*) appear next to breakpoint addresses. An asterisk shows the breakpoint is active. Also, if assembly level code is being displayed, the disassembled instruction mnemonic at the breakpoint address will show the breakpoint instruction.

To set a temporary breakpoint

- Type in the absolute or symbolic address of the breakpoint you want to set in the entry buffer. Then choose **Breakpoints→Temporary( )**, (or choose **Breakpoints→Set( )** if your version of HP 64700 system firmware is less than A.04.00).
- Choose **Breakpoints→Set All** to set all existing breakpoints in the breakpoint table.
- Inside the breakpoints list display, press and hold the *select* mouse button and then choose **Set All Breakpoints** from the pop-up menu.

Or, using the command line, enter commands as follows:

- To set a breakpoint at a location given by <address>, enter:

```
modify software_breakpoints set <address>
```

- To set all existing breakpoints in the breakpoint table, enter:

```
modify software_breakpoints set
```

To add a new breakpoint, you can choose **Breakpoints→Temporary( )** with the name of the new breakpoint in the entry buffer, or use the ***modify software_breakpoints set*** command and specify the address for the breakpoint. You can also use this method to reenable an existing breakpoint at that address.

If you choose **Breakpoints→Set All**, or use the ***modify software_breakpoints set*** command without an address parameter, all existing breakpoints in the breakpoints table will be enabled. The breakpoints feature must be enabled before individual breakpoints can be set.

When displaying memory in mnemonic format, asterisks (*) appear next to breakpoint addresses. An asterisk shows the breakpoint is active. Also, if assembly level code is being displayed, the disassembled instruction mnemonic at the breakpoint address will show the breakpoint instruction.

Examples

Set a new breakpoint at get_targets:

```
modify software_breakpoints set
update_sys.get_targets
```

Reenable all existing breakpoints:

```
modify software_breakpoints set
```

To clear an execution breakpoint

- Type in the name of the breakpoint you want to clear in the entry buffer. Then choose **Breakpoints→Clear().**
- Choose **Breakpoints→Clear All** to clear all existing breakpoints in the breakpoint table.
- Inside the breakpoints list display, press and hold the *select* mouse button and then choose **Clear (delete) Breakpoint** from the pop-up menu to clear the selected breakpoint.

Or, using the command line, enter commands as follows:

- To remove an existing breakpoint at a location given by <address>, enter:

```
modify software_breakpoints clear <address>
```

- To remove all existing breakpoints, enter:

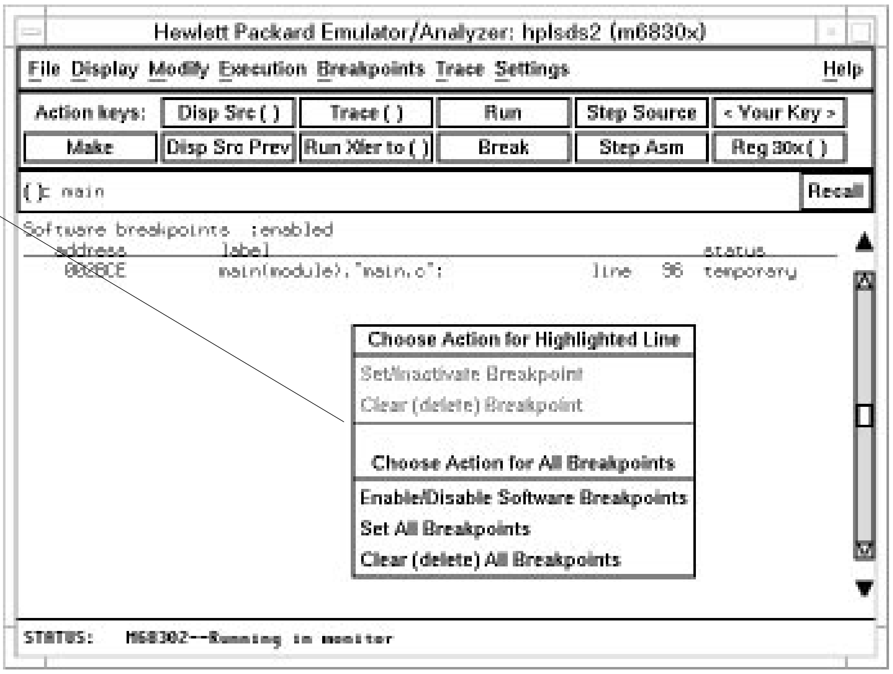
```
modify software_breakpoints clear
```

When you're finished using a particular breakpoint, you should clear the breakpoint table entry. The original instruction is restored to memory, and the breakpoint table entry is removed.

Examples

To clear a breakpoint using the breakpoints display pop-up menu:

Bring up the menu and choose this item to clear the highlighted breakpoint.


Hewlett Packard Emulator/Analyzer: hplsds2 (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x ( )
Recall
Choose Action for Highlighted Line
Set/Inactivate Breakpoint
Clear (delete) Breakpoint
Choose Action for All Breakpoints
Enable/Disable Software Breakpoints
Set All Breakpoints
Clear (delete) All Breakpoints

```
( ): main
Software breakpoints  :enabled
  address        label                              status
    0028CE        main(module),"main.c":   line  96  temporary
STATUS:   M68302--Running in monitor
```

To clear an existing breakpoint at get_targets:

`modify software_breakpoints clear` `update_sys.get_targets`

To clear all existing breakpoints:

`modify software_breakpoints clear`

To clear all execution breakpoints

- When displaying breakpoints, position the mouse pointer within the breakpoints display screen, press and hold the *select* mouse button, and choose **Clear (delete) All Breakpoints** from the pop-up menu.
- Choose **Breakpoints→Clear All**.

Or, using the command line, enter:

```
modify software_breakpoints clear
```

To display the status of all execution breakpoints

- Choose **Breakpoints→Display** or **Display→Breakpoints.**

Or, using the command line, enter:

```
display software_breakpoints
```

The breakpoints table shows you whether the breakpoints feature is currently enabled or disabled. Also, the status is shown for each breakpoint in memory. If “Pending,” the BKPT instruction is in memory at that location and the breakpoint is set. If “Inactive,” the memory location contains the original instruction, and the breakpoint will not be executed.

Active breakpoints are indicated in the memory mnemonic display by asterisks beside the lines with breakpoints set.

The status of a breakpoint can be one of the following:

temporary	This means the temporary breakpoint has been set but not encountered during program execution. These breakpoints are removed when the breakpoint is encountered.
permanent	This means the permanent breakpoint is active. Permanent breakpoints remain active after they are encountered during execution.
inactivated	This means the breakpoint has been inactivated. Pending breakpoints are inactivated when they are encountered during program execution. Both temporary and permanent breakpoints can be inactivated (and restored) using the breakpoints display pop-up menu.
pending	This means the temporary breakpoint has been set but not encountered during program execution. When encountered, these breakpoints are inactivated, but retained in the breakpoints list. Pending breakpoints can only be set using the softkey command line with commands like ***modify software_breakpoints set 1000*** and not selecting the additional options <temporary> or <permanent>. The "pending" breakpoints status is retained for compatibility with older product software versions.

In the breakpoints display, a pop-up menu is available, obtained by pressing the *select* mouse button. You can inactivate or restore the status of any breakpoint in the breakpoints list, as well as enable or disable the breakpoints feature, using the pop-up menu.

Displaying and Modifying Registers

This section describes tasks related to displaying and modifying emulation processor registers.

Within the interface, related registers are grouped into a class. For example, the <BASIC> register class includes general registers, such as the PC, ST, USP, SSP, and data and address registers. You can display the contents of an individual register, a register class, or all registers.

This section shows you how to:

- Display register contents.
- Modify register contents.

To display register contents

- Choose **Display→Registers→<register class>**.

Or, using the command line, enter:

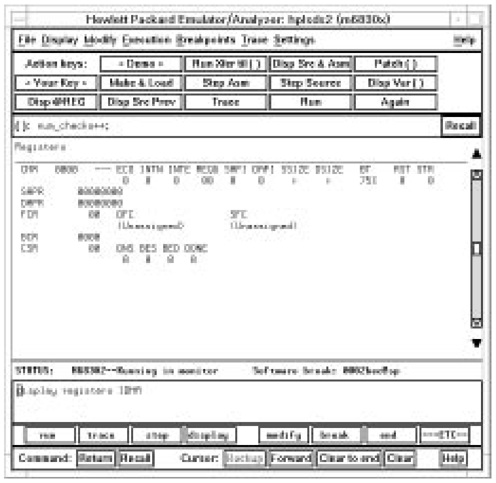
```
display registers <register class>
```

When displaying registers, you can display classes of registers and individual registers.

Examples

To display the IDMA register class:

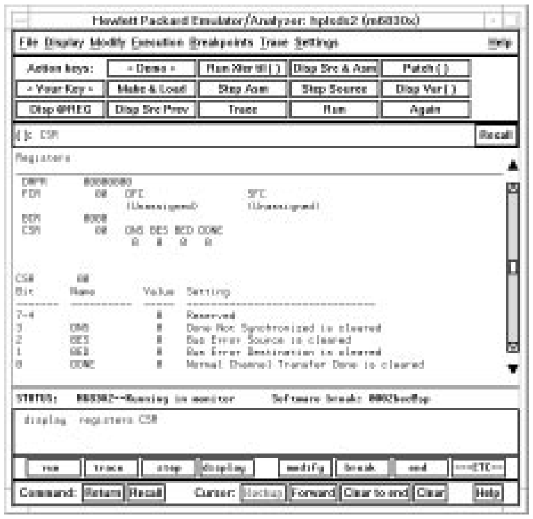
`display registers class IDMA`


Hewlett Packard Emulator/Analyzer: hplsds2 (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: « Demo » Run Xfer til ( ) Disp Src & Asm Patch ( )
« Your Key » Make & Load Step Asm Step Source Disp Var ( )
Disp @REG Disp Src Prev Trace Run Again
Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
( ): num_checks++;

Registers
 CMR   0000   --- ECO INTN INTE REQG SAPI DAPI SSIZE DSIZE  BT    RST STR
                   0   0    0    00   0    0    ?     ?     75%   0   0
 SAPR     00000000
 DAPR     00000000
 FCR        00   DFC               SFC
                 (Unassigned)      (Unassigned)
 BCR      0000
 CSR        00   DNS BES BED DONE
                  0   0   0   0

STATUS:   M68302--Running in monitor      Software break: 0002bc0@p

display registers IDMA
```

To display the individual register CSR:

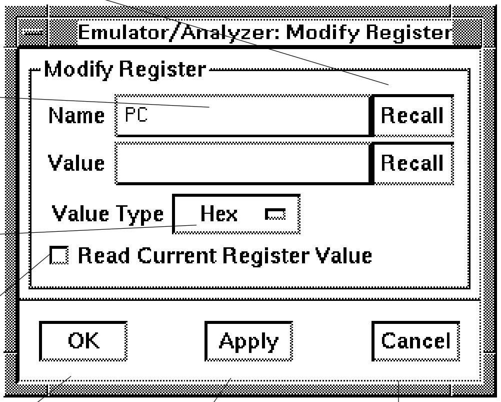
`display registers individual CMR`


Hewlett Packard Emulator/Analyzer: hplsds2 (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: « Demo » Run Xfer til ( ) Disp Src & Asm Patch ( )
« Your Key » Make & Load Step Asm Step Source Disp Var ( )
Disp @REG Disp Src Prev Trace Run Again
Recall
run trace step display modify break end ---ETC--
Command: Return Recall Cursor: Backup Forward Clear to end Clear Help

```
( ): CSR

Registers
 DAPR     00000000
 FCR        00   DFC               SFC
                 (Unassigned)      (Unassigned)
 BCR      0000
 CSR        00   DNS BES BED DONE
                  0   0   0   0


CSR      00
Bit      Name        Value  Setting
-------  ----------  -----  ------------------------------
7-4                    0    Reserved
3        DNS           0    Done Not Synchronized is cleared
2        BES           0    Bus Error Source is cleared
1        BED           0    Bus Error Destination is cleared
0        DONE          0    Normal Channel Transfer Done is cleared

STATUS:   M68302--Running in monitor      Software break: 0002bc0@p

display  registers CSR
```

To modify register contents

- Choose **Modify→Register...** and use the dialog box to name the register and specify its value.



Clicking the "Recall" pushbutton lets you select register names and values from predefined or previously specified entries.

Placing the mouse pointer in the text entry area lets you type in the register name and value.

To define the type of value, press and hold the *command select* mouse button and drag the mouse to select the value type.

Clicking this checkbox causes the current value of the named register to be placed in the "Value" text entry area.

Clicking this button modifies the register to the value specified and closes the dialog box.

Clicking this button modifies the register to the value specified and leaves the dialog box open.

Clicking this button cancels modification and closes the dialog box.

Or, using the command line, enter:

```
modify register <register> to <value>
```

Displaying and Modifying Memory

You can display and modify the contents of memory in hexadecimal formats and in real number formats. You can also display the contents of memory in assembly language mnemonic format.

This section shows you how to:

- Display memory.
- Display memory in mnemonic format.
- Display memory in mnemonic format at the current PC.
- Return to the previous mnemonic display.
- Display memory in hexadecimal format.
- Display memory in real number format.
- Display memory at an address.
- Display memory repetitively.
- Modify memory.
- Modify memory at an address.

To display memory

- Choose **Display→Memory**.

This command either re-displays memory in the format specified by the last memory display command, or, if no previous command has been executed, displays memory as hexadecimal bytes beginning at address zero.

To display memory in mnemonic format

- To display memory at a particular address, place an absolute or symbolic address in the entry buffer; then, choose **Display→Memory→Mnemonic ()**, or, using the command line, enter the ***display memory <address> mnemonic*** command.
- To display memory at the current program counter address, choose **Display→Memory→Mnemonic at PC**, or, using the command line, enter the ***display memory mnemonic at_pc*** command.

A highlighted bar shows the location of the current program counter address. This allows you to view the program counter while stepping through user program execution.

Whether source lines, assembly language instructions, or symbols are included in the display depends on the modes you choose with the **Settings→Source/Symbols Modes** or **Settings→Display Modes** pull-down menu items. See the "Changing the Interface Settings" section.

If symbols are loaded into the interface, the default is to display source only.

To return to the previous mnemonic display

- Choose **Display→Memory→Mnemonic Previous**.

  Or, using the command line, enter:

  ```
  display memory mnemonic previous_display
  ```

  This command is useful for quickly returning to the previous mnemonic memory display.

  For example, suppose you are stepping source lines and you step into a function that you would like to step over. You can return to the previous mnemonic memory display, set a breakpoint at the line following the function call, and run the program from the current program counter.

To display memory in hexadecimal format

- Place an absolute or symbolic address in the entry buffer; then, choose **Display→Memory→Hex ()** and select the size from the cascade menu.

Or, using the command line, enter:

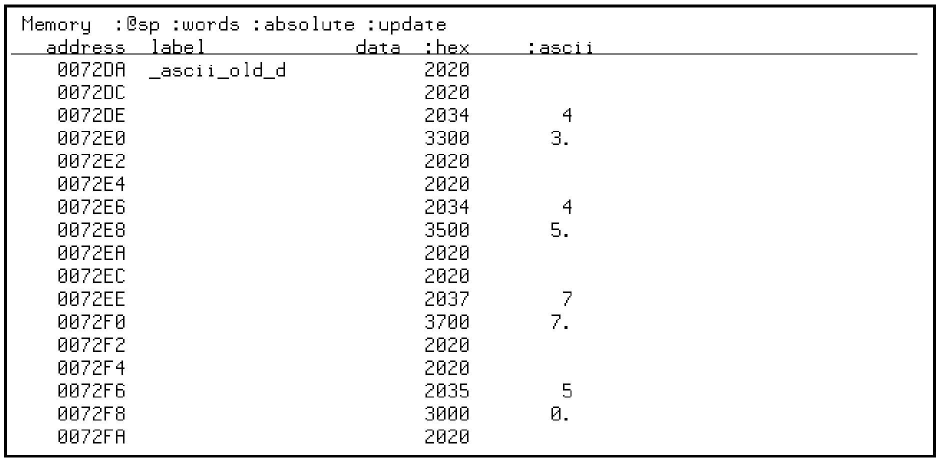
display memory `<address>` ***blocked*** `<size>`

This command displays memory as hexadecimal values beginning at the address in the entry buffer.

Examples

To display memory in absolute word format:

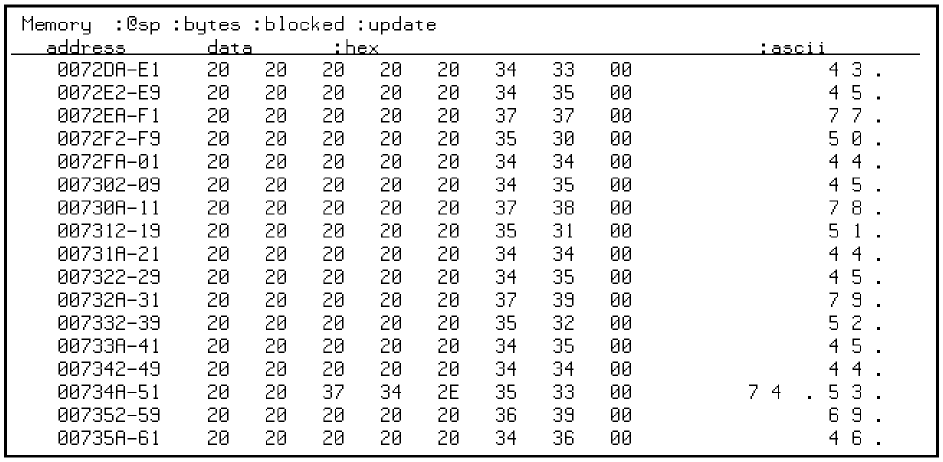
display memory `ascii_old_data` ***absolute words***



```
Memory  :@sp :words :absolute :update
   address  label            data  :hex     :ascii
    0072DA  _ascii_old_d           2020
    0072DC                         2020
    0072DE                         2034        4
    0072E0                         3300       3.
    0072E2                         2020
    0072E4                         2020
    0072E6                         2034        4
    0072E8                         3500       5.
    0072EA                         2020
    0072EC                         2020
    0072EE                         2037        7
    0072F0                         3700       7.
    0072F2                         2020
    0072F4                         2020
    0072F6                         2035        5
    0072F8                         3000       0.
    0072FA                         2020
```

To display memory in blocked byte format:

display memory ascii_old_data ***blocked bytes***



```
Memory  :@sp :bytes :blocked :update
  address       data       :hex                                      :ascii
   0072DA-E1    20   20   20   20   20   34   33   00                   4 3 .
   0072E2-E9    20   20   20   20   20   34   35   00                   4 5 .
   0072EA-F1    20   20   20   20   20   37   37   00                   7 7 .
   0072F2-F9    20   20   20   20   20   35   30   00                   5 0 .
   0072FA-01    20   20   20   20   20   34   34   00                   4 4 .
   007302-09    20   20   20   20   20   34   35   00                   4 5 .
   00730A-11    20   20   20   20   20   37   38   00                   7 8 .
   007312-19    20   20   20   20   20   35   31   00                   5 1 .
   00731A-21    20   20   20   20   20   34   34   00                   4 4 .
   007322-29    20   20   20   20   20   34   35   00                   4 5 .
   00732A-31    20   20   20   20   20   37   39   00                   7 9 .
   007332-39    20   20   20   20   20   35   32   00                   5 2 .
   00733A-41    20   20   20   20   20   34   35   00                   4 5 .
   007342-49    20   20   20   20   20   34   34   00                   4 4 .
   00734A-51    20   20   37   34   2E   35   33   00              7 4  . 5 3 .
   007352-59    20   20   20   20   20   36   39   00                   6 9 .
   00735A-61    20   20   20   20   20   34   36   00                   4 6 .
```

To display memory in real number format

- Place an absolute or symbolic address in the entry buffer; then, choose **Display→Memory→Real ()** and select the size from the cascade menu.

Or, using the command line, enter:

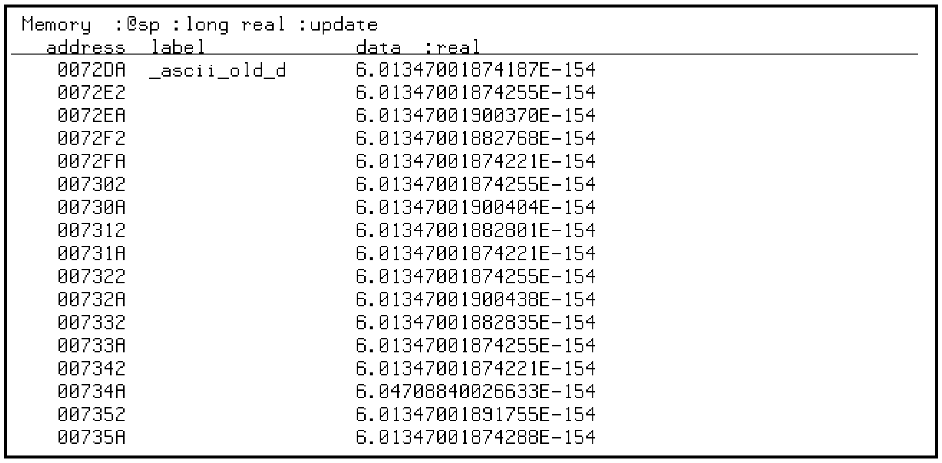
display memory <address> ***real*** <size>

Displays memory as a list of real number values beginning at the address in the entry buffer. Short means four-byte real numbers and long means eight-byte real numbers.

Examples

To display memory in 64-bit real number format:

display memory real long



```
Memory  :@sp :long real :update
  address  label             data  :real
    0072DA  _ascii_old_d      6.01347001874187E-154
    0072E2                    6.01347001874255E-154
    0072EA                    6.01347001900370E-154
    0072F2                    6.01347001882768E-154
    0072FA                    6.01347001874221E-154
    007302                    6.01347001874255E-154
    00730A                    6.01347001900404E-154
    007312                    6.01347001882801E-154
    00731A                    6.01347001874221E-154
    007322                    6.01347001874255E-154
    00732A                    6.01347001900438E-154
    007332                    6.01347001882835E-154
    00733A                    6.01347001874255E-154
    007342                    6.01347001874221E-154
    00734A                    6.04708840026633E-154
    007352                    6.01347001891755E-154
    00735A                    6.01347001874288E-154
```

To display memory at an address

- Place an absolute or symbolic address in the entry buffer; then, choose **Display→Memory→At ()**.

This command displays memory in the same format as that of the last ***memory display*** command. If no previous command has been issued, memory is displayed as hexadecimal bytes.

To display memory repetitively

- Choose **Display→Memory→Repetitively**.

Or, using the command line, enter:

```
display memory repetitively
```

The memory display is constantly updated. The format is specified by the last memory display command.

This command is ignored if the last memory display command was a mnemonic display.

To modify memory

- Choose **Modify→Memory** and complete the command using the command line.
- To modify memory at a particular address, place an absolute or symbolic address in the entry buffer; then, choose **Modify→Memory at ()** and complete the command using the command line.

Or, using the command line, enter:

```
modify memory
```

You can modify the contents of one memory location or a range of memory locations. Options allow you to modify memory in byte, short, word, and real number formats.

Displaying Data Values

The data values display lets you view the contents of memory as data types. You can display data values in the following formats:

- bytes
- 8-bit integers
- unsigned 8-bit integers
- chars
- words
- 16-bit integers
- unsigned 16-bit integers
- long words
- 32-bit integers
- unsigned 32-bit integers

This section shows you how to:

- Display data values.
- Clear the data values display and add a new item.
- Add item to the data values display.

To display data values

- Choose **Display→Data Values**.

Or, using the command line, enter:

```
display data
```

Items must be added to the data values display before you can use this command.

The data display shows the values of simple data types in the user program. When the display mode setting turns ON symbols, a label column that shows symbol values is added to the data display.

Step commands and commands that cause the emulator to enter the monitor (for example, encountering a breakpoint) cause the data values screen to be updated.

To clear the data values display and add a new item

- Place an absolute or symbolic address in the entry buffer; then, choose **Display**→**Data Values**→**New ()** and select the data type from the cascade menu.

  Or, using the command line, enter:

```
display data <address> <format>
```

To add items to the data values display

- Place an absolute or symbolic address in the entry buffer; then, choose **Display**→**Data Values**→**Add ()** and select the data type from the cascade menu.

  Or, using the command line, enter:

```
display data , <address> <format>
```

Changing the Interface Settings

This section shows you how to:

- Set the source/symbol modes.
- Set the display modes.

To set the source/symbol modes

- To display assembly language mnemonics with absolute addresses, choose **Settings→Source/Symbol Modes→Absolute**, or, using the command line, enter the ***set source off symbols off*** command.
- To display assembly language mnemonics with absolute addresses replaced by global and local symbols where possible, choose **Settings→Source/Symbol Modes→Symbols**, or, using the command line, enter the ***set source off symbols on*** command.
- To display assembly language mnemonics intermixed with high-level source lines, choose **Settings→Source/Symbol Modes→Source Mixed**, or, using the command line, enter the ***set source on inverse_video on symbols on*** command.
- To display only high-level source lines, choose **Settings→Source/Symbol Modes→Source Only**, or, using the command line, enter the ***set source only inverse_video off symbols on*** command.

The source/symbol modes affect mnemonic memory displays and trace displays. Each display mode cascade menu choice is a toggle. Choosing one of these items causes it to be the only one active and toggles all others off. Provided that symbols were loaded, the interface defaults to:

- Source only for mnemonic memory displays.
- Source mixed for trace listing displays.

To set the display modes

- Choose **Settings→Display Modes...** to open the display modes dialog box.

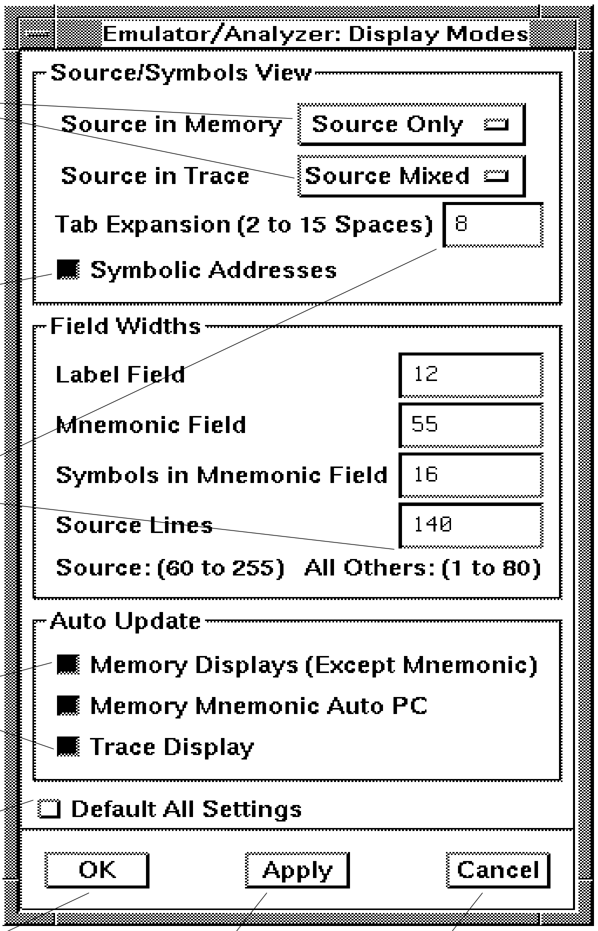
Press and hold the *select* mouse button and drag the mouse to select "Source Only", "Source Mixed", or "Off".

Clicking toggles whether symbolic information is displayed.

Move the mouse pointer to the text entry area and type in the value. Descriptions of the modes follow.

Clicking toggles auto update settings.

Clicking this checkbox changes all display mode settings to their defaults.



Source/Symbols View

Source in Memory: Source Only
Source in Trace: Source Mixed
Tab Expansion (2 to 15 Spaces): 8
■ Symbolic Addresses

Field Widths

Label Field: 12
Mnemonic Field: 55
Symbols in Mnemonic Field: 16
Source Lines: 140
Source: (60 to 255) All Others: (1 to 80)

Auto Update

■ Memory Displays (Except Mnemonic)
■ Memory Mnemonic Auto PC
■ Trace Display

☐ Default All Settings

Clicking this button saves your changes and closes the dialog box.

Clicking this button saves your changes and leaves the dialog box open.

Clicking this button cancels your changes and closes the dialog box.

Source/Symbols View

Source in Memory specifies whether source lines are included, mixed with assembly code, or excluded from mnemonic memory displays.

Source in Trace specifies whether source lines are included, mixed with stored states, or excluded from trace displays.

Symbolic Addresses specifies whether symbols are included in displays.

Tab Expansion sets the number of spaces displayed for tabs in source lines.

Source/Symbols View

Label Field sets the width (in characters) of the address field in the trace list or label (symbols) field in any of the other displays.

Mnemonic Field sets the width (in characters) of the mnemonic field in memory mnemonic, trace list, and register step mnemonic displays. It also changes the width of the status field in the trace list.

Symbols in Mnemonic Field sets the maximum width of symbols in the mnemonic field of the trace list, memory mnemonic, and register step mnemonic displays.

Source Lines sets the width (in characters) of the source lines in the memory mnemonic display.

Auto Update

Memory Displays (Except Mnemonic) toggles whether memory displays are automatically updated after commands that change memory contents or whether you must enter memory display commands to update the display. You may wish to turn off memory display updates, for example, when displaying memory mapped I/O.

Memory Mnemonic Auto PC toggles whether memory mnemonic displays automatically jump to the new PC location when the PC changes (such as during stepping or break). You may wish to turn off the automatic update of memory mnemonic displays when you want to examine a specific area of memory regardless of the location of the current PC (such as during stepping).

Trace Display toggles whether trace displays are automatically updated when trace measurements complete or whether you must enter trace display commands to update the display. You may wish to turn off trace display updates in one emulator/analyzer window in order to compare the display with a new trace display in another emulator/analyzer window.

Using System Commands

With the Softkey Interface system commands, you can:

- Set UNIX environment variables while in the Softkey Interface.
- Display the name of the emulation module.
- Display the event log.
- Display the error log.

To set UNIX environment variables

- **Using the command line, enter:**
- ***set <VAR>***

You can set UNIX shell environment variables from within the Softkey Interface with the ***set <environment_variable> = <value>*** command.

Examples

To set the PRINTER environment variable to "lp -s":

```
set PRINTER = "lp -s"
```

After you set an environment variable from within the Softkey Interface, you can verify the value of it by entering ***!set***.

To display the name of the emulation module

- **Using the command line, type the name_of_module command.**

While operating your emulator, you can verify the name of the emulation module. This is also the logical name of the emulator in the emulator device file.

Examples

To display the name of your emulation module:

```
name_of_module
```

The name of the emulation module is displayed on the status line.

To display the event log

- Choose **Display→Event Log**.
- Position the mouse pointer on the status line, press and hold the *select* mouse button, and then choose **Display Event Log** from the pop-up menu.

Or, using the command line, enter:

```
display event_log
```

The last 100 events that have occurred during the emulation session are displayed.

The status of the emulator and analyzer are recorded in the event log, as well as the conditions that cause the status to change (for example, software breakpoints and trace commands).

To display the error log

- Choose **Display→Error Log**.
- Position the mouse pointer on the status line, press and hold the *select* mouse button, and then choose **Display Error Log** from the pop-up menu.

Or, using the command line, enter:

```
display error_log
```

The last 100 error messages that have occurred during the emulation session are displayed.

To edit files

- Choose **File→Edit→File** and use the dialog box to specify the file name.
- To edit a file based on an address in the entry buffer, place an address reference (either absolute or symbolic) in the entry buffer; then, choose **File→Edit→At ( ) Location**.
- To edit a file based on the current program counter, choose **File→Edit→At PC Location**.
- To edit a file associated with a symbol when you are displaying symbols, position the mouse pointer over the symbol, press and hold the *select* mouse button, and choose **Edit File Defining Symbol** from the pop-up menu.
- To edit a file when displaying memory in mnemonic format, position the mouse pointer over the line of source where you want to begin the edit, press and hold the *select* mouse button, and choose **Edit Source** from the pop-up menu.

When editing files at addresses, the interface determines which source file contains the code generated for the address and opens an edit session on the file. The interface will issue an error if it cannot find a source file for the address.

If, upon starting the emulator session, you chose to copy files to a new directory, the file you try to edit will have write permissions. This same file will be "read only" if you did not choose to copy files to a new directory upon starting up the emulator.

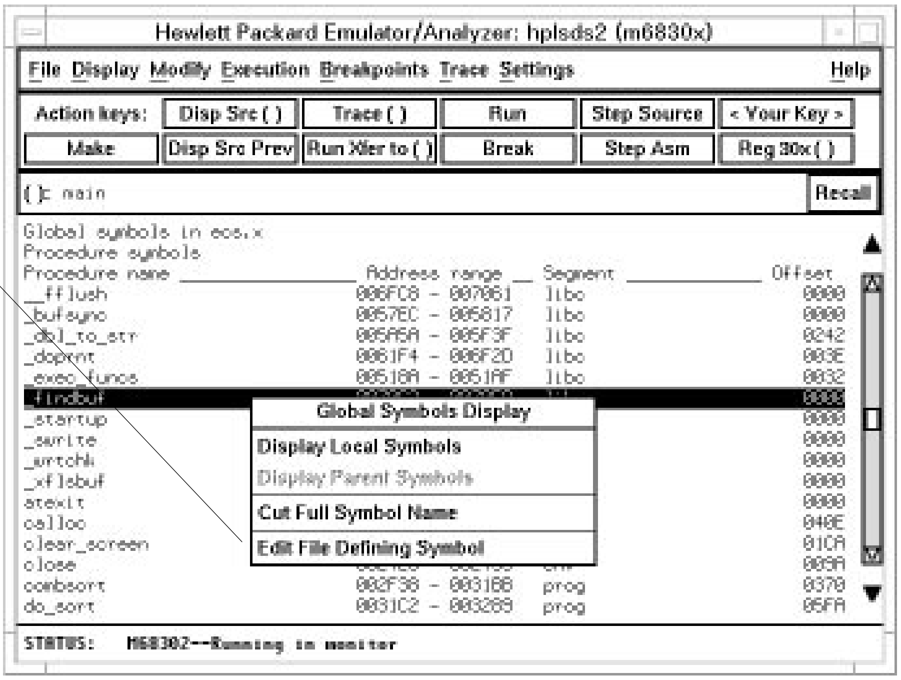
The interface will choose the "vi" editor as its default editor, unless you specify another editor by setting an X resource. Refer to the "Setting X Resources" chapter for more information about setting this resource.

You must load symbols before most commands will work because symbol information is needed to be able to locate the files.

Examples

To edit a file that defines a symbol:

Choosing the Edit File Defining Symbol menu item brings up a terminal window with an edit session open on the file where the highlighted symbol is defined.


Hewlett Packard Emulator/Analyzer: hplsds2 (m6830x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 30x( )
Recall
Global Symbols Display
Display Local Symbols
Display Parent Symbols
Cut Full Symbol Name
Edit File Defining Symbol

```
( ): main

Global symbols in ecs.x
Procedure symbols
Procedure name _______________ Address range __ Segment _____________ Offset
__fflush                        006FC8 - 007061   libc                  0000
_bufsync                        0057EC - 005817   libc                  0000
_dbl_to_str                     005A5A - 005F3F   libc                  0242
_doprnt                         0061F4 - 006F2D   libc                  003E
_exec_funcs                     00518A - 0051AF   libc                  0032
_findbuf                                                                0000
_startup                                                                0000
_swrite                                                                 0000
_wrtchk                                                                 0000
_xflsbuf                                                                0000
atexit                                                                  0000
calloc                                                                  040E
clear_screen                                                            01CA
close                                                                   005A
combsort                        002F38 - 0031BB   prog                  0370
do_sort                         0031C2 - 003289   prog                  05FA

STATUS:   M68302--Running in monitor
```

To copy information to a file or printer

- **Choose File→Copy, select the type of information from the cascade menu, and use the dialog box to select the file or printer.**

Or, using the command line, enter:

```
copy
```

ASCII characters are copied to the file or printer.

If you copy information to an existing file, it will be appended to the file.

Refer to the following paragraphs for details about the different copy options.

Display ... Copies information currently in the display area. This option is useful for restricting the number of lines that are copied. Also, this option is useful for copying the contents of register classes other than BASIC.

Memory ... Copies the contents of a range of memory. The format is the same as specified in the last display memory command. For example, if you copy memory after displaying a range of memory in mnemonic format, the file would contain the mnemonic memory information. If there is no previous display memory command, the format used is a blocked hex byte format beginning at address zero.

Data Values ... Copies the contents of the defined data values last displayed. An error occurs if you try to copy data values to a file if you have not yet displayed data values.

Configuration Info ... Copies the contents of the configuration information last displayed. An error occurs if you try to copy configuration information to a file if you have not yet displayed any.

Trace ... The most recently captured trace is copied to the file. The copied trace listing is formatted according to the current display mode.

You can set the display mode with the **Settings→Source/Symbols Modes** or **Settings→Display Modes** pull-down menu items. See the "Changing the Interface Settings" section.

Registers ... Copies the current values of the BASIC register class to a file. To copy the contents of the other register classes, first display the registers in that class, and then use the **File→Copy→Display ...** command.

Breakpoints ... Copies the breakpoints list. If no breakpoints are present in the list, only the enable/disable status is copied.

Status ... Copies the emulator/analyzer status display.

Global Symbols ... Copies the global symbols. If symbols have not been loaded, this menu item is grayed-out and unresponsive.

Local Symbols ( ) ... Copies the local symbols from the symbol scope named (by an enclosing symbol) in the entry buffer. If symbols have not been loaded, this menu item is grayed-out and unresponsive.

Pod Commands ... Copies the last 100 lines from the pod commands display.

Error Log ... Copies the last 100 lines from the error log display.

Event Log ... Copies the last 100 lines from event log display.

To open a terminal emulation window

- Choose **File→Term...**

This command opens a terminal window into the current working directory context.

Using Simulated I/O

Simulated I/O is a feature of the emulator/analyzer interface that lets you use the same keyboard and display that you use with the interface to provide input to programs and display program output.

To use simulated I/O, your programs must communicate with the simulated I/O control address and the buffer locations that follow it. (The Hewlett-Packard AxLS compilers, if your program uses I/O, automatically link with environment dependent routines that communicate with the simulated I/O control address and buffer.)

Also, before simulated I/O can work, the emulator must be configured to enable polling of the simulated I/O control address and to define the control address location.

This section shows you how to:

- Display the simulated I/O screen.
- Use simulated I/O keyboard input.

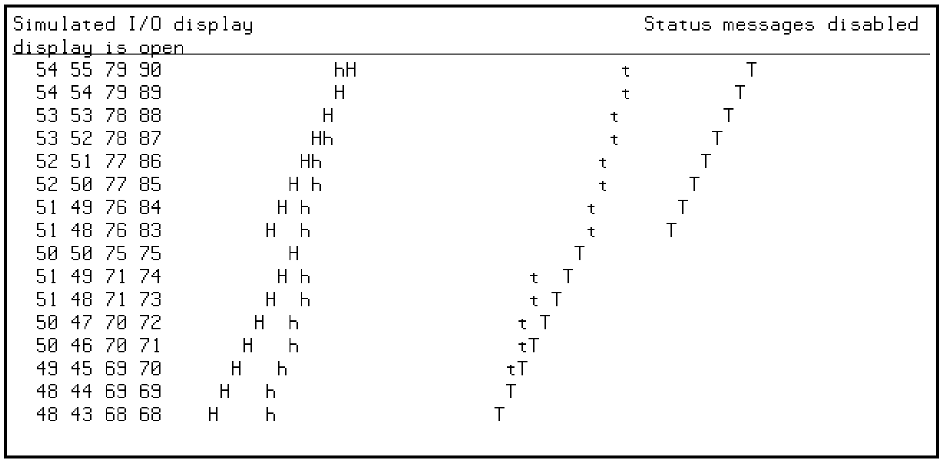
Refer to the *Simulated I/O User's Guide* for complete details on how simulated I/O works.

To display the simulated I/O screen

- Choose **Display→Simulated IO**.

Before you can display simulated I/O, polling for simulated I/O must be enabled in the emulator configuration.

Examples



```
Simulated I/O display                                  Status messages disabled
display is open
  54 55 79 90               hH                       t          T
  54 54 79 89               H                        t         T
  53 53 78 88              H                        t         T
  53 52 78 87             Hh                        t        T
  52 51 77 86            Hh                        t        T
  52 50 77 85           H h                        t       T
  51 49 76 84          H h                        t       T
  51 48 76 83         H  h                        t      T
  50 50 75 75           H                        T
  51 49 71 74          H h                   t  T
  51 48 71 73         H  h                   t T
  50 47 70 72        H  h                   t T
  50 46 70 71       H   h                   tT
  49 45 69 70      H   h                   tT
  48 44 69 69     H   h                    T
  48 43 68 68    H    h                   T
```

A message tells you whether the display is open or closed. You can modify the configuration to enable status messages.

To use simulated I/O keyboard input

- To begin using simulated I/O input, choose **Settings→Simulated IO Keyboard**.
- To end simulated I/O and return to using the interface, use the ***suspend*** softkey.

For Simulated I/O to work, you must configure the emulator to enable polling of simulated I/O.

The command line entry area is used for simulated input with the keyboard. Therefore, if the command line is turned off, choosing this menu item with turn command line display back on.

If you are planning to use even a modest amount of simulated I/O input during an emulation session, it might be a good idea to open another Emulator/Analyzer window to be used exclusively for simulated I/O input and output.

Using Basis Branch Analysis

Basis branch analysis (BBA) is provided by the HP Branch Validator product. This product is used to analyze the testing of your programs, create more complete test suites, and quantify your level of testing.

The HP Branch Validator records branches executed in a program and generates reports that provide information about program execution during testing. It uses a special C preprocessor to add statements that write to a data array when program branches are taken. After running the program in the emulator (using test input), you can store the BBA information to a file. Then, you can generate reports based on the stored information.

This section shows you how to store BBA data to a file. Refer to the *HP Branch Validator (BBA) User's Guide* for complete details on the BBA product and how it works.

To store BBA data to a file

- Choose **File→Store→BBA Data** and use the selection dialog box to specify the file name.

  The default file name "bbadump.data" can be selected from the dialog box.

7

Using the Emulation Analyzer

Using the Emulation Analyzer

This chapter describes tasks you may wish to perform while using the emulation analyzer. These tasks are grouped into the following sections:

- The basics of starting, stopping, and displaying traces.
- Qualifying trigger and store conditions.
- Using the sequencer.
- Displaying the trace list.
- Saving and restoring trace data and specifications.

The Basics of Starting, Stopping, and Displaying Traces

This section describes the basic tasks that relate to starting and stopping trace measurements.

When you start a trace measurement, the analyzer begins looking at the data on the emulation processor's bus and control signals on each analyzer clock signal. The information seen on a particular clock is called a state.

When one of these states matches the "trigger state" you specify, the analyzer stores states in trace memory. When trace memory is filled, the trace is said to be "complete." The default trigger state specification is "any state," so when you start a trace measurement after initializing the analyzer, the analyzer will "trigger" on the first state it sees and store the following states in trace memory.

Once you start a trace measurement, you can view the progress of the measurement by displaying the trace status.

In some situations, for example, when the trigger state is never found, the trace measurement does not complete. In these situations, you can halt the trace measurement.

Once a trace is displayed, you can use the cursor keys and other keys to position the trace list on the display. To speed up the display of traces, you can reduce the depth of the trace list. Also, when entering trace commands, there is a special command that allows you to recall and modify the last trace command entered.

This section describes how to:

- Start a trace measurement.
- Display the trace list.
- Display the trace status.
- Change the trace depth.
- Modify the last trace command entered.

- Repeat the previous trace command.
- Position the trace display on the screen.

To start a trace measurement

- Choose **Trace→Everything**.

Or, using the command line, enter:

```
trace
```

When you use the ***trace*** command without any options, the analyzer begins recording processor bus cycles immediately, and continues until the trace buffer is filled. In the default trace configuration, the analyzer stores all bus cycles.

If you are using the emulation-bus analyzer with deep memory, the depth of the trace list buffer depends on whether or not you installed memory modules on the analyzer card, and the capacity of the memory modules installed. Refer to the Hewlett-Packard *Emulator-Bus Analyzer (with deep trace memory) User's Guide* for details. If you are using the 1K analyzer, the trace list buffer is 512 or 1024 states deep (depending on whether or not you turn on the state/time count). See "To count states or time" in this chapter.)

Example

From the demo directory /usr/hp64000/demo/debug_env/hp64798, start the demo program and trace from the program start:

```
Startemul
reset
trace
run from transfer_address
```

To stop a trace measurement

- Choose **Trace→Stop**.

Or, using the command line, enter:

```
stop_trace
```

You must use this command to stop a trace started with a **Trace→Until Stop** command (refer to "To trace activity leading up to a program halt" later in this chapter). Several other conditions may occur that will make you want to stop a trace. The analyzer may not record any trace states because your trigger specification isn't correct, or because you have a target system problem. At other times, a valid trace may be capturing data slowly. You can use the ***stop_trace*** command to prevent the analyzer from storing additional data.

You do not have to stop a trace in order to begin viewing a partial trace because the interface supports incremental trace uploading. After the trigger condition occurs, the interface begins uploading and displaying trace states as they are captured.

To display the trace list

- Choose **Trace→Display** or **Display→Trace**.

Or, using the command line, enter:

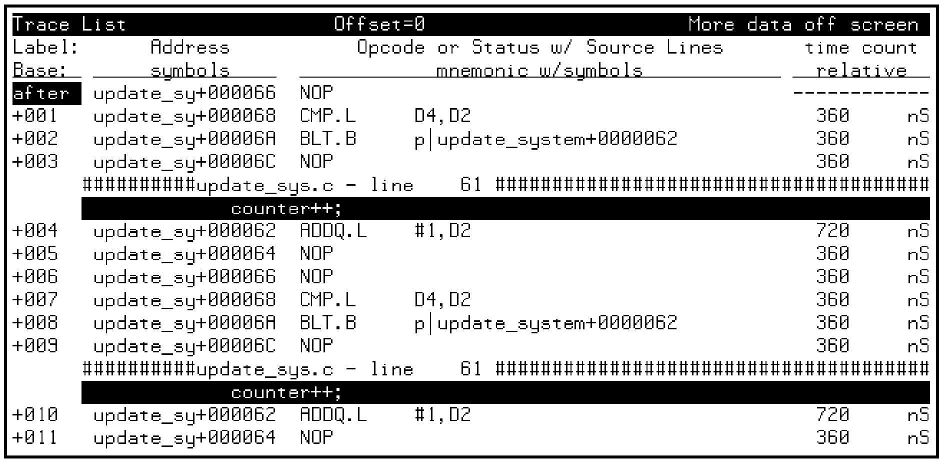
```
display trace
```

You can display captured trace data with the ***display trace*** command. The available options to the display trace command are described in the "Modifying the Trace Display" section later in this chapter.

Examples

To display the trace:

display trace



```
Trace List                    Offset=0                     More data off screen
Label:      Address          Opcode or Status w/ Source Lines       time count
Base:       symbols                mnemonic w/symbols                relative
after  update_sy+000066  NOP                                       -----------
+001   update_sy+000068  CMP.L     D4,D2                             360     nS
+002   update_sy+00006A  BLT.B     p|update_system+0000062           360     nS
+003   update_sy+00006C  NOP                                         360     nS
      ##########update_sys.c - line    61 #####################################
                   counter++;
+004   update_sy+000062  ADDQ.L    #1,D2                             720     nS
+005   update_sy+000064  NOP                                         360     nS
+006   update_sy+000066  NOP                                         360     nS
+007   update_sy+000068  CMP.L     D4,D2                             360     nS
+008   update_sy+00006A  BLT.B     p|update_system+0000062           360     nS
+009   update_sy+00006C  NOP                                         360     nS
      ##########update_sys.c - line    61 #####################################
                   counter++;
+010   update_sy+000062  ADDQ.L    #1,D2                             720     nS
+011   update_sy+000064  NOP                                         360     nS
```

The first column in the trace list contains the line number. The trigger is always on line 0.

The second column contains the address information associated with the trace states. Addresses in this column may be locations of instruction opcodes on fetch cycles, or they may be sources or destinations of operand cycles.

The third column shows mnemonic information about the emulation bus cycle.

The next column shows the count information (time is counted by default). "Relative" indicates that each count is relative to the previous state.

If your analyzer card contains external analysis (for example, HP 64703), the next column shows the data captured on the external trace signals.

You can use the <NEXT> and <PREV> keys to scroll through the trace list a page at a time. The <Up arrow> and <Down arrow> keys will scroll through the trace list a line at a time. You can also display the trace list centered around a specific line number (for example, ***display trace* 100**). Refer to the "Modifying the Trace Display" section for more information on the trace list display.

Note that when a trigger condition is found but not enough states are captured to fill trace memory, the status line will show the trace is still running. You can display all but the last captured state in this situation; you must halt the trace to display the last captured state.

To display the trace status

- Choose **Display→Status**.

Or, using the command line, enter:

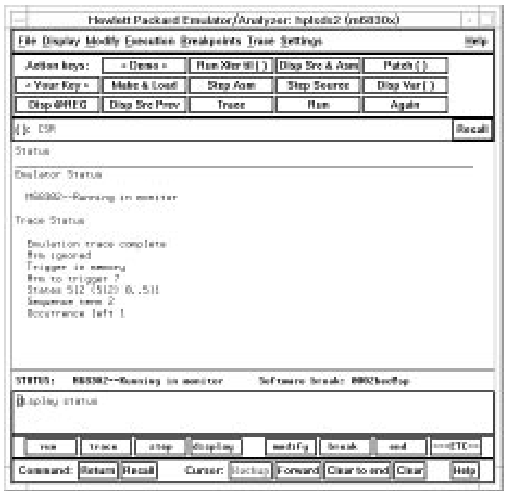
display status

In addition to the analyzer information shown on the status line (Emulation trace started, Emulation trace complete, etc.), you can display complete analyzer status with the command below.

Examples

To display the trace status:

display status



```
( ): CSR

Status

Emulator Status

  M68302--Running in monitor

Trace Status

   Emulation trace complete
   Arm ignored
   Trigger in memory
   Arm to trigger ?
   States 512 (512) 0..511
   Sequence term 2
   Occurrence left 1

STATUS:    M68302--Running in monitor      Software break: 0002bed0@p

display status
```

The first line of the emulation trace status display shows the user trace has been "completed"; other possibilities are that the trace is still "running" or that the trace has been "halted".

The second line of the trace status display contains information on the arm condition. If the analyzer is always armed, the message "Arm ignored" is displayed. If the analyzer is to be armed by one of the internal signals, either the message "Arm not received" or "Arm received" is displayed. The display indicates if the arm condition happened any time since the most recent trace started, even if it happened after the trace was halted or became complete. (The "Making Coordinated Measurements" chapter explains arm conditions.)

The "Arm to trigger" line displays the amount of time between the arm condition and the trigger. The time displayed will be from -0.04 microseconds to 41.943 milliseconds, less than -0.04 microseconds, or greater than 41.943 milliseconds. If the arm signal is ignored or the trigger is not in memory, a question mark (?) is displayed.

The "States" line shows the number of states that have been stored (out of the number that is possible to store) and the line numbers that the stored states occupy. (The trigger state is always stored on line 0.)

The "Sequence term" line of the trace status display shows the number of the term the sequencer was in when the trace completed. Because a branch out of the last sequence term constitutes the trigger, the number displayed is what would be the next term (2 in the preceding example) even though that term is not defined. If the trace is halted, the sequence term number just before the halt is displayed; otherwise, the current sequence term number is displayed. If the current sequence term is changing too quickly to be read, a question mark (?) is displayed.

The "Occurrence left" line of the trace status display shows the number of occurrences remaining before the primary branch can be taken out of the current sequence term. If the occurrence left is changing too quickly to be read, a question mark (?) is displayed.

To change the trace depth

- **Choose Trace→Display Options... and in the dialog box, enter the desired trace unload depth in the field beside Unload Depth. Then click the OK or Apply pushbutton.**

Or, using the command line, enter:

```
display trace depth <depth>
```

Using one of the above command forms, you specify the number of states that will be unloaded for display, copy, or file storage. By reducing the trace unload depth, you shorten the time it takes for the interface to unload the trace information. You can increase the trace unload depth to view more states of the current trace. Regardless of how much or how little unload depth you specify, the entire trace memory will be filled with captured states during a trace.

In the emulation-bus analyzer with deep memory, the maximum number of trace states depends on whether or not you installed memory modules in the analyzer card, and the capacity of the memory modules. Refer to the Hewlett-Packard *Emulation-Bus Analyzer (with deep trace memory) User's Guide* for details. In the 1K analyzer, the maximum number of trace states is 1024 when counting is turned off, and 512 otherwise. In either analyzer, the minimum trace depth is 9.

Trace data must be unloaded before it can be displayed, copied, or stored in a file. If you wish to reduce the number of states that are unloaded for display, you must enter the unload depth specification (in one of the two ways shown above) before you enter the trace command. The above commands cannot be used to reduce the number of states displayed in the current trace. You can enter a new unload depth specification after a trace is complete to increase the amount of trace memory that is unloaded, if desired.

To modify the last **trace** command entered

- **Choose Trace→Trace Spec... and use the dialog box to select and edit a trace command.**

Or, using the command line, enter:

```
trace modify_command
```

The Trace Specification Selection dialog box lets you recall, edit, and enter trace commands that have been executed during the emulation session or trace commands that have been predefined. If you make an error in a trace command or want to change the measurement slightly, it’s often easier to recall the previous trace command and edit it than it is to enter a new trace command.

You can predefine trace specifications and set the maximum number of entries for the dialog box by setting X resources (see the "Setting X Resources" chapter).

The ***trace modify_command*** command recalls the last trace command. The advantage of this command over command recall is that you do not have to move forward and backward over other commands to find the last trace command; also, the last trace command is always available, no matter how many commands have since been entered.

To repeat the previous trace command

- **Choose Trace→Again.**
- **To continually repeat the last trace, choose Trace→Repetitively.**

Or, using the command line, enter:

```
trace again
```

The ***trace again*** command is most useful when you want to repeat a measurement with the same trace specification. It saves you the trouble of reentering the complete trace command specification.

The "repetitively" choice continually repeats the last trace command. Successive traces begin as soon as the results from the just-completed trace are displayed.

Also, this command is useful when you load a trace specification from a file. (See "To load a trace specification" in this chapter.)

To position the trace display on screen

- **Use the scroll bar or the <Up arrow>, <Down arrow>, <PREV>, <NEXT>, <CTRL>f, and <CTRL>g keys.**

The trace display command can display more states than can appear on the screen at one time. However, you can reposition the display on the screen with the keys described below.

The <Up arrow> and <Down arrow> (or roll up and roll down) keys move the display up or down on the screen one line at a time.

The <PREV> and <NEXT> (or page up and page down) keys allow you to move the display up or down a page at a time.

The <CTRL>f and <CTRL>g keys allow you to move the display left or right, respectively. These keys are used when the width of the address or mnemonic/absolute columns is increased so that not all the trace display data can be displayed across the screen.

Qualifying Trigger and Store Conditions

This section describes tasks relating to the qualification of trigger and storage states.

You can trigger on, or store, specific states or specific values on a set of trace signals (which are identified by trace labels).

Also, you can *prestore* states. The prestore qualifier is a second storage qualifier used for storing states that occur before the normally stored states. Prestore is useful for capturing entry points to procedures or for identifying where global variables are accessed from.

This section describes how to:

- Specify a trigger and set the trigger position.
- Use address, data, and status values in trace expressions.
- Enter a range in a trace expression.
- Trigger on a number of occurrences of some state.
- Break emulator on execution on the analyzer trigger.
- Count states or time.
- Define a storage qualifier.
- Define a prestore qualifier.
- Trace activity leading up to a program halt.

Expressions in Trace Commands

When modifying the analysis specification, you can enter expressions which consist of values, symbols, and operators.

Values Values are numbers in hexadecimal, decimal, octal, or binary. These number bases are specified by the following characters:

B b	Binary (example: 10010110b).
Q q O o	Octal (example: 377o or 377q).
D d (default)	Decimal (example: 2048d or 2048).
H h	Hexadecimal (example: 0a7fh).

You must precede any hexadecimal number that begins with an A, B, C, D, E, or F with a zero.

Don’t care digits may be included in binary, octal, or hexadecimal numbers and they are represented by the letters X or x. A zero must precede any numerical value that begins with an "X".

Symbols A symbol database is built when the absolute file is loaded into the emulator. Both global and local symbols can be used when entering expressions. Global symbols are entered as they appear in the global symbols display. When specifying a local symbol, you must include the name of the module ("anly.c") as shown below.

```
anly.c:cmp_function
```

Operators Analysis specification expressions may contain operators. All operations are carried out on 32-bit, two’s complement integers. (Values which are not 32 bits will be sign extended when expression evaluation occurs.)

The available operators are listed below in the order of evaluation precedence. Parentheses are also allowed in expressions to change the order of evaluation.

-, **~**	Unary two’s complement, unary one’s complement. The unary two’s complement operator is not allowed on constants containing don’t care bits.
*****, **/**, **%**	Integer multiply, divide, and modulo. These operators are not allowed on constants containing don’t care bits.
+, **-**	Addition, subtraction. These operators are not allowed on constants containing don’t care bits.
&	Bitwise AND.
|	Bitwise inclusive OR.

Values, symbols, and operators may be used together in analysis specification expressions. For example, if the local symbol exists, the following is a valid expression:

```
module.c:symb+0b67dh&0fff00h
```

However, you cannot add two symbols unless one of them is an EQU type symbol.

Emulation Analyzer Trace Signals

When you qualify states, you specify values that should be found on the analyzer trace signals. The signals are described in the table that follows.

Emulation Analyzer Trace Signals			
Analyzer Channel #	Status Bit #	Signal Name	Signal Description
0-23		A0-A23	Address lines 0-23
24-31		A24-A31	Address lines 24-31 (68306 only)
32-47		D0-D15	Data lines 0-15
48	0	*BKG	0 = in background monitor 1 = in foreground monitor
49-51	1-3	FC0-FC2	Function Code lines 0-2
52	4	R/*W	0 = write bus cycle 1 = read bus cycle
53	5	IAC	0 = external processor cycle 1 = internal processor cycle
54	6	DMA	0 = processor cycle 1 = dma cycle (bus arbitrated)
55	7	*MAP_BYTE	0 = emulation memory access mapped as byte 1 = normal access (This bit is currently unused)
56-58	8-10	IPL0-IPL2	Interrupt Priority Level lines 0-2 from target
59	11		(Unused)
60	12	*HALT	0 = processor halt 1 = processor not halted
61	13	*BERR	0 = bus error 1 = no bus error
62	14	*DATA_LSB	0 = lower byte of data bus not valid 1 = lower byte of data bus valid

Emulation Analyzer Trace Signals			
Analyzer Channel #	Status Bit #	Signal Name	Signal Description
63	15	DATA_MSB	0 = upper byte of data bus not valid 1 = upper byte of data bus valid

State Qualifiers

Whenever a state can be specified in the trace command (trigger state, storage state, prestore state, etc.), you will see the following softkeys that allow you to qualify the state:

address	The value following this softkey is searched for on the lines that monitor the emulation processor's address bus.
data	The value following this softkey is searched for on the lines that monitor the emulation processor's data bus.
status	The value following this softkey is searched for on the lines that monitor other emulation processor signals.

When a value is specified without one of these softkeys it is assumed to be an address value.

Predefined Values for Qualifiers When you specify status qualifiers for analyzer states (by pressing the status softkey), you will be given the following softkeys which are predefined values for the qualifiers.

Qualifier	Status Values	Description
bgd	0xxxx xxxx xxxx xxx0y	emulator in background
berr	0xx01 xxxx xxxx xxxxy	bus cycle bus error
data	0xxxx xxxx xxxx x01xy	bus cycle is a data transfer
ded_int1	0xxxx xxx0 xxxx xxxxy	dedicated mode interrupt 1
ded_int6	0xxxx xx0x xxxx xxxxy	dedicated mode interrupt 6
ded_int7	0xxxx x0xx xxxx xxxxy	dedicated mode interrupt 7
dma	0xxxx xxxx x1xx xxxxy	bus released to DMA device
ext_cyc	0xxxx xxxx xx0x xxxxy	external processor cycle
fetch	0xx11 xxxx xxx1 x10xy	
fgd	0xxxx xxxx xxxx xxx1y	emulator in foreground
halt	0xx10 xxxx xxxx xxxxy	processor halt
hibyte	010xx xxxx xxxx xxxxy	high byte bus transfer
intack	0xxxx xxxx xxxx 111xy	interrupt acknowledge cycle
int_cyc	0xxxx xxxx xx1x xxxxy	internal processor cycle
lobyte	001xx xxxx xxxx xxxxy	low byte bus transfer
map_byte	0xxxx xxxx 0xxx xxxxy	emulation memory address is byte wide
map_word	0xxxx xxxx 1xxx xxxxy	emulation memory address is word wide
no_dma	0xxxx xxxx x0xx xxxxy	bus not released to DMA device
no_intr	0xxxx x111 xxxx xxxxy	no interrupt
nor_int1	0xxxx x110 xxxx xxxxy	normal mode interrupt level 1
nor_int2	0xxxx x101 xxxx xxxxy	normal mode interrupt level 2
nor_int3	0xxxx x100 xxxx xxxxy	normal mode interrupt level 3
nor_int4	0xxxx x011 xxxx xxxxy	normal mode interrupt level 4
nor_int5	0xxxx x010 xxxx xxxxy	normal mode interrupt level 5
nor_int6	0xxxx x001 xxxx xxxxy	normal mode interrupt level 6
nor_int7	0xxxx x000 xxxx xxxxy	normal mode interrupt level 7
prog	0xxxx xxxx xxxx x10xy	program cycle
read	0xxxx xxxx xxx1 xxxxy	bus cycle is a read
retry	0xx00 xxxx xxxx xxxxy	
sup	0xxxx xxxx xxxx 1xxxy	supervisor cycle
supdata	0xxxx xxxx xxxx 101xy	supervisor data cycle
supprog	0xxxx xxxx xxxx 110xy	supervisor program cycle
user	0xxxx xxxx xxxx 0xxxy	user cycle
userdata	0xxxx xxxx xxxx 001xy	user data cycle
userprog	0xxxx xxxx xxxx 010xy	user program cycle
word	011xx xxxx xxxx xxxxy	word cycle
write	0xxxx xxxx xxx0 xxxxy	memory write

These predefined values may be used as other values would be used. For example:

trace after status write

is the same as:

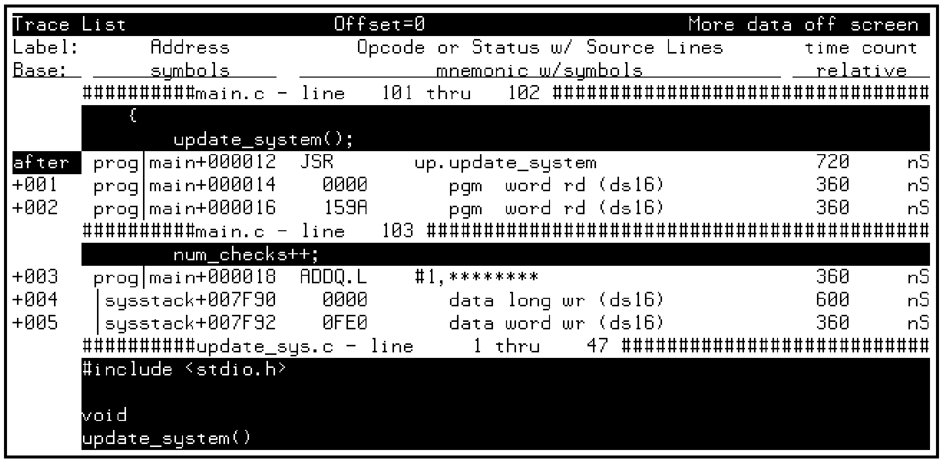
trace after status 0xxxxxxxxxxx0xxxxy

To specify a trigger and set the trigger position

- Enter a trigger state specification in the entry buffer; then, choose **Trace→After ()**, **Trace→About ()**, or **Trace→Before ()**.
- When displaying memory in mnemonic format, position the mouse pointer over the source line where you want to set the trace trigger, press and hold the *select* mouse button and choose **Trace After**, **Trace Before**, or **Trace About** from the pop-up menu.

Or, using the command line, enter:

`trace after`, **`trace about`**, or **`trace before`**



```
Trace List                    Offset=0                        More data off screen
Label:      Address              Opcode or Status w/ Source Lines          time count
Base:        symbols                       mnemonic w/symbols                  relative
      ##########main.c - line   101 thru   102 ###################################
           {
               update_system();
after   prog|main+000012  JSR        up.update_system                       720     nS
+001    prog|main+000014    0000          pgm  word rd (ds16)              360     nS
+002    prog|main+000016    159A          pgm  word rd (ds16)              360     nS
      ##########main.c - line   103 ############################################
               num_checks++;
+003    prog|main+000018  ADDQ.L     #1,********                            360     nS
+004       |sysstack+007F90    0000          data long wr (ds16)            600     nS
+005       |sysstack+007F92    0FE0          data word wr (ds16)            360     nS
      ##########update_sys.c - line     1 thru    47 ###########################
      #include <stdio.h>

      void
      update_system()
```

Tracing after the trigger state says states that occur after the trigger state should be saved; in other words, the trigger is positioned at the top of the trace.

Tracing before the trigger state says states that occur before the trigger state should be saved; in other words, the trigger is positioned at the bottom of the trace.

Tracing about the trigger state says states that occur before and after the trigger state should be saved; in other words, the trigger is positioned at the center of the trace.

When the analyzer counts time or states, the actual trigger position is within +/- 1 state of the number specified. When counts are turned OFF, the actual trigger position is within +/- 3 states of the number specified.

Usually, when you enter a trace about command, the trigger state (line 0) is labeled "about". However, if there are three or fewer states before the trigger, the trigger state is labeled "after". Likewise, if there are 3 or fewer states after the trigger, the trigger state is labeled "before".

The state you define after trace after, trace about, or trace before is the state that will trigger the analyzer and cause states to be stored.

Examples

Suppose you want to look at the execution of the demo program after the call of the "update_system()" function (main.c: line 102) occurs. To trigger on this address, enter:

```
trace after address main."main.c": line 102

set source on inverse_video on symbols on

display trace
```

In the preceding trace list, line 0 (labeled "after") shows the beginning of the program loop.

To use address, data, and status values in trace expressions

- Enter the value(s) desired in the entry buffer (such as **address 1000h**). Then Choose **Trace→After( ), Trace→Before( ),** or **Trace→About( ),** as desired.

Or, using the command line, enter commands as follows:

- To specify an address expression, enter:

```
<expression> -or- address <expression>
```

- To specify a data expression, enter:

```
data <expression>
```

- To specify a status expression, enter:

```
status <expression>
```

Many trace commands require that you enter address, data and status expressions to specify the bus state. You can combine multiple expressions on the same command line to build a complete bus state qualifier. You can also use logical operators to build more complex states. Refer to the "Emulator/Analyzer Interface Commands" chapter for details.

The default expression type is address, therefore you don’t need to specify the address keyword when you enter an address expression.

Example

Start a trace and store only writes of 0 hex to the graph address in the demo program:

```
trace only graph data 0 status write
```

To enter a range in a trace expression

- Use the command line rules (described below) to create your expression in the entry buffer. Then Choose **Trace→After( ),** **Trace→Before( ),** or **Trace→About( ),** as desired.

Or, using the command line, enter commands as follows:

- To specify an address range enter:

```
address range <expression> thru <expression>
```

- To specify a data range, enter:

```
data range <expression> thru <expression>
```

- To specify a status range enter:

```
status range <expression> thru <expression>
```

- To take the logical not of a range, use the not keyword before the range keyword.

Ranges allow you to qualify analyzer actions on a contiguous set of values. Mostly, you'll use address ranges to trigger or store on access to a data block such as a lookup table. But, you can also use data ranges to qualify a trigger or storage on a range of data values.

There is only one range term available in the trace specification. Once it has been used, it cannot be reused. That is, if you specify a range in a trigger specification, you can't duplicate it in the storage specification. (The Terminal Interface does allow this type of measurement, though there is still only one range term. Refer to the Hewlett-Packard *Emulation-Bus Analyzer (with deep trace memory) User's Guide* for details.)

Since address is the default range type, you can omit the address keyword. You can't omit the data or status keywords if those are the bus parts you want to qualify.

You can use the logical or operator to combine the range term with several state qualifiers. See the examples.

Examples

Store only the accesses to the demo program's current_humid location:

```
trace only range current_humid thru +1h
```

Store only bus cycles where data is in the range 6h..26h or is 29h:

```
trace only data range 6h thru 26h or data 29h
```

To trigger on a number of occurrences of some state

- Use the **occurs <#TIMES>** after specifying the trigger state.

When specifying a trigger state, you can include an occurrence count. The occurrence count specifies that the analyzer trigger on the Nth occurrence of some state.

The default base for an occurrence count is decimal. You may specify occurrence counts from 1 to 65535.

Examples

To trigger on the 20th occurrence of the call of the "update_system()" function (main.c: line 102):

```
trace after address main."main.c": line 102
occurs 20
```

To break emulator execution on the analyzer trigger

- Enter a trigger state specification in the entry buffer, then choose **Trace→Until ()**.
- When displaying memory in mnemonic format, position the mouse pointer over the program line which you wish to trace before, press and hold the *select* mouse button and choose **Trace Until** from the pop-up menu.

Or, using the command line, use the break_on_trigger option to the trace command.

The break_on_trigger option to the trace command allows you to cause the emulator to break when the analyzer finds the trigger state.

Note that the actual break may be several cycles after the analyzer trigger.

Examples

To trace before source line 102 and cause the emulator to break into the monitor when the analyzer triggers:

```
trace before address main."main.c": line 102
break_on_trigger
```

To count states or time

- Create your first specification form on the command line. That will enter the proper format in the Trace Specification Selection dialog box. Obtain that dialog box by choosing **Trace→Trace Spec...** You can click on your specification in the dialog box, edit it if desired, and click OK.

Or, using the command line, enter commands as follows:

- To count occurrences of a particular bus state in the trace, enter:

```
trace counting <bus_state>
```

<bus_state> represents a combination of address, data and status expressions that must be matched to satisfy the trigger qualifier.

- To count all states in the trace, enter:

```
trace counting anystate
```

- To count time in the trace, enter:

```
trace counting time
```

- To disable counting in the trace, enter:

```
trace counting off
```

You can use the analyzer's state/time counter to count time or bus states. If using the emulation-bus analyzer with deep memory, counting imposes no restrictions on memory depth. If using the 1K analyzer, use of the counter restricts the trace memory to a maximum depth of 512 states. If you disable the counter in the 1K analyzer, using the trace counting off command, maximum trace depth is 1024 states.

When using the 1K analyzer, the MC6830x emulator defaults to counting off. To count states or time, you must configure the analyzer clocks correctly. See "To configure the analyzer clock" in the "Configuring the Emulator" chapter for more information.

Use the display trace count command to determine how the count is displayed in the trace list. See “To display count information in the trace” for more information.

Examples

To count occurrences of a particular bus state in the trace (this requires the 1K analyzer speed to be set to "Slow" in the configuration):

```
trace counting state address 10h
```

Count all states in the trace:

```
trace counting anystate
```

Count time in the trace:

```
trace counting time
```

Disable counting in the trace:

```
trace counting off
```

To define a storage qualifier

- Enter the storage qualifier (such as **status read**) in the entry buffer. Then choose **Trace→Only().**

Or, using the command line, enter:

```
trace only <bus_state>
```

<bus_state> represents a combination of address, data and status expressions that must be matched to satisfy the storage qualifier.

Storage qualifiers can help filter unwanted information from program execution and improve your trace measurement. The analyzer stores only the information specified in the storage qualifier. Note that if you have a sequencer or trigger specification, any states given there are shown in the trace list even if they don’t meet the storage qualifier.

Examples

Trace only address 10h:

```
trace only address 10h
```

Trace only data value 0ffh:

```
trace only data 0ffh
```

Trace only write operations

```
trace only status write
```

To define a prestore qualifier

- **Place your prestore qualification into the entry buffer, then choose Trace→Only( ) Prestore.**

Using the command line, enter commands as follows:

- Specify a prestore qualifier by entering:

```
trace prestore <bus_state>
```

<bus_state> represents a combination of address, data and status expressions that must be matched to satisfy the prestore qualifier.

- Disable prestore qualification by entering:

```
trace prestore anything
```

You use the prestore qualifier to save states that are related to other routines that you're tracing. For example, you might be tracing a subprogram, and want to see which program called it. You can specify calls be prestored and that entries to the subprogram be stored. The easiest way to do this is to prestore program reads that are outside the address range of the subprogram being called.

You may have several program modules that write to a variable, and sometime during execution of your program, that variable gets bad data written to it. Using a prestore measurement, you can find out which module is writing the bad data. Store-qualify writes to the variable, and uses prestore to capture the instructions that caused those writes to occur (perhaps by prestoring program reads).

Examples

Specify a prestore qualifier:

```
trace prestore address not range gen_ascii_data
thru gen_ascii_data end status program and read
only gen_ascii_data
```

Disable prestore qualification:

```
trace prestore anything
```

To trace activity leading up to a program halt

- Choose **Trace→Until Stop**.
  Or, using the command line, enter:

```
trace on_halt
```

The above commands cause the analyzer to continuously fill the trace buffer until you issue a **Trace→Stop** or ***stop_trace*** command.

Sometimes you may have a program failure that can't be attributed to a specific trigger condition. For example, the emulator may access guarded memory and break to the monitor. You want to trace the events leading up to the guarded memory access but you don't know what to specify for a trigger. Use the above command. The analyzer will capture and record states until the break occurs. The trace list will display the last processor states leading up to the break condition.

Note that the ***trace until stop*** command may not capture the desired information when you are using a foreground monitor (unless the code that causes the break also causes the processor to halt) because the analyzer will continue to capture foreground monitor states after the break. When using a foreground monitor, you can use the command line to enter a trace command that stores only states outside the range of the foreground monitor program (for example, ***trace on_halt only not range <mon_start_addr> thru <mon_end_addr> on_halt***).

To capture a continuous stream of program execution no matter how large your program

The following example can be performed in emulation systems using the emulation-bus analyzer with deep memory (it cannot be done with the 1K analyzer). It shows you how to capture all of the execution of your target program. You may wish to capture target program execution for storage, for future reference, and/or for comparison with execution after making program modifications. The execution of a typical target program will require more memory space than is available in the trace memory of an analyzer. This example shows you how to capture all of your target program execution while excluding unwanted execution of the emulation monitor.

1 Choose **Trace→Display Options ...**, and in the dialog box, enter 0 or the total depth of your deep analyzer trace memory in the entry field beside Unload Depth. Then click OK or Apply. This sets unload depth to maximum.

2 For this measurement, the analyzer will drive trig1 and the emulator will receive trig1 from the trigger bus inside the card cage. The trig1 signal is used to cause the emulator to break to its monitor program shortly before the trace memory is filled. This use of trig1 is not supported in workstation interface commands. Therefore, terminal interface commands (accessible through the pod command feature) must be used. Enter the following commands:

a. Choose **Settings→Pod Command Keyboard**.
b. Enter tgout trig1 -c <states before end of memory>
(trigger output trig1 before trace complete).
c. Enter bc -e trig1 (break conditions enabled on trig1).
d. Click the ***suspend*** softkey.

Note that "tgout trig1 -c <states...>" means generate trig1 as an output when the state that is <states...> before the end of the trace memory is captured in the trace memory; "bc -e trig1" means enable the emulator to break to its monitor program when it receives trig1.

Select a value for <states before end of memory> that allows enough time and/or memory space for the emulator to break to its monitor program before the trace memory is filled. Otherwise, some of your program execution will not be captured in the trace. Many states may be executed before the emulation break occurs, depending on the state of the processor when the trig1 signal arrives. Also, if your program executes critical routines in which interrupts are masked, the occurrence of trig1 may be ignored until the critical routine is completed (when using a foreground monitor).

3 If you are using a foreground monitor, enter the following additional pod commands to prevent the trace memory from capturing monitor execution. The following example commands will obtain this result in some emulators:

a. Choose **Settings→Pod Command Keyboard**.
b. Enter trng addr=<address range occupied by your monitor>
(trigger on range address = <address range>)
where <address range> is expressed as <first addr>..<last addr>.
c. Enter tsto !r (trace store not range).
d. Click the ***suspend*** softkey.

Note that "trng addr=<addr>..<addr>" means define an address range for the analyzer; "tsto !r" means store all trace activity except activity occurring in the defined address range.

4 Start the analyzer trace with the **Trace→Again** command

5 Start your program running using **Execution→Run→from( )**, **from Transfer Address**, or **from Reset**, as appropriate.

The **Trace→Again** (or ***trace again***) command starts the analyzer trace with the most recent trace specifications (including the pod_command specifications you entered). The trace command cannot be used by itself because it defaults the "bc -e trig1", "trng addr=...", and "tsto !r" specifications, returning them to their default values before the trace begins.

You can see the progress of your trace with the command, **Display→Status**. A line in the Trace Status listing will show how many states have been captured.

6 **The notation "trig1 break" usually followed by "Emulation trace complete" will appear on the status line. If "trig1 break" remains on the status line without "Emulation trace complete", manually stop the trace with the command:**

Trace→Stop

You must wait for the notation "trig1 break" and/or "Emulation trace complete" to appear on the status line; this ensures the trace memory is filled during the trace (except for the unfilled space you specified in step 2 above).

Note that when you set a delay specification using tgout -c or tgout -t (trigger output delay before trace complete/after trigger), the trace will indicate complete as soon as the analyzer has captured the state specified, even though the entire trace memory has not been filled.

If the notation "trig1 break" remains on the status line without being replaced by "Emulation trace complete", it indicates the trace memory is not completely filled, and no more states are being captured.

7 Store the entire trace memory content in a file with a command like:

wait *measurement_complete ; copy trace to* <directory/filename>

The "wait" command is inserted ahead of the "copy" command to ensure that the unload of trace data is complete before you try to store it. Without "wait", you will get an ERROR message warning that the unload is still in process. The <filename> is an ASCII filename for a binary file that can be viewed using the load trace command.

8 Start a new trace with the command: **trace again**

9 Resume the program run from the point where it was interrupted when the emulator broke to the monitor with the command ***run***.

10 Wait until the notation "trig1 break" and/or "Emulation trace complete" appears on the status line. Then store the new trace memory content in a new file with commands like:

stop_trace
wait *measurement_complete ; copy trace to* <directory/filename+1>

Note that "filename+1" in the above command suggests use of consecutive filenames to store your execution files, such as FILENAME1, FILENAME2, etc.

Repeat steps 8 through 10 above until all program execution has been captured. Your destination directory will have a set of files that, taken together, contain all of your program execution. Note that if you did not prevent capture of foreground monitor cycles in step 3 above, the last few trace lines in each file may contain monitor cycles.

Using the Sequencer

When you use the analyzer's sequencer, you can specify traces that trigger on a series, or sequence, of states. You can specify a state which, when found, causes the analyzer to restart the search for the sequence of states. Also, the analyzer's sequencer allows you to trace "windows" of code execution.

This section describes how to:

- Trigger after a sequence of states.
- Specify a global restart state.
- Trace "windows" of program execution.
- Specify both sequencing and windowing.

The sequencing and windowing capabilities from within the Softkey Interface are not as powerful or flexible as they are from within the Terminal Interface. For example, in the Terminal Interface, you can specify different restart states for each sequence term and you can set up a windowing trace specification where the trigger does not have to be in the window. If you do not find the sequencing flexibility you need from within Softkey Interface, refer to the *6830x Installation/Service/Terminal Interface User's Guide*.

To trigger after a sequence of states

- Create your first specification form on the command line. That will enter the proper format in the Trace Specification Selection dialog box. Obtain the dialog box by choosing **Trace→Trace Spec...** You can click on your specification in the dialog box, edit it if desired, and click OK.

Or, using the command line, enter:

```
trace find_sequence <bus_state> occurs <#times>
then <bus_state> occurs <#times> trigger
<bus_state>
```

<bus_state> represents a combination of address, data and status expressions that must be matched to satisfy the trigger or sequence qualifier. <#times> is the number of times that bus state must occur to satisfy the qualifier.

The analyzer's sequencer has several levels (also called *sequence terms*). Each state in the series of states to be found before triggering, as well as the trigger state, is associated with a sequence term.

When triggering using the sequencer, the analyzer searches for the state associated with the first sequence term. When that state is captured, the analyzer starts searching for the state associated with the second term, and so on. The last sequence term used is associated with the trigger state. When the trigger state is captured the analyzer is triggered. Up to seven sequence terms and an optional occurrence count for each term are available.

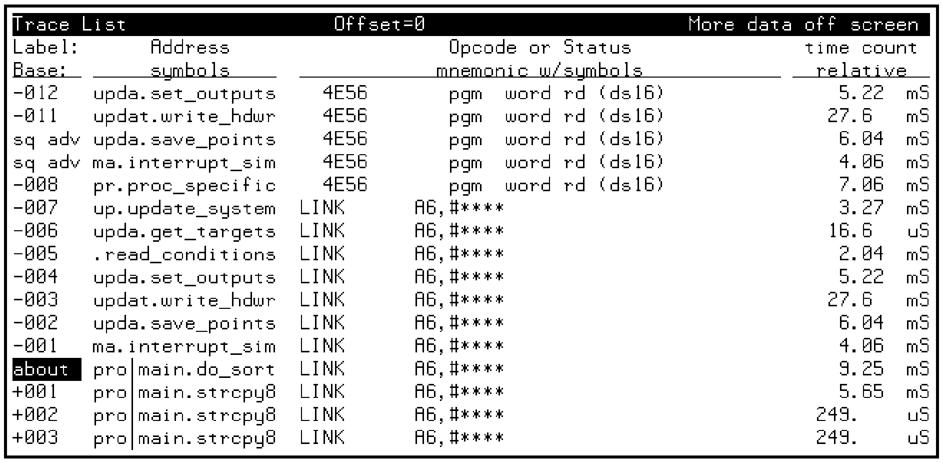
Examples

In the demo program, suppose you wish to trigger on the following sequence of events: the "save_points" function, the "interrupt_sim" function, and finally the "do_sort" function. Also, suppose you wish to store only opcode fetches of the assembly language LINK A6,#0 instruction (data values that equal 4E56H) to show function entry addresses.

To set up the sequencing trace specification, enter the following trace command.

trace find_sequence `save_points` ***then*** `interrupt_sim` ***trigger about*** `do_sort` ***only data*** `4e56h`

set source off



```
Trace List                  Offset=0                          More data off screen
Label:      Address                      Opcode or Status                time count
Base:       symbols                    mnemonic w/symbols                 relative
-012   upda.set_outputs    4E56       pgm  word rd (ds16)                 5.22  mS
-011   updat.write_hdwr    4E56       pgm  word rd (ds16)                27.6   mS
sq adv upda.save_points    4E56       pgm  word rd (ds16)                 6.04  mS
sq adv ma.interrupt_sim    4E56       pgm  word rd (ds16)                 4.06  mS
-008   pr.proc_specific    4E56       pgm  word rd (ds16)                 7.06  mS
-007   up.update_system  LINK      A6,#****                               3.27  mS
-006   upda.get_targets  LINK      A6,#****                              16.6   uS
-005   .read_conditions  LINK      A6,#****                               2.04  mS
-004   upda.set_outputs  LINK      A6,#****                               5.22  mS
-003   updat.write_hdwr  LINK      A6,#****                              27.6   mS
-002   upda.save_points  LINK      A6,#****                               6.04  mS
-001   ma.interrupt_sim  LINK      A6,#****                               4.06  mS
about   pro|main.do_sort  LINK      A6,#****                               9.25  mS
+001    pro|main.strcpy8  LINK      A6,#****                               5.65  mS
+002    pro|main.strcpy8  LINK      A6,#****                             249.    uS
+003    pro|main.strcpy8  LINK      A6,#****                             249.    uS
```

Notice the states that contain "sq adv" in the first column (you may have to press <PREV> in order to see the states captured prior to the trigger). These are the states associated with (or captured for) each sequence term. Just as the trigger state is always stored in trace memory, the states captured in the sequence are always stored if the trace buffer is deep enough.

To specify a global restart state

- Create your first specification form on the command line. That will enter the proper format in the Trace Specification Selection dialog box. Obtain the dialog box by choosing **Trace→Trace Spec...** You can click on your specification in the dialog box, edit it if desired, and click OK.

Or, using the command line, enter:

```
trace find_sequence <bus_state> occurs <#times>
then <bus_state> occurs <#times> restart
<bus_state> trigger <bus_state>
```

<bus_state> represents a combination of address, data and status expressions that must be matched to satisfy the trigger or sequence qualifier. <#times> is the number of times the selected bus state must occur to satisfy the qualifier.

The restart qualifier allows you to restart the trace sequence whenever a certain instruction or data access occurs. For example, you might have a complicated trace sequence that searches for an intermittent failure condition. You could set the restart term to restart the sequence whenever a bus cycle occurred that ensures that the code segment would perform correctly. Thus, the trace will be satisfied only when that restart term never occurs and the code segment fails.

Examples

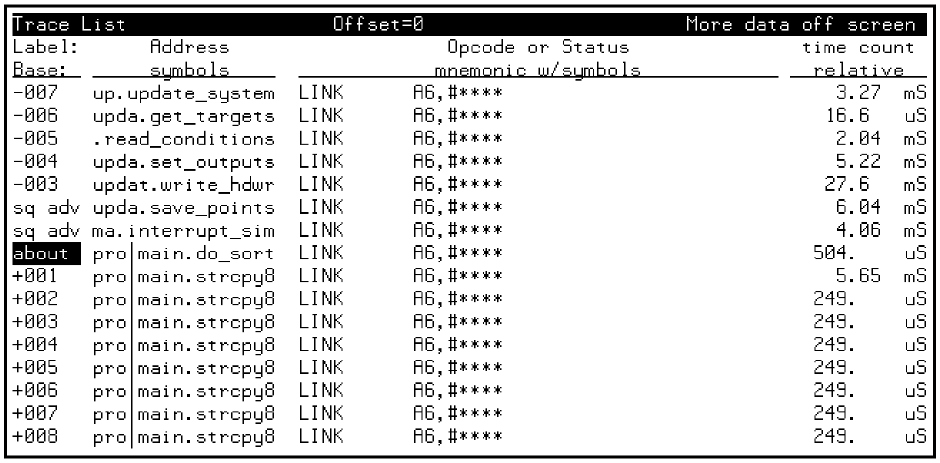
In the demo program, suppose you wish to trigger on the following sequence of events: the "save_points" function, the "interrupt_sim" function, and the "do_sort" function. However, you only want to trigger when the "interrupt_sim" calls the "do_sort" function. In other words, if the "proc_specific" function is entered before the "do_sort" function is entered, you know "interrupt_sim" did not call "do_sort" this time, and the analyzer should start searching again from the beginning.

Again, suppose you wish to store only opcode fetches of the assembly language LINK A6,#0 instruction (data values that equal 4E56H).

To set up this sequencing trace specification, enter the following trace command.

trace find_sequence `save_points` ***then*** `interrupt_sim` ***restart*** `proc_specific` ***trigger about*** `do_sort` ***only data*** `4e56h`

set source off



```
Trace List                        Offset=0                         More data off screen
Label:        Address                  Opcode or Status                      time count
Base:     symbols                  mnemonic w/symbols                       relative
-007     up.update_system  LINK      A6,#****                                3.27  mS
-006     upda.get_targets  LINK      A6,#****                               16.6   uS
-005     .read_conditions  LINK      A6,#****                                2.04  mS
-004     upda.set_outputs  LINK      A6,#****                                5.22  mS
-003     updat.write_hdwr  LINK      A6,#****                               27.6   mS
sq adv   upda.save_points  LINK      A6,#****                                6.04  mS
sq adv   ma.interrupt_sim  LINK      A6,#****                                4.06  mS
about    pro|main.do_sort  LINK      A6,#****                              504.    uS
+001     pro|main.strcpy8  LINK      A6,#****                                5.65  mS
+002     pro|main.strcpy8  LINK      A6,#****                              249.    uS
+003     pro|main.strcpy8  LINK      A6,#****                              249.    uS
+004     pro|main.strcpy8  LINK      A6,#****                              249.    uS
+005     pro|main.strcpy8  LINK      A6,#****                              249.    uS
+006     pro|main.strcpy8  LINK      A6,#****                              249.    uS
+007     pro|main.strcpy8  LINK      A6,#****                              249.    uS
+008     pro|main.strcpy8  LINK      A6,#****                              249.    uS
```

Notice in the preceding trace (you may have to press <PREV> in order to see the states captured prior to the trigger) that, in addition to states captured in the sequence, "sq adv" is also shown next to states which cause a sequencer restart.

To trace "windows" of program execution

- Create your first specification form on the command line. That will enter the proper format in the Trace Specification Selection dialog box. Obtain the dialog box by choosing **Trace→Trace Spec...** You can click on your specification in the dialog box, edit it if desired, and click OK.

Or, using the command line, enter commands as follows:

- To trace only the states occurring after a particular bus cycle, enter:

trace enable `<bus_state>`

- To trace only the states occurring between two particular bus cycles, enter:

trace enable `<bus_state>` ***disable*** `<bus_state>`

<bus_state> represents a combination of address, data and status expressions that must be matched to satisfy the windowing qualifier.

Windowing refers to the analyzer feature that allows you to turn on, or enable, the capturing of states after some state occurs then to turn off, or disable, the capturing of states when another state occurs. In effect, windowing allows you capture "windows" of code execution.

Windowing is different than storing states in a range (the only range option in the trace command syntax) because it allows you to capture execution of all states in a window of code whereas storing states in a range won't capture the execution of subroutines that are called in that range or reads and writes to locations outside that range.

When you use the windowing feature of the analyzer, the trigger state must be in the window or else the trigger will never be found.

If you wish to combine the windowing and sequencing functions of the analyzer, note the following restrictions:

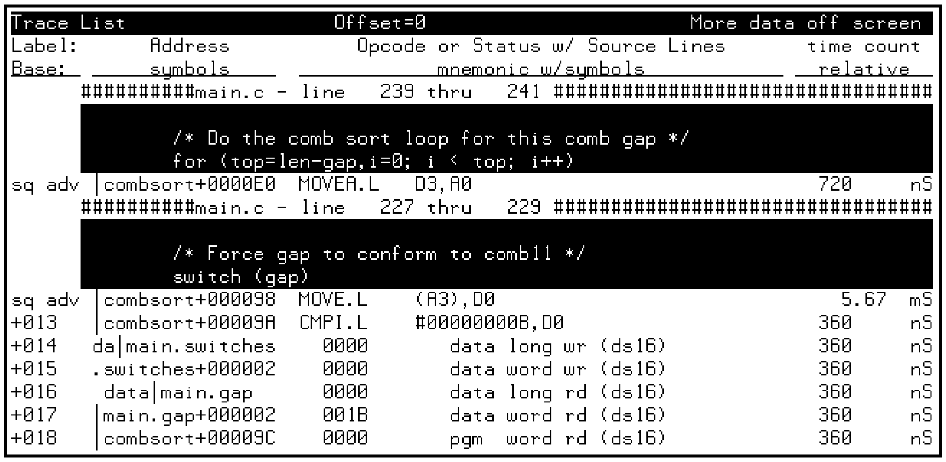
- Up to four sequence terms are available when windowing is in effect.
- Global restart is not available when windowing is in effect.
- Occurrence counts are not available.

Examples

In the demo program, suppose you are only interested in the execution that occurs within the switch statement of the "combsort" function. You could specify source line number 229 as the window enable state and the source line number of the next statement (line number 241) as the window disable state. Set up the windowing trace specification with the following command.

trace enable `main."main.c": line 229` ***disable*** `main."main.c": line 241`

set source on



```
Trace List                     Offset=0                        More data off screen
Label:      Address          Opcode or Status w/ Source Lines        time count
Base:       symbols                 mnemonic w/symbols                 relative
      ##########main.c - line   239 thru   241 ##################################

              /* Do the comb sort loop for this comb gap */
              for (top=len-gap,i=0; i < top; i++)
sq adv |combsort+0000E0  MOVEA.L   D3,A0                              720     nS
      ##########main.c - line   227 thru   229 ##################################

              /* Force gap to conform to comb11 */
              switch (gap)
sq adv |combsort+000098  MOVE.L    (A3),D0                             5.67  mS
+013   |combsort+00009A  CMPI.L    #0000000B,D0                       360     nS
+014  da|main.switches    0000       data long wr (ds16)              360     nS
+015  .switches+000002    0000       data word wr (ds16)              360     nS
+016   data|main.gap      0000       data long rd (ds16)              360     nS
+017   |main.gap+000002   001B       data word rd (ds16)              360     nS
+018   |combsort+00009C   0000       pgm  word rd (ds16)              360     nS
```

Notice in the resulting trace (you have to press the <NEXT> key) that the enable and disable states have the "sq adv" string in the line number column. This is because the windowing feature uses the analyzer's sequencer.

To specify both sequencing and windowing

- Create your first specification form on the command line. That will enter the proper format in the Trace Specification Selection dialog box. Obtain that dialog box by choosing **Trace→Trace Spec...** You can click on your specification in the dialog box, edit it if desired, and click OK.

Or, using the command line, enter:

```
trace enable <bus_state> disable <bus_state>
find_sequence <bus_state> then <bus_state>
trigger <bus_state>
```

<bus_state> represents a combination of address, data and status expressions that must be matched to satisfy the trigger or sequence qualifier. <#times> is the number of times that bus state must occur to satisfy the qualifier.

You can use the sequencing and windowing specifications together to make specification of complex qualifiers easier. If you use the windowing specification, the sequence specification is limited to four sequence terms. Also, note that when you use a windowing specification, you cannot use a restart term with your sequence specification.

Example

Use the analyzer sequencer to trace states occurring between the start of the example program and the call to the message interpreter, then trigger after access to the variable that stores the value of current humidity, but only if it is accessed after a specific series of events:

```
trace enable main disable proc_spec find_sequence
update_sys.get_targets then update_sys.write_hdwr
trigger after current_humid
```

Displaying the Trace List

The trace list is your view of the analyzer's record of processor bus activity. You can specify what is shown in the trace list to make it easier to find the information of interest. For example, you can display symbol information where available, or source lines from the high-level languages used to write the target system program. You can also change column widths and set options for trace list disassembly.

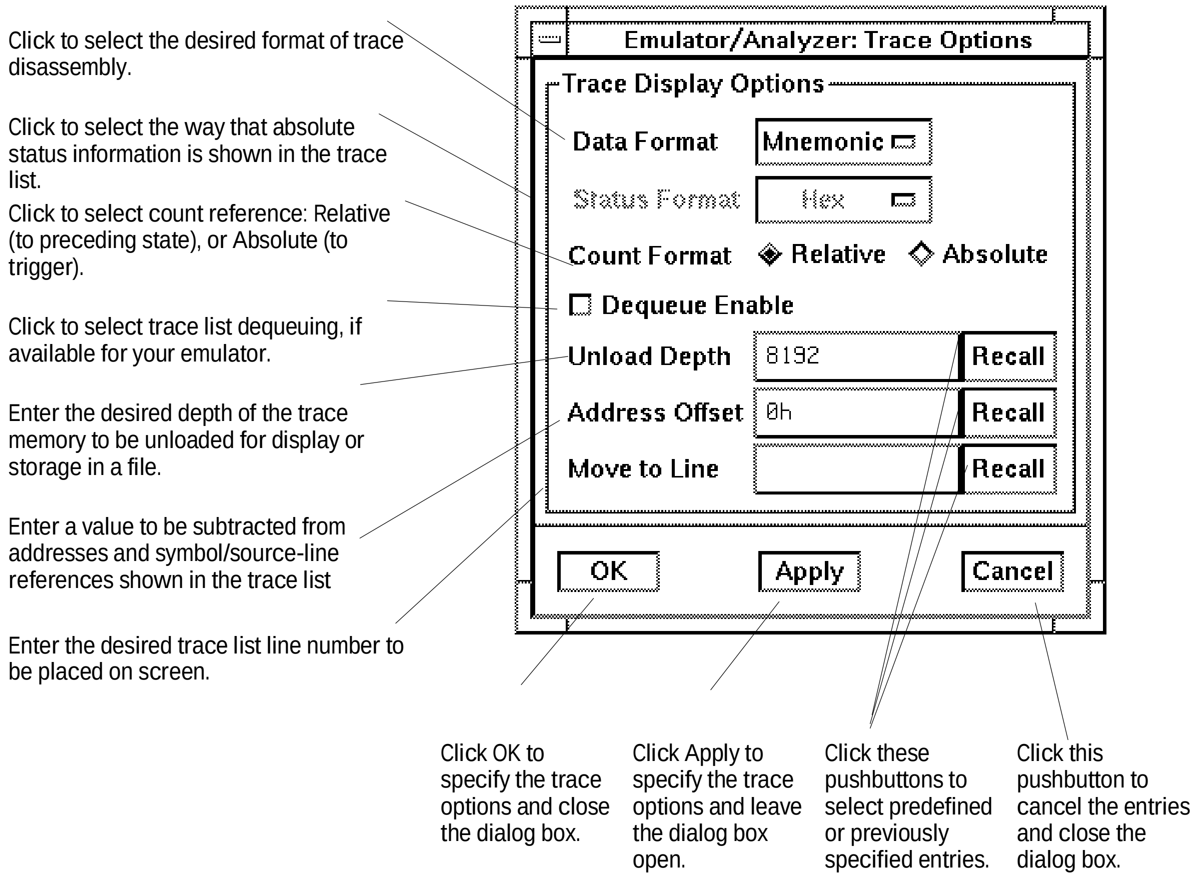
Display control is available through the **Trace→Display Options...** dialog box, the trace list pop-up menu, and the command line. You can combine most options within a single command line command to obtain a desired trace display. See the display trace and set command descriptions in the "Emulator/Analyzer Interface Commands" chapter for more information. This section describes how to:

- Use the Trace Options dialog box.
- Use the trace list pop-up menu.
- Display the trace about a line number.
- Move through the trace list.
- Disassemble the trace list.
- Specify trace disassembly options.
- Specify trace dequeueing options.
- Display the trace without disassembly.
- Display symbols in the trace list.
- Display source lines in the trace list.
- Change the column width.
- Select the type of count information in the trace list.
- Offset addresses in the trace list.
- Reset the trace display defaults.
- Change the number of states available for display.
- Display program memory associated with a trace list line.

- Open an edit window into the source file associated with a trace list line.

Examples

To use the Trace Options dialog box:


Click to select the desired format of trace disassembly.
Click to select the way that absolute status information is shown in the trace list.
Click to select count reference: Relative (to preceding state), or Absolute (to trigger).
Click to select trace list dequeuing, if available for your emulator.
Enter the desired depth of the trace memory to be unloaded for display or storage in a file.
Enter a value to be subtracted from addresses and symbol/source-line references shown in the trace list
Enter the desired trace list line number to be placed on screen.
Click OK to specify the trace options and close the dialog box.
Click Apply to specify the trace options and leave the dialog box open.
Click these pushbuttons to select predefined or previously specified entries.
Click this pushbutton to cancel the entries and close the dialog box.

Emulator/Analyzer: Trace Options

Trace Display Options

Data Format: Mnemonic

Status Format: Hex

Count Format: ◆ Relative ◇ Absolute

☐ Dequeue Enable

Unload Depth: 8192 Recall

Address Offset: 0h Recall

Move to Line: Recall

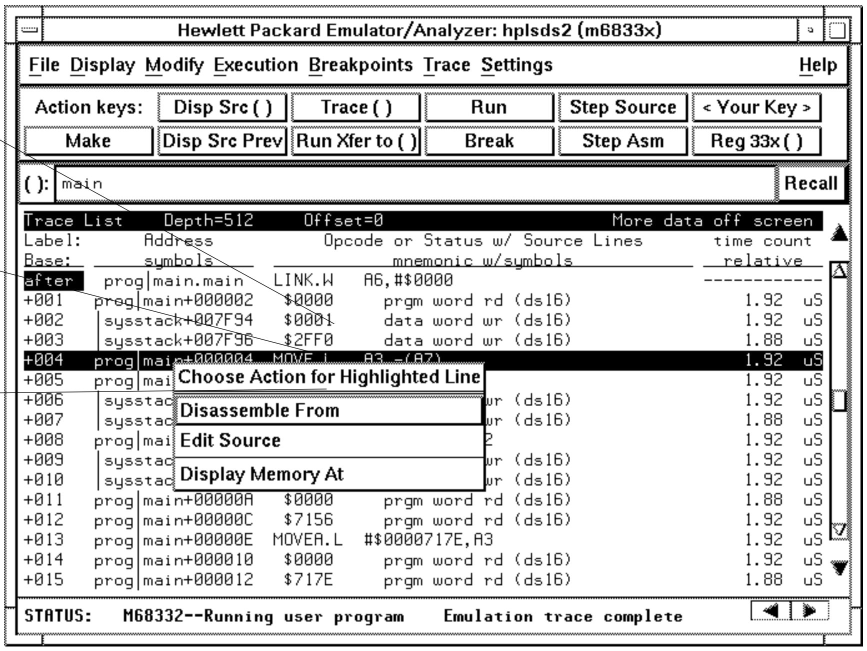
OK Apply Cancel

To use the trace list pop-up menu:

Click to begin trace disassembly from the selected line, moving that line to the top of the display.

Click to open an edit window into the source file that contains the address of the selected line.

Click to open a display window into memory containing the address of the selected line. Note that the format of the memory display will be mnemonic for addresses in the code segment and absolute otherwise.


Hewlett Packard Emulator/Analyzer: hplsds2 (m6833x)
File Display Modify Execution Breakpoints Trace Settings Help
Action keys: Disp Src ( ) Trace ( ) Run Step Source < Your Key >
Make Disp Src Prev Run Xfer to ( ) Break Step Asm Reg 33x ( )
( ): main Recall
Choose Action for Highlighted Line
Disassemble From
Edit Source
Display Memory At

```
Trace List     Depth=512      Offset=0               More data off screen
Label:       Address          Opcode or Status w/ Source Lines        time count
Base:        symbols                 mnemonic w/symbols                 relative
after    prog|main.main   LINK.W   A6,#$0000                       ------------
+001    prog|main+000002   $0000     prgm word rd (ds16)                1.92  uS
+002     |sysstack+007F94  $0001     data word wr (ds16)                1.92  uS
+003     |sysstack+007F96  $2FF0     data word wr (ds16)                1.88  uS
+004    prog|main+000004  MOVE.L   A3,-(A7)                            1.92  uS
+005    prog|mai                                                        1.92  uS
+006     |sysstac                  wr (ds16)                           1.92  uS
+007     |sysstac                  wr (ds16)                           1.88  uS
+008    prog|mai                  2                                   1.92  uS
+009     |sysstac                  wr (ds16)                           1.92  uS
+010     |sysstac                  wr (ds16)                           1.92  uS
+011    prog|main+00000A   $0000     prgm word rd (ds16)                1.88  uS
+012    prog|main+00000C   $7156     prgm word rd (ds16)                1.92  uS
+013    prog|main+00000E  MOVEA.L  #$0000717E,A3                       1.92  uS
+014    prog|main+000010   $0000     prgm word rd (ds16)                1.92  uS
+015    prog|main+000012   $717E     prgm word rd (ds16)                1.88  uS
STATUS:   M68332--Running user program    Emulation trace complete
```

To display the trace about a line number

- Choose **Trace→Display Options...** and in the dialog box, enter the desired trace list line number in the field beside Move to Line. Then click the OK or Apply pushbutton.

Or, using the command line, enter:

```
display trace <LINE #>
```

If you need to move to a particular state quickly, you can use this command. The command places the specified state in the center of the current trace display.

Examples

Display the trace about line number 20:

Choose **Trace→Display Options...** and in the dialog box, enter 20 in the field beside Move to Line. Then click the OK or Apply pushbutton.

Enter the following command on the command line to display the trace about line number 256:

```
display trace 256
```

To move through the trace list

- **Use the scroll bar at the right of the display to scroll up and down. Use the arrows at the bottom of the display (if any) to scroll left and right.**

Using the command line, enter commands as follows:

- To roll the trace display to the left, press <Ctrl>f simultaneously.
- To roll the trace display to the right, press <Ctrl>g simultaneously.
- To roll the display down one line, press the down arrow key.
- To roll the display up one line, press the up arrow key.
- To move to the previous page in the trace list, press the Pg Up or Prev key.
- To move to the next page in the trace list, press the Pg Dn or Next key.

Though the trace display is set to 256 or more states, only 15 lines may be displayed in the interface window, depending on your terminal type. You can move through the trace list display using various key combinations.

You can roll the display left and right only if the trace list is wider than 80 columns. This may occur if you increased the width of the columns.

To disassemble the trace list

- Choose **Trace→Display Options...** and in the dialog box, select Data Format Mnemonic. Then click the OK or Apply pushbutton.
- Use the mouse to place the cursor on a line in the trace list where you want disassembly to begin. Then press the *select* mouse button, and click on **Disassemble From** in the trace list pop-up menu.

Or, using the command line, enter commands as follows:

- To disassemble instruction data in the trace list, enter:

```
display trace mnemonic
```

- To control where trace list disassembly starts, enter:

```
display trace disassemble_from_line_number
<LINE #>
```

<LINE #> is a line number corresponding to a state in the trace list.

Disassembly of instruction data means that you will see instructions as they would appear in an assembly language program listing. That is, instruction mnemonics and operands are shown instead of hexadecimal instruction data.

The analyzer interface normally disassembles instruction data in the trace list. However, if you specify absolute data display, that mode remains in effect until you select the mnemonic option.

When you identify a particular trace list line where disassembly is to begin, be sure to specify a line number that corresponds to an analyzer state with an opcode fetch. The analyzer interface disassembles and displays the trace starting with the state you specify.

Examples

To disassemble instruction data in the trace list starting at line 40:

Place the cursor on line 40, press the select mouse button, and click on Disassemble From in the pop-up menu.

Or, using the command line, enter:

```
display trace disassemble_from_line_number 40
```

To specify trace disassembly options

- Selection of disassembly options is not supported in pull-downs of the Graphical User Interface. By default, the Graphical User Interface selects **high_word** and **all_cycles**. Use the command line if you need to specify trace disassembly using other options.

Or, using the command line, enter commands as follows:

- To show only instruction cycles in the trace list, enter:

```
display trace disassemble_from_line_number
<LINE#> instructions_only
```

- To show all bus cycles in the trace list, enter:

```
display trace disassemble_from_line_number
<LINE#> all_cycles
```

Normally, the MC6830x presents the trace list data as it was stored by the analyzer. That is, all bus cycles are shown, and disassembly starts with the most significant word of the data.

If you don't want to see operand cycles in the trace list, specify the instructions_only option.

The disassembly options remain in effect until you specify a new disassembly option.

Note that although the "high_word," "low_word" and "align" options are displayed, these are for 32-bit processors only, and will not perform any action.

Examples

Show only instruction cycles in the trace list starting at line 40:

```
display trace disassemble_from_line_number 40
instructions_only
```

Show all bus cycles in the trace list:

```
display trace disassemble_from_line_number 40
all_cycles
```

To specify trace dequeueing options

- **Choose Trace→Display Options... and in the dialog box, select Dequeue Enable. Then click the OK or Apply pushbutton.**

Or, using the command line, enter commands as follows:

- To dequeue the trace list, enter:

```
display trace dequeue on
```

- To display the trace list without dequeueing, enter:

```
display trace dequeue off
```

<LINE #> is a line number corresponding to a state in the trace list. <STATE#> is the line number of the data operand that is associated with the instruction at <LINE#>.

A dequeued trace list is available through the disassembly options. In a dequeued trace list, unused instruction prefetch cycles are discarded, and operand cycles are placed immediately following the corresponding instruction fetch. If you choose a non-dequeued trace list, instruction and operand fetches are shown exactly as captured by the analyzer.

Once the dequeuer has been started on the correct opcode, it will continue to disassemble correctly unless an unusual condition causes it to misinterpret the data. By specifying the first instruction state for disassembly and the number of the first operand cycle for that instruction, you can resynchronize the disassembly.

You may see TAKEN, NOT TAKEN, or ?TAKEN? beside a branch in your dequeued trace list. TAKEN is shown beside a branch if the dequeuer determines that the branch was taken. NOT TAKEN is shown if the dequeuer determines that the branch was definitely not taken. ?TAKEN? means the dequeuer was not able to determine whether or not the branch was taken. If you read down the trace list and see that the branch was taken, use the ***disassemble_from_line_number*** command to restart disassembly at the trace list line number of the branch destination.

Examples

Dequeue the trace list:

Choose **Trace→Display Options...** and in the dialog box, select **Dequeue Enable**. Then click the OK or Apply pushbutton.

Or, using the command line, enter:

```
display trace dequeue on
```

Display the trace list without dequeueing:

```
display trace dequeue off
```

To display the trace without disassembly

- Choose **Trace→Display Options...** and in the dialog box, select Data Format Absolute. You can select Hex, Binary, or Mnemonic format for display of status information. Then click the OK or Apply pushbutton.

Or, using the command line, enter commands as follows:

- To display the trace list without instruction disassembly and with status information in binary format, enter:

```
display trace absolute status binary
```

- To display the trace list without instruction disassembly and with status information in hexadecimal format, enter:

```
display trace absolute status hex
```

- To display the trace list without instruction disassembly and with status information in mnemonic format, enter:

```
display trace absolute status mnemonic
```

For some measurements, it may be more convenient for you to view the trace data without instruction disassembly. The **Data Format Absolute** selection in the **Trace→Display Options...** dialog box, or the ***display trace absolute*** command allows you to do this. Notice that once you enter this format selection, subsequent trace lists will displayed in this format until you select the mnemonic format with the dialog box or display trace mnemonic command again.

You can select the display format for the status information when you choose **Data Format Absolute** in the dialog box, or when you use the ***display trace absolute*** command. The status information can be displayed in binary, hex, or as mnemonics that indicate the nature of the current bus cycle (such as a read or write).

Examples

Display the trace list without instruction disassembly and with status information in binary format:

Choose **Trace→Display Options...** and in the dialog box, select **Data Format Absolute**. Select **Status Format Binary**. Then click the OK or Apply pushbutton.

Or, using the command line, enter:

```
display trace absolute status binary
```

Display the trace list without instruction disassembly and with status information in hexadecimal format, make appropriate entries in the **Trace→Display Options...** dialog box, or enter the following command:

```
display trace absolute status hex
```

Display the trace list without instruction disassembly and with status information in mnemonic format, make appropriate entries in the **Trace→Display Options...** dialog box, or enter the following command:

```
display trace absolute status mnemonic
```

To display symbols in the trace list

- Choose **Settings→Source/Symbol Modes→Symbols**, or choose **Settings→Display Modes ...,** and in the dialog box, click on **Symbolic Addresses**. In the Field Widths area of the dialog box, you can select the widths of the Label Field and Symbols in Mnemonic Field to control the display space allocated to the symbols. To select symbol types, use the command line, described below.

Or, using the command line, enter commands as follows:

- To display symbols in the trace list, enter:

```
set symbols on
```

- To display only high level symbols, enter:

```
set symbols high
```

- To display only low level symbols, enter:

```
set symbols low
```

- To display all symbols (both high and low level), enter:

```
set symbols all
```

When you enable symbol display, addresses and operands are replaced by the symbols that correspond to those values. The symbol information is derived from the SRU symbol database for that command file. See Chapter 6, "Using the Emulator" for more information on SRU and symbol handling.

High-level symbols are those that are available only from high-level languages such as a compiler. Low-level symbols are those that are available from assembly language modules (which may include symbols generated internally by a compiler).

The **Settings→Source/Symbol Modes...**, **Settings→Display Modes...,** or ***set symbols*** command remains in effect until you enter a new **Settings→Source/Symbol Modes...**, **Settings→Display Modes...**, or ***set symbols*** command with different options.

Refer to Chapter 6, "Using the Emulator", for details of how to set up and use the Display Modes dialog box.

To display source lines in the trace list

- Choose **Settings→Source/Symbol Modes→Source Mixed** or **Settings→Source/Symbol Modes→Source Only.**
- Choose **Settings→Display Modes...,** and in the dialog box, click on **Source in Trace** and select either **Source Mixed** or **Source Only** from the submenu.

Or, using the command line, enter commands as follows:

- To display mixed source and assembly language in the trace list, enter:

```
set source on
```

- To display only source language statements in the trace list, enter:

```
set source only
```

- To display only assembly language in the trace list, enter:

```
set source off
```

If you developed your target programs in a high-level language such as “C,” you can display the source code in the trace list with the corresponding assembly language statements. Or, you can choose to display only the source listing without the assembly language information.

The analyzer uses the line-number information in the SRU symbol database for the absolute file to reference between source lines and assembly language information. Refer to Chapter 6, "Using the Emulator" for more information on SRU and symbol handling.

To change the column width

- **Choose Settings→Display Modes..., and select desired widths for information in the trace list by using the dialog box. Refer to the "Examples" page under "To display symbols in the trace list", earlier in this chapter for details of how to use the dialog box.**

Or, using the command line, enter commands as follows:

- To set the column width for the address column in the trace list, enter:

```
set width label <WIDTH>
```

- To set the column width for the mnemonic column in the trace list, enter:

```
set width mnemonic <WIDTH>
```

- To set the column width for source lines in the trace list, enter:

```
set width source <WIDTH>
```

<WIDTH> is an integer between 1 and 80, specifying the width of the column in characters. (<WIDTH> is restricted to certain values which are shown if you press the <WIDTH> softkey.)

You can display more information by widening a column or ignore the information by narrowing the column. For example, you might want to widen the label column so that you can see the complete names of the symbols in that column.

You can combine multiple options on the command line to set the width for several columns at once.

Example

Set the width of the address label column to 30 characters and the width of the mnemonic column to 50 characters:

```
set width label 30 mnemonic 50
```

To select the type of count information in the trace list

- Choose **Trace→Display Options...** and in the dialog box, select Count Format Relative or Absolute, as desired. Then click the OK or Apply pushbutton.

Or, using the command line, enter commands as follows:

- To display count information in the trace list relative to the trigger state, enter:

```
display trace count absolute
```

- To display count information in the trace list relative to the previous trace list state, enter:

```
display trace count relative
```

Count information may be displayed two ways: relative (which is the default), or absolute. When relative is selected, count information is displayed relative to the previous state. When absolute is selected, count information is displayed relative to the trigger condition.

The count information in the trace list is always displayed if count display is turned on. To turn on the trace counting function, enter a command beginning with trace counting on the command line. Refer to "To count states or time" later in this manual for details.

When using the 1K analyzer, the trace memory is 512 states deep if counting states or time is turned on and 1024 states deep if counting is turned off. To disable counting in the 1K analyzer, use the command trace counting off. When using the emulation-bus analyzer with deep memory, full memory depth is always available; the depth of the deep analyzer is not affected by the counting selected.   See “To count states or time.”

Examples

Count time and store only each iteration of the update_sys symbol in the demo program (if using the 1K analyzer, make sure the clock speed is set to "Slow" in the configuration):

Specify the trace for the emulator:

```
trace only update_sys counting time
```

Now, start the program run; then display the trace:

```
run from transfer_address
```

```
display trace count relative
```

Count absolute entries into the get_targets routine of the demo program:

```
trace only address range update_sys thru
update_sys end counting state get_targets
```

```
run from transfer_address
```

```
display trace count absolute
```

To offset addresses in the trace list

- **Choose Trace→Display Options... and in the dialog box, enter the desired offset value in the field beside Address Offset. Then click the OK or Apply pushbutton.**

Or, using the command line, use the offset_by command line option to the display trace command.

The Address Offset or offset_by trace display options allow you to cause the address information in the trace display to be offset by the amount specified. The offset value is subtracted from the instruction's physical address to yield the address that is displayed.

If code gets relocated and therefore makes symbolic information invalid, you can use the Address Offset or offset_by option to change the address information so that it again agrees with the symbolic information.

You can also specify an offset to cause the listed addresses to match the addresses in compiler or assembler listings.

Example

Trace execution from entry of the demo program (the main label) then offset by the value of main so that the addresses appear the same as the location counter in the assembler listing:

```
reset
trace
run from transfer_address
display trace offset_by main
```

To reset the trace display defaults

- Choose **Settings→Display Modes...** Then in the dialog box, click on Default All Settings, and click the OK pushbutton. This leaves the trace display in the "source intermixed and symbols on" mode.

Or, using the command line, enter:

```
set default
```

This turns off all symbolics and source references in the interface.

To change the number of states available for display

- Choose **Trace→Display Options...** and in the dialog box, enter the desired number of states to be made available for display in the field beside Unload Depth. Then click the OK or Apply pushbutton.

Or, using the command line, enter:

```
display trace depth <DEPTH#>
```

<DEPTH#> is the number of states to be available in the trace list for displaying, copying, or storing to a file. If you are using the emulation-bus analyzer with deep memory, the depth of the trace list buffer depends on whether or not you installed memory modules on the analyzer card, and the capacity of the memory modules installed. Refer to the Hewlett-Packard *Emulator-Bus Analyzer (with deep memory) User's Guide* for details. If you are using the 1K analyzer, the trace list buffer is 512 or 1024 states deep (depending on whether or not you turn on the state/time count). See "To count states or time" in this chapter.)

When you display the trace list, the interface requests the number of states specified by the trace depth from the emulator. If you want faster trace display, you can decrease the trace depth. To display more states, you can increase the trace depth. Notice that the trace depth setting only regulates the number of states sent from the emulation-bus analyzer to the interface. You still need to use the Pg Up and Pg Dn keys to page through the trace list.

Examples

Set the depth of the trace memory to 256 states:

Choose **Trace→Display Options...** and in the dialog box, enter 256 in the field beside Unload Depth. Then click the OK or Apply pushbutton.

Set the depth of the trace to 1024 states:

```
display trace depth 1024
```

To display program memory associated with a trace list line

- Using the mouse, place the cursor on the line in the trace list where you want to see the associated content of program memory. Then press the *select* mouse button, and click on **Display Memory At** in the trace list pop-up menu.

You will see a display of memory at the location of the program that emitted the selected trace list line. This is the same as placing the program address of the selected trace list line in the entry buffer and choosing **Display→Memory→At()** in the pull-down menus.

To open an edit window into the source file associated with a trace list line

- Using the mouse, place the cursor on the line in the trace list whose source file you wish to edit. Then press the *select* mouse button, and click on **Edit Source** in the trace list pop-up menu.

A new window will open. It will show the source file that emitted the line you selected in the trace list. An edit session will be in progress on the source file in the new window. When you complete the desired edit, save the file and close the window.

Saving and Restoring Trace Data and Specifications

The emulator/analyzer can save trace data and trace specifications in a file for later use. This can help you record measurement results to use for comparison with other tests, and to automate measurements.

Suppose you're using the emulator in a manufacturing test application. The target system is your product board. You might build a command file that recalls a trace specification, makes the trace on the target board, and then recalls a previous measurement result (from a working product) and compares it to the new measurement (using the UNIX diff command).

This section describes how to:

- Store a trace specification.
- Store trace data.
- Load a trace specification.
- Load trace data.

To store a trace specification

- Choose **File→Store→Trace Spec...** In the dialog box, select an existing filename or specify a new filename to contain the present trace specification. Then click OK.
- Using the command line, store the current trace specification by entering:

```
store trace_spec <filename>
```

<filename> is any UNIX file name including paths. The extension .TS is automatically added to the file name. The trace specification file is a binary file.

The ***store trace_spec*** command allows you to save a trace specification (effectively the current trace command with all trigger, storage and sequence options) in a file for later use. For example, you might have several trace commands that you want to make every time your target system program is modified. You can store each trace command in a separate file and recall it later using the ***load trace_spec*** command.

Example

Store a trace specification to a file:

```
store trace_spec tspec.TS
```

To store trace data

- Choose **File→Store→Trace Data...** In the dialog box, select an existing filename or specify a new filename to contain the present trace memory content, then click OK.

Or, using the command line, enter:

```
store trace <filename>
```

<filename> is any UNIX file name including paths. The trace data file is a binary file. The extension .TR is automatically added to the file name. A trace data file can be reloaded and displayed like other trace listings.

You can store the trace data resulting from a measurement. This can be useful if you want to compare the results of later measurements with a reference result obtained in an earlier measurement.

Example

Store a trace to a file:

```
store trace trace1.TR
```

To load a trace specification

- Choose **File→Load→Trace Spec...** In the dialog box, click on the name of the trace specification you want to load (placing it in the Load Trace Specification box), then click OK.

Or, using the command line, enter:

```
load trace_spec <filename>
```

<filename> is any UNIX file name including paths. The extension .TS is assumed.

Once you save a trace specification in a file using the **File→Store→Trace Spec...** or ***store trace_spec*** command, you can load it using the appropriate command above. To start a trace with the trace specification that you loaded, use the **Trace→Again** or ***trace again*** command.

Example

Load a trace specification from a file and start the trace:

```
load trace_spec tspec

trace again
```

To load trace data

- Choose **File→Load→Trace Data...** In the dialog box, click on the name of the trace data file (file of trace memory content) you want to load (placing it in the Load Trace Data box). Then click OK.

Or, using the command line, enter:

```
load trace <filename>
```

<filename> is any UNIX file name including paths. The extension .TR is assumed.

Loads a previously saved trace from a binary trace data file (with a ".TR" suffix).

Once you save trace data in a file using the **File→Store→Trace Data...** or ***store trace*** command, you can reload it. To view the data you loaded, use the **Display→Trace**, **Trace→Display**, or ***display trace*** command. Remember that a new trace measurement will overwrite this trace data (but not the file from which it was loaded).

The interface will try to display the trace listing in the display format active when the trace data was stored. If the interface needs symbols to replace absolute addresses or to find high-level source lines, and symbols are not loaded, an error occurs.

For example, suppose "source-mixed" was the display mode when the trace was captured and the executable file "test1" was the file being executed in the emulator/target system. To reload and display a trace listing saved from that emulation session requires reloading the symbols for "test1".

Example

Load a trace from a file:

```
load trace trace1
```

8

Making Software Performance Measurements

Making Software Performance Measurements

The Software Performance Measurement Tool (SPMT) is a feature of the Softkey Interface that allows you to make software performance measurements on your programs.

The SPMT allows you to make some of the measurements that are possible with the HP 64708 Software Performance Analyzer and its Graphical User Interface (HP B1487).

The SPMT post-processes information from the analyzer trace list. When you end a performance measurement, the SPMT dumps the post-processed information to a binary file, which is then read using the perf32 report generator utility.

Two types of software performance measurements can be made with the SPMT: activity measurements, and duration measurements.

This chapter describes tasks you perform while using the Software Performance Measurement Tool (SPMT). These tasks are grouped into the following sections:

- Activity performance measurements.
- Duration performance measurements.
- Running performance measurements and creating reports.

Activity Performance Measurements

Activity measurements are measurements of the number of accesses (reads or writes) within an address range. The SPMT shows you the percentage of analyzer trace states that are in the specified address range, as well as the percentage of time taken by those states. Two types of activity are measured: memory activity, and program activity.

Memory activity is all activity that occurs within the address range.

Program activity is the activity caused by instruction execution in the address range. Program activity includes opcode fetches and the cycles that result from the execution of those instructions (reads and writes to memory, stack pushes, etc.).

For example, suppose an address range being measured for activity contains an opcode that causes a stack push, which results in multiple write operations to the stack area (outside the range). The memory activity measurement will count only the stack push opcode cycle. However, the program activity measurement will count the stack push opcode cycle and the write operations to the stack.

By comparing the program activity and the memory activity in an address range, you can get an idea of how much activity in other areas is caused by the code being measured. An activity measurement report of the code (prog), data, and stack sections of a program is shown below.

This section describes how to:

- Set up the trace command for activity measurements.
- Initialize activity performance measurements.
- Interpret activity measurement reports.

```
 Label

prog
      Address Range       ADEH thru     1261H


      Memory Activity
           State Percent  Rel =  57.77  Abs =  57.77
                         Mean = 295.80  Sdv =  26.77
           Time  Percent  Rel =  60.97  Abs =  60.97

      Program Activity
           State Percent  Rel =  99.82  Abs =  99.82
                         Mean = 511.10  Sdv =   0.88
           Time  Percent  Rel =  99.84  Abs =  99.84

data
      Address Range     6007AH thru    603A5H

      Memory Activity
           State Percent  Rel =  30.51  Abs =  30.51
                         Mean = 156.20  Sdv =  31.87
           Time  Percent  Rel =  28.09  Abs =  28.09

      Program Activity
           State Percent  Rel =   0.18  Abs =   0.18
                         Mean =   0.90  Sdv =   0.88
           Time  Percent  Rel =   0.16  Abs =   0.16

stack
      Address Range     40000H thru    43FFFH


      Memory Activity
           State Percent  Rel =  11.72  Abs =  11.72
                         Mean =  60.00  Sdv =  29.24
           Time  Percent  Rel =  10.94  Abs =  10.94

      Program Activity
           State Percent  Rel =   0.00  Abs =   0.00
                         Mean =   0.00  Sdv =   0.00
           Time  Percent  Rel =   0.00  Abs =   0.00
```

```
         Graph of Memory Activity relative state percents >= 1
prog                  57.77%  *****************************
data                  30.51%  ****************
stack                 11.72%  ******

         Graph of Memory Activity relative time percents >= 1
prog                  60.97%  *******************************
data                  28.09%  **************
stack                 10.94%  ******


         Graph of Program Activity relative state percents >= 1
prog                  99.82% **************************************************


         Graph of Program Activity relative time percents >= 1
prog                  99.84% **************************************************


      Summary Information for     10 traces


           Memory Activity
           State count
                Relative count     5120
                Mean sample      170.67
                Mean Standard Dv 29.30
                95% Confidence 12.28% Error tolerance
           Time  count
                Relative Time - Us 2221.20


           Program Activity
           State count
                Relative count     5120
                Mean sample      170.67
                Mean Standard Dv 0.58
                95% Confidence 0.24% Error tolerance
           Time  count
                Relative Time - Us 2221.20
      Absolute Totals
                Absolute count - state     5120
                Absolute count - time - Us 2221.20
```

To set up the trace command for activity measurements

1 **Specify a trace display depth of 512.**

2 **Trace after any state, store all states, and count time.**

Before you initialize and run performance measurements, the current trace command (in other words, the last trace command entered) must be properly set up.

In general, you want to give the SPMT as many trace states as possible to post-process, so you should increase the trace depth to the maximum number, as shown in the following command.

If you wish to measure activity as a percentage of all activity, the current trace command should be the default (in other words, ***trace*** <RETURN>). The default trace command triggers on any state, and all captured states are stored. It is important that time be counted by the analyzer; otherwise, the SPMT measurements will not be correct. Also, since states are stored "after" the trigger state, the maximum number of captured states appears in each trace list.

You can qualify trace commands any way you like to obtain specific information. However, when you qualify the states that get stored in the trace memory, your SPMT results will be biased by your qualifications; the percentages shown will be of only those states stored in the trace list.

Examples

To specify a trace depth of 512:

```
display trace depth 512
```

To trace after any state, store all states, and count time:

```
trace counting time
```

To initialize activity performance measurements

- Use the **performance_measurement_initialize** command.

After you set up the trace command, you must tell the SPMT the address ranges on which you wish to make activity measurements. This is done by initializing the performance measurement. You can initialize the performance measurement in the following ways:

- Use default initialization (using global symbols if the symbols database is loaded).
- Initialize with user-defined files.
- Initialize with global symbols.
- Initialize with local symbols.
- Restore a previous performance measurement (if the emulation system has been exited and re-entered).

Default Initialization

Entering the ***performance_measurement_initialize*** command with no options specifies an activity measurement. If a valid symbolic database has been loaded, the addresses of all global procedures and static symbols will be used; otherwise, a default set of ranges that cover the entire processor address range will be used.

Initialization with User Defined Ranges

You can specifically give the SPMT address ranges to use by placing the information in a file and entering the file name in the ***performance_measurement_initialize*** command.

Address range files may contain program symbols (procedure name or static), user defined address ranges, and comments. An example address range file is shown below.

```
# Any line which starts with a # is a comment.
# All user’s labels must be preceded by a "|".

|users_label  10H 1000H
program_symbol

# A program symbol can be a procedure name or a static.  In the case of a pro-
# cedure name the range of that procedure will be used.

|users_label2 program_symbol1 -> program_symbol2

# "->" means thru.  The above will define a range which starts with symbol1
# and goes thru symbol2.  If both symbols are procedures then the range will
# be defined as the start of symbol1 thru the end of symbol2.

dir1/dir2/source_file.s:local_symbol

# The above defines a range based on the address of local_symbol.
```

Initialization with Global Symbols

When the ***performance_measurement_initialize*** command is entered with no options or with the global_symbols option, the global symbols in the symbols database become the address ranges for which activity is measured. If the symbols database is not loaded, a default set of ranges that cover the entire processor address range will be used.

The global symbols database contains procedure symbols, which are associated with the address range from the beginning of the procedure to the end, and static symbols, which are associated with the address of the static variable.

Initialization with Local Symbols

When the ***performance_measurement_initialize*** command is entered with the local_symbols_in option and a source file name, the symbols associated with that source file become the address ranges for which activity is measured. If the symbols database is not loaded, an error message will occur telling you that the source filename symbol was not found.

You can also use the local_symbols_in option with procedure symbols; this allows you to measure activity related to the symbols defined in a single function or procedure.

Restoring the Current Measurement

The ***performance_measurement_initialize*** restore command allows you to restore old performance measurement data from the perf.out file in the current directory.

If you have not exited and re-entered emulation, you can add traces to a performance measurement simply by entering another ***performance_measurement_run*** command. However, if you exit and re-enter the emulation system, you must enter the ***performance_measurement _initialize restore*** command before you can add traces to a performance measurement. When you restore a performance measurement, make sure your current trace command is identical to the command used with the restored measurement.

The restore option checks the emulator software version and will only work if the perf.out files you are restoring were made with the same software version as is presently running in the emulator. If you ran tests using a former software version and saved perf.out files, then updated your software to a new version number, you will not be able to restore old perf.out measurement files.

Examples

Suppose the "addr_ranges" file contains the names of all the functions in the "ecs" demo program loop:

```
combsort
do_sort
gen_ascii_data
get_targets
graph_data
interrupt_sim
proc_specific
read_conditions
save_points
set_outputs
strcpy8
update_system
write_hdwr
```

Since these labels are program symbols, you do not have to specify the address range associated with each label; the SPMT will search the symbol database for the addresses of each label.

An easy way to create the "addr_ranges" file is to use the copy global_symbols command to copy the global symbols to a file named "addr_ranges"; then, fork a shell to UNIX (by entering "! <RETURN>" on the Softkey Interface command line) and edit the file so that it contains the procedure names shown above. Enter a <CTRL>d at the UNIX prompt to return to the Softkey Interface.

To initialize the activity measurement with a user-defined address range file:

```
performance_measurement_initialize addr_ranges
```

To interpret activity measurement reports

- **View the performance measurement report.**

Activity measurements are measurements of the number of accesses (reads or writes) within an address range. The reports generated for activity measurements show you the percentage of analyzer trace states that are in the specified address range, as well as the percentage of time taken by those states. The performance measurement must include four traces before statistics (mean and standard deviation) appear in the activity report. The information you will see in activity measurement reports is described below.

Memory Activity All activity found within the address range.

Program Activity All activity caused by instruction execution in the address range. Program activity includes opcode fetches and the cycles that result from the execution of those instructions (reads and writes to memory, stack pushes, etc.).

Relative With respect to activity in all ranges defined in the performance measurement.

Absolute With respect to all activity, not just activity in those ranges defined in the performance measurement.

Mean Average number of states in the range per trace. The following equation is used to calculate the mean:

$$mean = \frac{states\ \ in\ \ range}{total\ \ states}$$

Standard Deviation Deviation from the mean of state count. The following equation is used to calculate standard deviation:

$$std\ dev = \sqrt{\frac{1}{N-1} \times \sum_{i=1}^{N} S_{sumq} - N\ (mean)^2}$$

Where:

N	Number of traces in the measurement.
mean	Average number of states in the range per trace.
S_{sumq}	Sum of squares of states in the range per trace.

Symbols Within Range Names of other symbols that identify addresses or ranges of addresses within the range of this symbol.

Additional Symbols for Address Names of other symbols that also identify this address.

Note that some compilers emit more than one symbol for certain addresses. For example, a compiler may emit "interrupt_sim" and "_interrupt_sim" for the first address in a routine named interrupt_sim. The analyzer will show the first symbol it finds to represent a range of addresses, or a single address point, and it will show the other symbols under either "Symbols within range" or "Additional symbols for address", as applicable. In the "interrupt_sim" example, it may show either "interrupt_sim" or "_interrupt_sim" to represent the range, depending on which symbol it finds first. The other symbol will be shown below "Symbols within range" in the report. These conditions appear particularly in default measurements that include all global and local symbols.

Relative and Absolute Counts Relative count is the total number of states associated with the address ranges in the performance measurement. Relative time is the total amount of time associated with the address ranges in the performance measurement. The absolute counts are the number of states or amount of time associated with all the states in all the traces.

Error Tolerance and Confidence Level An approximate error may exist in displayed information. Error tolerance for a level of confidence is calculated using the mean of the standard deviations and the mean of the means. Error tolerance gives an indication of the stability of the information. For example, if the error is 5% for a confidence level of 95%, then you can be 95% confident that the information has an error of 5% or less.

The Student's "T" distribution is used in these calculations because it improves the accuracy for small samples. As the size of the sample increases, the Student's "T" distribution approaches the normal distribution.

The following equation is used to calculate error tolerance:

$$error\ pct. = \frac{O_m \times t}{N \times P_m} \times 100$$

Where:

O_m	Mean of the standard deviations.
t	Table entry in Student's "T" table for a given confidence level.
N	Number of traces in the measurement.
P_m	Mean of the means (i.e., mean sample).

Examples

Consider the following activity measurement report (generated with the commands shown):

```
display trace depth 512
trace counting time
performance_measurement_initialize addr_ranges
performance_measurement_run 20
performance_measurement_end
!perf32 | more
```

```
 Label

set_outputs
         Address Range      1784H thru     1814H


         Memory Activity
                 State Percent  Rel =  30.28  Abs =  25.00
                               Mean = 128.00  Sdv = 227.46
                 Time  Percent  Rel =  30.45  Abs =  25.45

         Program Activity
                 State Percent  Rel =  28.97  Abs =  25.00
                               Mean = 128.00  Sdv = 227.46
                 Time  Percent  Rel =  29.28  Abs =  25.45

update_system
         Address Range      159CH thru     1656H


         Memory Activity
                 State Percent  Rel =  30.28  Abs =  25.00
                               Mean = 128.00  Sdv = 227.46
                 Time  Percent  Rel =  30.44  Abs =  25.45

         Program Activity
                 State Percent  Rel =  28.99  Abs =  25.02
                               Mean = 128.10  Sdv = 227.40
                 Time  Percent  Rel =  29.29  Abs =  25.46

read_conditions
         Address Range      16EEH thru     177CH


         Memory Activity
                 State Percent  Rel =  12.11  Abs =  10.00
                               Mean =  51.20  Sdv = 157.59
                 Time  Percent  Rel =  12.18  Abs =  10.18

         Program Activity
                 State Percent  Rel =  11.59  Abs =  10.00
                               Mean =  51.20  Sdv = 157.59
                 Time  Percent  Rel =  11.71  Abs =  10.18

strcpy8
```

```
         Address Range      10B0H thru     110AH


         Memory Activity
                 State Percent  Rel =   9.75  Abs =   8.05
                               Mean =  41.20  Sdv = 116.63
                 Time  Percent  Rel =   9.45  Abs =   7.90

         Program Activity
                 State Percent  Rel =  12.39  Abs =  10.69
                               Mean =  54.75  Sdv = 149.76
                 Time  Percent  Rel =  11.83  Abs =  10.28

interrupt_sim
         Address Range      101EH thru     10A8H


         Memory Activity
                 State Percent  Rel =   6.15  Abs =   5.08
                               Mean =  26.00  Sdv = 114.41
                 Time  Percent  Rel =   5.96  Abs =   4.98

         Program Activity
                 State Percent  Rel =   5.97  Abs =   5.16
                               Mean =  26.40  Sdv = 114.35
                 Time  Percent  Rel =   5.81  Abs =   5.05

write_hdwr
         Address Range      181CH thru     1894H


         Memory Activity
                 State Percent  Rel =   6.06  Abs =   5.00
                               Mean =  25.60  Sdv = 114.49
                 Time  Percent  Rel =   6.10  Abs =   5.10

         Program Activity
                 State Percent  Rel =   5.79  Abs =   5.00
                               Mean =  25.60  Sdv = 114.49
                 Time  Percent  Rel =   5.86  Abs =   5.10

proc_specific
         Address Range      1A6CH thru     1A8CH
         Memory Activity
                 State Percent  Rel =   3.84  Abs =   3.17
                                Mean =  16.25  Sdv =  72.67
                 Time  Percent  Rel =   3.86  Abs =   3.23

         Program Activity
                 State Percent  Rel =   3.70  Abs =   3.19
                                Mean =  16.35  Sdv =  73.12
                 Time  Percent  Rel =   3.73  Abs =   3.24

combsort
         Address Range      124EH thru     1444H


         Memory Activity
                 State Percent  Rel =   1.06  Abs =   0.88
                                Mean =   4.50  Sdv =  20.12
                 Time  Percent  Rel =   1.06  Abs =   0.89

         Program Activity
                 State Percent  Rel =   1.90  Abs =   1.64
                                Mean =   8.40  Sdv =  37.57
                 Time  Percent  Rel =   1.80  Abs =   1.56

do_sort
         Address Range      144CH thru     14EAH


         Memory Activity
                 State Percent  Rel =   0.47  Abs =   0.39
                                Mean =   2.00  Sdv =   5.30
                 Time  Percent  Rel =   0.49  Abs =   0.41

         Program Activity
                 State Percent  Rel =   0.70  Abs =   0.61
                                Mean =   3.10  Sdv =   7.68
                 Time  Percent  Rel =   0.69  Abs =   0.60

gen_ascii_data
         Address Range      1112H thru     1246H


         Memory Activity
                 State Percent  Rel =   0.00  Abs =   0.00
```

```
                              Mean =   0.00  Sdv =   0.00
                Time  Percent  Rel =   0.00  Abs =   0.00

        Program Activity
                State Percent  Rel =   0.00  Abs =   0.00
                              Mean =   0.00  Sdv =   0.00
                Time  Percent  Rel =   0.00  Abs =   0.00

get_targets
        Address Range      165EH thru     16E6H


        Memory Activity
                State Percent  Rel =   0.00  Abs =   0.00
                              Mean =   0.00  Sdv =   0.00
                Time  Percent  Rel =   0.00  Abs =   0.00

        Program Activity
                State Percent  Rel =   0.00  Abs =   0.00
                              Mean =   0.00  Sdv =   0.00
                Time  Percent  Rel =   0.00  Abs =   0.00

graph_data
        Address Range      1988H thru     1A40H


        Memory Activity
                State Percent  Rel =   0.00  Abs =   0.00
                              Mean =   0.00  Sdv =   0.00
                Time  Percent  Rel =   0.00  Abs =   0.00

        Program Activity
                State Percent  Rel =   0.00  Abs =   0.00
                              Mean =   0.00  Sdv =   0.00
                Time  Percent  Rel =   0.00  Abs =   0.00

proc_spec_init
        Address Range      1A48H thru     1A64H


        Memory Activity
                State Percent  Rel =   0.00  Abs =   0.00
                              Mean =   0.00  Sdv =   0.00
                Time  Percent  Rel =   0.00  Abs =   0.00
```

```
         Program Activity
                 State Percent  Rel =   0.00  Abs =   0.00
                               Mean =   0.00  Sdv =   0.00
                 Time  Percent  Rel =   0.00  Abs =   0.00

save_points
         Address Range      189CH thru     1980H


         Memory Activity
                 State Percent  Rel =   0.00  Abs =   0.00
                               Mean =   0.00  Sdv =   0.00
                 Time  Percent  Rel =   0.00  Abs =   0.00

         Program Activity
                 State Percent  Rel =   0.00  Abs =   0.00
                               Mean =   0.00  Sdv =   0.00
                 Time  Percent  Rel =   0.00  Abs =   0.00



         Graph of Memory Activity relative state percents >= 1
set_outputs           30.28%  ***************
update_system         30.28%  ***************
read_conditions       12.11%  ******
strcpy8                9.75%  *****
interrupt_sim          6.15%  ***
write_hdwr             6.06%  ***
proc_specific          3.84%  **
combsort               1.06%  *


         Graph of Memory Activity relative time percents >= 1
set_outputs           30.45%  ***************
update_system         30.44%  ***************
read_conditions       12.18%  ******
strcpy8                9.45%  *****
interrupt_sim          5.96%  ***
write_hdwr             6.10%  ***
proc_specific          3.86%  **
combsort               1.06%  *


         Graph of Program Activity relative state percents >= 1
```

```
set_outputs           28.97%  ***************
update_system         28.99%  ***************
read_conditions       11.59%  ******
strcpy8               12.39%  ******
interrupt_sim          5.97%  ***
write_hdwr             5.79%  ***
proc_specific          3.70%  **
combsort               1.90%  *


         Graph of Program Activity relative time percents >= 1
set_outputs           29.28%  ***************
update_system         29.29%  ***************
read_conditions       11.71%  ******
strcpy8               11.83%  ******
interrupt_sim          5.81%  ***
write_hdwr             5.86%  ***
proc_specific          3.73%  **
combsort               1.80%  *

         Summary Information for     20 traces


                 Memory Activity
                 State count
                         Relative count     8455
                         Mean sample       30.20
                         Mean Standard Dv 75.44
                         95% Confidence 116.98% Error tolerance
                 Time  count
                         Relative Time - Us 3500.92


                 Program Activity
                 State count
                         Relative count     8838
                         Mean sample       31.56
                         Mean Standard Dv 79.24
                         95% Confidence 117.55% Error tolerance
                 Time  count
                         Relative Time - Us 3641.08
         Absolute Totals
                         Absolute count - state    10240
                         Absolute count - time - Us 4188.56
```

The measurements for each label are printed in descending order according to the amount of activity. You can see that the set_outputs function has the most activity. Also, you can see that no activity is recorded for several of the functions. The histogram portion of the report compares the activity in the functions that account for at least 1% of the activity for all labels defined in the measurement.

Duration Performance Measurements

Duration measurements provide a best-case/worst-case characterization of code execution time. These measurements record execution times that fall within a set of specified time ranges. The analyzer trace command is set up to store only the entry and exit states of the module to be measured (for example, a C function or Pascal procedure). The SPMT provides two types of duration measurements: module duration, and module usage.

Module duration measurements record how much time it takes to execute a particular code segment (for example, a function in the source file).

Module usage shows how much of the execution time is spent outside of the module (from exit to entry). This measurement gives an indication of how often the module is being used.

When using the SPMT to perform duration measurements, there should be only two addresses stored in the trace memory: the entry address, and the exit address. Recursion can place several entry addresses before the first exit address, and/or several exit addresses before the first entry address. Duration measurements are made between the last entry address in a series of entry addresses, and the last exit address in a series of exit addresses (see the figure below). All of the entry and exit addresses which precede these last addresses are assumed to be unused prefetches, and are ignored during time measurements.

When measuring a recursive function, module duration will be measured between the last recursive call and the true end of the recursive execution. This will affect the accuracy of the measurement.

If a module is entered at the normal point, and then exited by a point other than the defined exit point, the entry point will be ignored. It will be judged the same as any other unused prefetch, and no time-duration measurement will be made. Its time will be included in the measure of time spent outside the procedure or function.

If a module is exited from the normal point, and then re-entered from

```
START  - assumed prefetch
START  - assumed prefetch
START  - assumed prefetch
START  - last ENTRY address -
END    - assumed prefetch
END    - assumed prefetch                  Measure duration
END    - assumed prefetch
END    - last EXIT address -
START  - assumed prefetch
START  - assumed prefetch                  Measure duration
START  - assumed prefetch
START  - last ENTRY address -
END    - assumed prefetch
END    - assumed prefetch
```

some other point, the exit will also be assumed to be an unused prefetch of the exit state.

Note that if you are making duration measurements on a function that is recursive, or one that has multiple entry and/or exit points, you may wind up with invalid information.

This section describes how to:

- Set up the trace command for duration measurements.
- Initialize duration performance measurements.
- Interpret duration measurement reports.

To set up the trace command for duration measurements

1 Specify a trace display depth of 512.

2 Trace after and store only function start and end addresses.

For duration measurements, the trace command must be set up to store only the entry and exit points of the module of interest. Since the trigger state is always stored, you should trigger on the entry or exit points. For example:

```
trace after symbol_entry or symbol_exit only
symbol_entry or symbol_exit counting time
```

CAUTION The previous command depends on the generation of correct exit address symbols by the software development tools.

Or:

```
trace after module_name start or module_name end
only module_name start or module_name end
counting time
```

Where "symbol_entry" and "symbol_exit" are symbols from the user program. Or, where "module_name" is the name of a C function or Pascal procedure (and is listed as a procedure symbol in the global symbol display).

Examples

To specify a trace display depth of 512:

```
display trace depth 512
```

To set up the trace command for duration measurements on the interrupt_sim function:

```
trace after interrupt_sim start or interrupt_sim
end only interrupt_sim start or interrupt_sim end
counting time
```

The trace specification sets up the analyzer to capture only the states that contain the start address of the interrupt_sim function or the end address of the interrupt_sim function. Since the trigger state is also stored, the analyzer is set up to trigger on the entry or exit address of the interrupt_sim function. With these states in memory, the analyzer will derive two measurements: time from start to end of interrupt_sim, and time from end to start of interrupt_sim.

To initialize duration performance measurements

- Use the **performance_measurement_initialize** command with the **duration** option.

After you set up the trace command, you must tell the SPMT the time ranges to be used in the duration measurement. This is done by initializing the performance measurement. You can initialize the performance measurement in the following ways:

- Initialize with user-defined files.
- Restore a previous performance measurement (if the emulation system has been exited and re-entered).

Initialization with User Defined Ranges

You can specifically give the SPMT time ranges to use by placing the information in a file and entering the file name in the ***performance_measurement_initialize*** command.

Time range files may contain comments and time ranges in units of microseconds (us), milliseconds (ms), or seconds (s). An example time range file is shown below.

```
# Any line which starts with a # is a comment.

1 us 20 us
10.1 ms 100.6 ms
3.55 s  6.77 s

# us microseconds
# ms milliseconds
#  s seconds
#
# The above are the only abbreviations allowed.  The space between the  number
# and the units abbreviation is required.
```

When no user defined time range file is specified, the following set of default time ranges are used.

```
1 us 10 us
10.1 us 100 us
100.1 us 500 us
500.1 us 1 ms
1.001 ms 5 ms
```

```
5.001 ms 10 ms
10.1 ms 20 ms
20.1 ms 40 ms
40.1 ms 80 ms
80.1 ms 160 ms
160.1 ms 320 ms
320.1 ms 640 ms
640.1 ms 1.2 s
```

Restoring the Current Measurement

The ***performance_measurement_initialize*** restore command allows you to restore old performance measurement data from the perf.out file in the current directory.

If you have not exited and re-entered emulation, you can add traces to a performance measurement simply by entering another ***performance_measurement_run*** command. However, if you exit and re-enter the emulation system, you must enter the ***performance_measurement _initialize restore*** command before you can add traces to a performance measurement. When you restore a performance measurement, make sure your current trace command is identical to the command used with the restored measurement.

The restore option checks the emulator software version and will only work if the perf.out files you are restoring were made with the same software version as is presently running in the emulator. If you ran tests using a former software version and saved perf.out files, then updated your software to a new version number, you will not be able to restore old perf.out measurement files.

Examples

To initialize the duration measurement:

```
performance_measurement_initialize duration
```

To interpret duration measurement reports

- View the performance measurement report.

Duration measurements provide a best-case/worst-case characterization of code execution time. These measurements record execution times that fall within a set of specified time ranges. The information you will see in duration measurement reports is described below.

Number of Intervals Number of "from address" and "to address" pairs (after prefetch correction).

Maximum Time The greatest amount of time between the "from address" to the "to address".

Minimum Time The shortest amount of time between the "from address" to the "to address".

Average Time Average time between the "from address" and the "to address". The following equation is used to calculate the average time:

$$mean = \frac{amount\ \ of\ \ time\ \ \text{for}\ \ all\ \ intervals}{number\ \ of\ \ intervals}$$

Standard Deviation Deviation from the mean of time. The following equation is used to calculate standard deviation:

$$std\ dev = \sqrt{\frac{1}{N-1} \times \sum_{i=1}^{N} S_{sumq} - N\ (mean)^2}$$

Where:

N	Number of intervals.
mean	Average time.
S_{sumq}	Sum of squares of time in the intervals.

Error Tolerance and Confidence Level An approximate error may exist in displayed information. Error tolerance for a level of confidence is calculated using the mean of the standard deviations and the mean of the means. Error tolerance gives an indication of the stability of the information. For example, if the error is 5% for a confidence level of 95%, then you can be 95% confident that the information has an error of 5% or less.

The Student's "T" distribution is used in these calculations because it improves the accuracy for small samples. As the size of the sample increases, the Student's "T" distribution approaches the normal distribution.

The following equation is used to calculate error tolerance:

$$error\ pct. = \frac{O_m \times t}{N \times P_m} \times 100$$

Where:

O_m	Mean of the standard deviations in each time range.
t	Table entry in Student's "T" table for a given confidence level.
N	Number of intervals.
P_m	Mean of the means (i.e., mean of the average times in each time range).

Examples

Consider the following duration measurement report (generated with the commands shown):

```
display trace depth 512
trace after interrupt_sim start or interrupt_sim
end only interrupt_sim start or interrupt_sim end
counting time
performance_measurement_initialize duration
performance_measurement_run 10
performance_measurement_end
!perf32 | more
```

```
                 Time Interval Profile



From Address      10A8
         File main(module)."/users/guest/demo/debug_env/hp64782/main.c"
         Symbolic Reference at interrupt_sim+8A
To Address        101E
         File main(module)."/users/guest/demo/debug_env/hp64782/main.c"
         Symbolic Reference at main.interrupt_sim
Number of intervals   2550
Maximum Time 73297.920 us
Minimum Time 48230.400 us
Avg Time     55672.752 us


         Statistical summary - for     10 traces
                 Stdv 11442.64
                 95% Confidence 0.80% Error tolerance


         Graph of relative percents
1 us 10 us              0.00%
10.1 us 100 us          0.00%
100.1 us 500 us         0.00%
500.1 us 1 ms           0.00%
1.001 ms 5 ms           0.00%
5.001 ms 10 ms          0.00%
```

```
10.1 ms  20 ms         0.00%
20.1 ms  40 ms         0.00%
40.1 ms  80 ms       100.00%
**************************************************
80.1 ms  160 ms        0.00%
160.1 ms 320 ms        0.00%
320.1 ms 640 ms        0.00%
640.1 ms 1.2 s         0.00%


From Address      101E
         File main(module)."/users/guest/demo/debug_env/hp64782/main.c"
         Symbolic Reference at main.interrupt_sim
To Address        10A8
         File main(module)."/users/guest/demo/debug_env/hp64782/main.c"
         Symbolic Reference at interrupt_sim+8A
Number of intervals   2550
Maximum Time 342343.680 us
Minimum Time 52.320 us
Avg Time     36987.751 us


         Statistical summary - for     10 traces
                 Stdv 76924.84
                 95% Confidence 8.07% Error tolerance


         Graph of relative percents
1 us 10 us             0.00%
10.1 us 100 us        14.82%  ********
100.1 us 500 us        5.06%  ***
500.1 us 1 ms          0.00%
1.001 ms 5 ms         24.82%  *************
5.001 ms 10 ms        20.27%  **********
10.1 ms  20 ms        10.08%  *****
20.1 ms  40 ms         0.00%
40.1 ms  80 ms         9.88%  *****
80.1 ms  160 ms        5.02%  ***
160.1 ms 320 ms        7.57%  ****
320.1 ms 640 ms        2.47%  *
640.1 ms 1.2 s         0.00%
```

Two sets of information are given in the duration measurement report: module duration and module usage.

The first set is the "module usage" measurement. Module usage measurements show how much time is spent outside the module of interest; they indicate how often the module is used. The information shown in the first part of the duration report above shows that the average amount of time spent outside the interrupt_sim function is about 55.7 milliseconds.

The second set of information in the duration measurement report is the "module duration" measurement. The module duration report shows that the amount of time it takes for the interrupt_sim function to execute varies from 52.3 microseconds to 342.3 milliseconds. The average amount of time it takes for the interrupt_sim module to execute is roughly 37 milliseconds.

Running Measurements and Creating Reports

Several performance measurement tasks are the same whether you are making activity or duration measurements.

This section describes how to:

- Run performance measurements.
- End performance measurements.
- Create a performance measurement report.

To run performance measurements

- Use the **performance_measurement_run** command.

The ***performance_measurement_run*** command processes analyzer trace data. When you end the performance measurement, this processed data is dumped to the binary "perf.out" file in the current directory. The perf32 report generator utility is used to read the binary information in the "perf.out" file.

If the ***performance_measurement_run*** command is entered without a count, the current trace data is processed. If a count is specified, the current trace command is executed consecutively the number of times specified. The data that results from each trace command is processed and combined with the existing processed data. The STATUS line will say "Processing trace <NO.>" during the run so you will know how your measurement is progressing. The only way to stop this series of traces is by using <CTRL>c (sig INT).

The more traces you include in your sample, the more accurate will be your results. At least four consecutive traces are required to obtain statistical interpretation of activity measurement results.

Examples

To run the performance measurement, enter the following command:

```
performance_measurement_run 20
```

The command above causes 20 traces to occur. The SPMT processes the trace information after each trace, and the number of the trace being processed is shown on the status line.

To end performance measurements

- Use the **performance_measurement_end** command.

The ***performance_measurement_end*** command takes the data generated by the ***performance_measurement_run*** command and places it in a file named perf.out in the current directory. If a file named "perf.out" already exists in the current directory, it will be overwritten. Therefore, if you wish to save a performance measurement, you must rename the perf.out file before performing another measurement.

The ***performance_measurement_end*** command does not affect the current performance measurement data which exists within the emulation system. In other words, you can add more traces later to the existing performance measurement by entering another performance_measurement_run command.

Once you have entered the ***performance_measurement_end*** command, you can use the perf32 report generator to look at the data saved in the perf.out file.

Note that the "perf.out" file is a binary file. Do not try to read it with the UNIX more or cat commands. The perf32 report generator utility (described in the following section) must be used to read the contents of the "perf.out" file.

Examples

To cause the processed trace information to be dumped to the "perf.out" file:

```
performance_measurement_end
```

To create a performance measurement report

- **Use the perf32 command at the UNIX prompt.**

The perf32 report generator utility must be used to read the information in the "perf.out" file and other files dumped by the SPMT (in other words, renamed "perf.out" files). The perf32 utility is run from the UNIX shell. You can fork a shell while in the Softkey Interface and run perf32, or you can exit the Softkey Interface and run perf32.

Options to "perf32"

A default report, containing all performance measurement information, is generated when the perf32 command is used without any options. The options available with perf32 allow you to limit the information in the generated report. These options are described below.

-h	Produce outputs limited to histograms.
-s	Produce a summary limited to the statistical data.
-p	Produce a summary limited to the program activity.
-m	Produce a summary limited to the memory activity.
-f<file>	Produce a report based on the information contained in <file> instead of the information contained in perf.out.

For example, the following commands save the current performance measurement information in a file called "perf1.out", and produce a histogram showing only the program activity occupied by the functions and variables.

```
mv perf.out perf1.out
perf32 -hpf perf1.out
```

Options -h, -s, -p, and -m affect the contents of reports generated for activity measurements. These options have no effect on the contents of reports generated for duration (time interval) measurements.

Examples

Now, to generate a report from the "perf.out" file, type the following on the command line to fork a shell and run the perf32 utility:

```
!perf32 | more
```

9

Using the External State Analyzer

Using the External State Analyzer

The HP 64703A analyzer provides an external analyzer with 16 external trace channels. These trace channels allow you to capture activity on signals external to the emulator, typically other target system signals. The external analyzer may be configured as an extension to the emulation analyzer, as an independent state analyzer, or as an independent timing analyzer.

When the external analyzer is configured as an independent state analyzer, the emulator/analyzer interface does not control the external analyzer. However, you can use pod commands to control the independent state analyzer via the terminal interface. Refer to the *6830x Installation/Service/Terminal Interface User's Guide* for information on using the terminal interface commands for the external analyzer when it is configured as an independent state analyzer.

When the external analyzer is configured as an independent timing analyzer, you must use a special Timing Analyzer Interface program. Refer to the *Timing Analyzer Interface User's Guide* for information on using the external analyzer when it is configured as an independent timing analyzer.

The tasks you perform with the external analyzer are grouped into the following sections:

- Setting up the external analyzer.
- Configuring the external analyzer.

Setting Up the External Analyzer

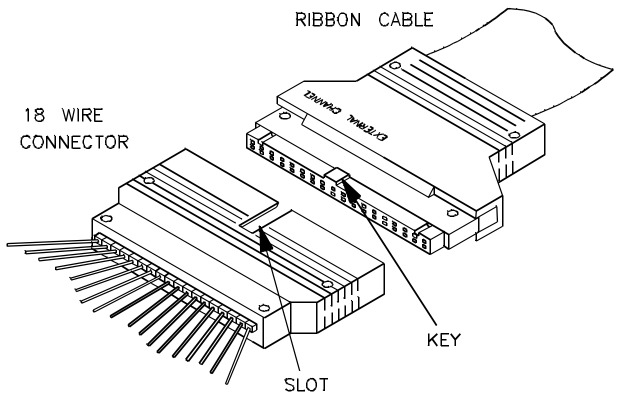
This section assumes you have already connected the external analyzer probe to the HP 64700 Card Cage.

Before you can use the external analyzer, you must:

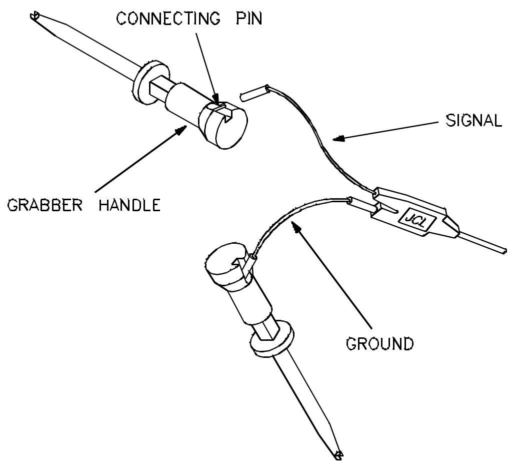
- Connect the external analyzer probe to the target system.
- Specify threshold voltages of external trace signals.
- Label the external trace signals.
- Select the external analyzer mode.

To connect the external analyzer probe to the target system

1 Assemble the Analyzer Probe. The analyzer probe is a two-piece assembly, consisting of ribbon cable and 18 probe wires (16 data channels and the J and K clock inputs) attached to a connector. Either end of the ribbon cable may be connected to the 18-wire connector, and the connectors are keyed so they may only be attached one way. Align the key of the ribbon cable connector with the slot in the 18-wire connector, and firmly press the connectors together.



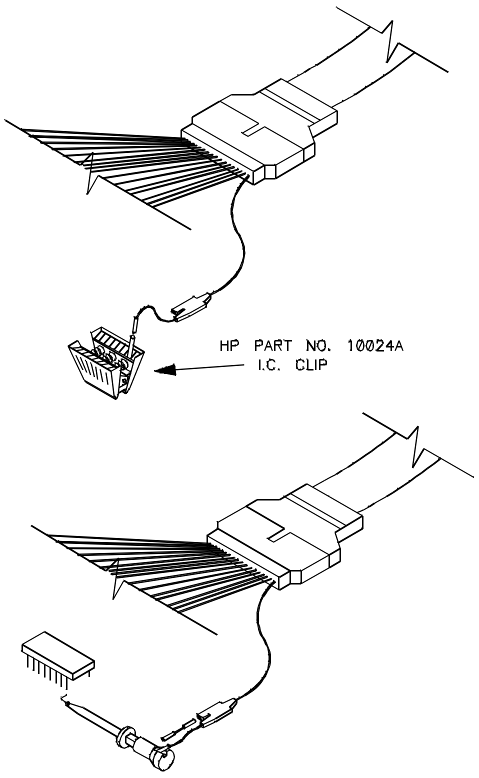
2 Attach grabbers to probe wires. Each of the 18 probe wires has a signal and a ground connection. Each probe wire is labeled for easy identification. Thirty-six grabbers are provided for the signal and ground connections of each of the 18 probe wires. The signal and ground connections are attached to the pin in the grabber handle.



CAUTION

Turn OFF target system power before connecting analyzer probe wires to the target system. The probe grabbers are difficult to handle with precision, and it is extremely easy to short the pins of a chip (or other connectors which are close together) with the probe wire while trying to connect it.

3 You can connect the grabbers to pins, connectors, wires, etc., in the target system. Pull the hilt of the grabber towards the back of the grabber handle to uncover the wire hook. When the wire hook is around the desired pin or connector, release the hilt to allow the grabber spring tension to hold the connection.



Configuring the External Analyzer

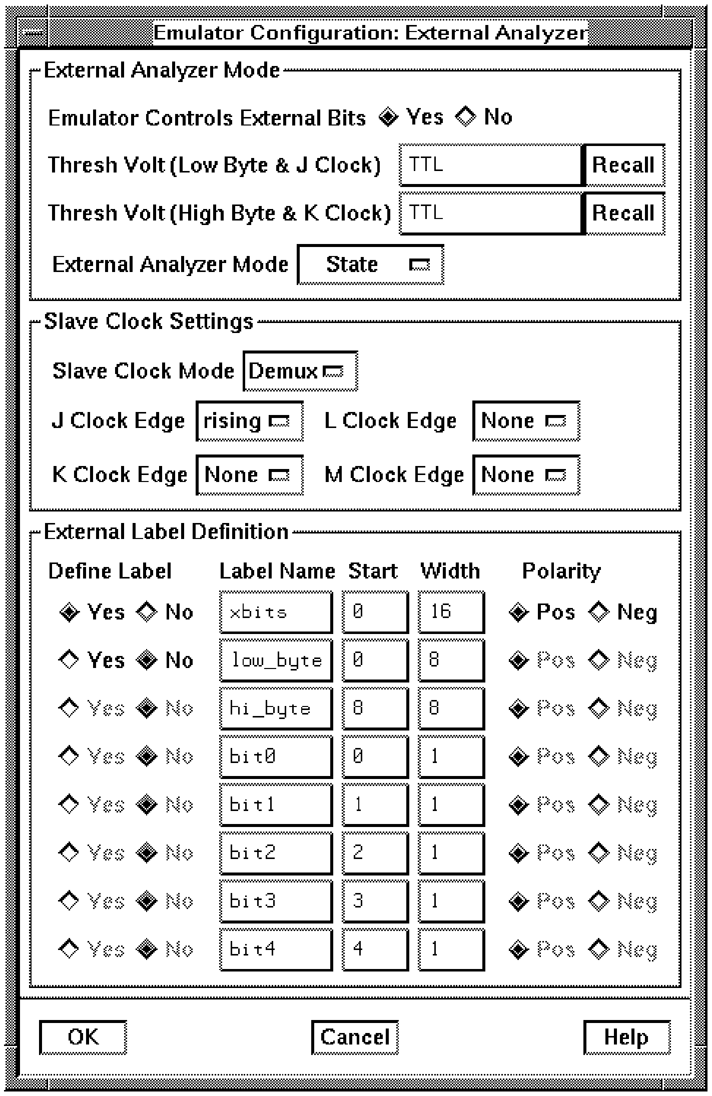
After you have assembled the external analyzer probe and connected it to the emulator and target system, the next step is to configure the external analyzer.

The external analyzer is a versatile instrument, and you can configure it to suit your needs. For example, you can specify threshold voltage levels on the external analyzer channels, and you can operate the external analyzer in several different modes.

The default configuration specifies that the external analyzer is aligned with the emulation analyzer. TTL level threshold voltages are defined, as well as an external label named "xbits" which contains all 16 channels.

In order to configure the external analyzer, you must first start the configuration interface and access the "External Analyzer" configuration section (refer to the "Using the Configuration Interface" section in the "Configuring the Emulator" chapter).

Figure 24



Emulator Configuration: External Analyzer

External Analyzer Mode

Emulator Controls External Bits [✓ Yes] No

Thresh Volt (Low Byte & J Clock) TTL Recall

Thresh Volt (High Byte & K Clock) TTL Recall

External Analyzer Mode State

Slave Clock Settings

Slave Clock Mode Demux

J Clock Edge rising L Clock Edge None

K Clock Edge None M Clock Edge None

External Label Definition

Define Label	Label Name	Start	Width	Polarity
[✓ Yes] No	xbits	0	16	[✓ Pos] Neg
Yes [✓ No]	low_byte	0	8	[✓ Pos] Neg
Yes [✓ No]	hi_byte	8	8	[✓ Pos] Neg
Yes [✓ No]	bit0	0	1	[✓ Pos] Neg
Yes [✓ No]	bit1	1	1	[✓ Pos] Neg
Yes [✓ No]	bit2	2	1	[✓ Pos] Neg
Yes [✓ No]	bit3	3	1	[✓ Pos] Neg
Yes [✓ No]	bit4	4	1	[✓ Pos] Neg

If you're using the Softkey Interface from a terminal or terminal emulation window, you don't get a dialog box from which to choose configuration sections; however, you have access to the same configuration options through a series of configuration questions. To access the questions in the

"External Analyzer" section, answer "yes" to the "Modify external analyzer configuration?" question.

This section describes how to:

- Specify whether the emulation emulator/analyzer interface should control the external analyzer.
- Specify the threshold voltages for the external channels.
- Select the external analyzer mode.
- Specify the slave clock mode when configured as an independent state analyzer.
- Define labels for the external analyzer channels.

To control the external analyzer with the emulator/analyzer interface

- **Choose "yes" or "no" for the "Should emulation control the external bits" configuration option.**

Answer "yes" if the emulation emulator/analyzer interface should control the external analyzer. You must answer "yes" to access the remaining external analyzer configuration questions. At the end of the configuration process the external analyzer mode and threshold voltages will be set; existing labels will be deleted, and only the labels specified in response to the questions below will be defined.

Answer "no" if the emulation emulator/analyzer interface shouldn't control the external analyzer. If emulation does not control the external bits, the external analyzer configuration will not be modified in any way by the emulation interface.

To specify the threshold voltage

1. **Choose "yes" for the "Should emulation control the external bits" configuration option.**
2. **Enter the "Threshold voltage for bits 0-7 and J clock" value.**
3. **Enter the "Threshold voltage for bits 8-15 and K clock" value.**

The external analyzer probe signals are divided into two groups: the lower byte (channels 0 through 7 and the J clock), and the upper byte (channels 8 through 15 and the K clock). You can specify a threshold voltage for each of these groups.

The default threshold voltages are specified as TTL which translates to 1.40 volts.

Voltages may be in the range from -6.40 volts to 6.35 volts (with a 0.05V resolution). You may also specify CMOS (which translates to 2.5 volts), or ECL (which translates to -1.3 volts).

To specify the external analyzer mode

1 Choose "yes" for the "Should emulation control the external bits" configuration option.

2 Choose "Emulation", "State", or "Timing" for the "External analyzer mode" configuration option.

The default configuration selects the "emulation" external analyzer mode. In this mode, you have 16 external trace signals on which data is captured synchronously with the emulation clock.

The external analyzer may also operate as an independent state analyzer, or it may operate as an independent timing analyzer if a host computer interface program is used.

Choose "emulation" to select the emulation mode. In this mode, the external analyzer becomes an extension of the emulation analyzer. In other words, they operate as one analyzer. The external bits are clocked with the emulation clock. External labels may be used in trace commands to qualify trigger, storage, prestore, or count states. External labels may be viewed in the trace display.

Choose "state" to select the independent state mode of the external analyzer. The external bits are not available for use from the emulation interface. You can, however, use pod commands to control the external state analyzer in its independent mode.

Choose "timing" to select the timing mode of the external analyzer. The external bits are not available for use from the emulation interface. Because the pod commands for the timing analyzer dump information in binary format, you will need to use Timing Analyzer Interface, or other interface program, to capture the timing analyzer data.

To specify the slave clock mode

1 **Choose "yes" for the "Should emulation control the external bits" configuration option.**

2 **Choose "State" for the "External analyzer mode" configuration option.**

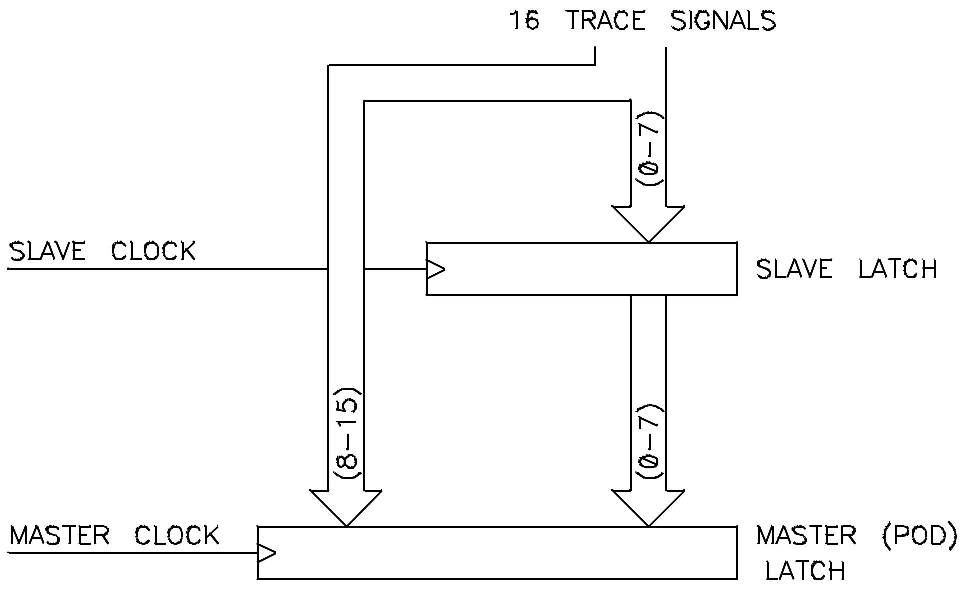
3 **Choose "Off", "Mixed", or "Demux" for the "Slave clock mode for external bits" configuration item.**

There are two modes of demultiplexing that can be set for the 16 channels of the external analyzer: mixed clocks and true demultiplexing.

Choose "off" to turn slave clocks OFF. If the slave clock is "off", all 16 external bits are clocked with the emulation clock.

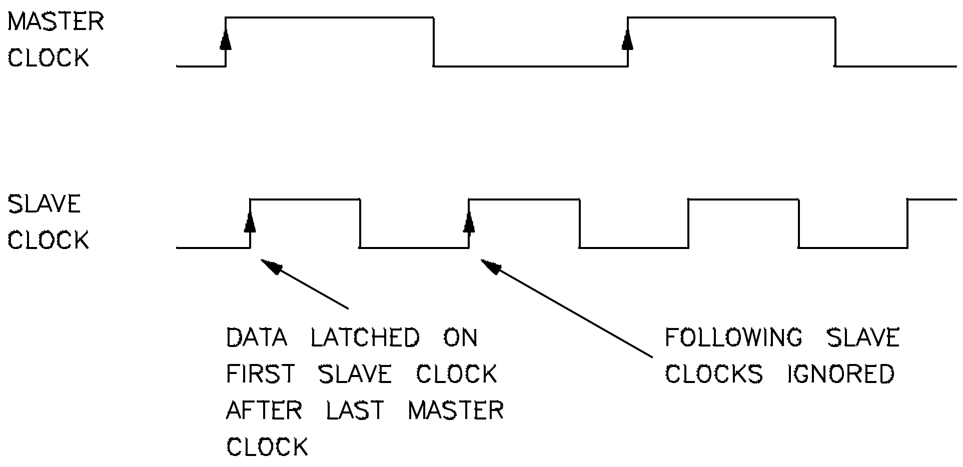
Choose "mixed" to specify the mixed clock demultiplexing mode. In this mode, the lower eight external bits (0-7) are latched when the slave clock (as specified by your answers to the next four questions) is received. The upper eight bits and the latched lower eight are then clocked into the analyzer when the emulation clock is received (see the figure below).

Figure 25



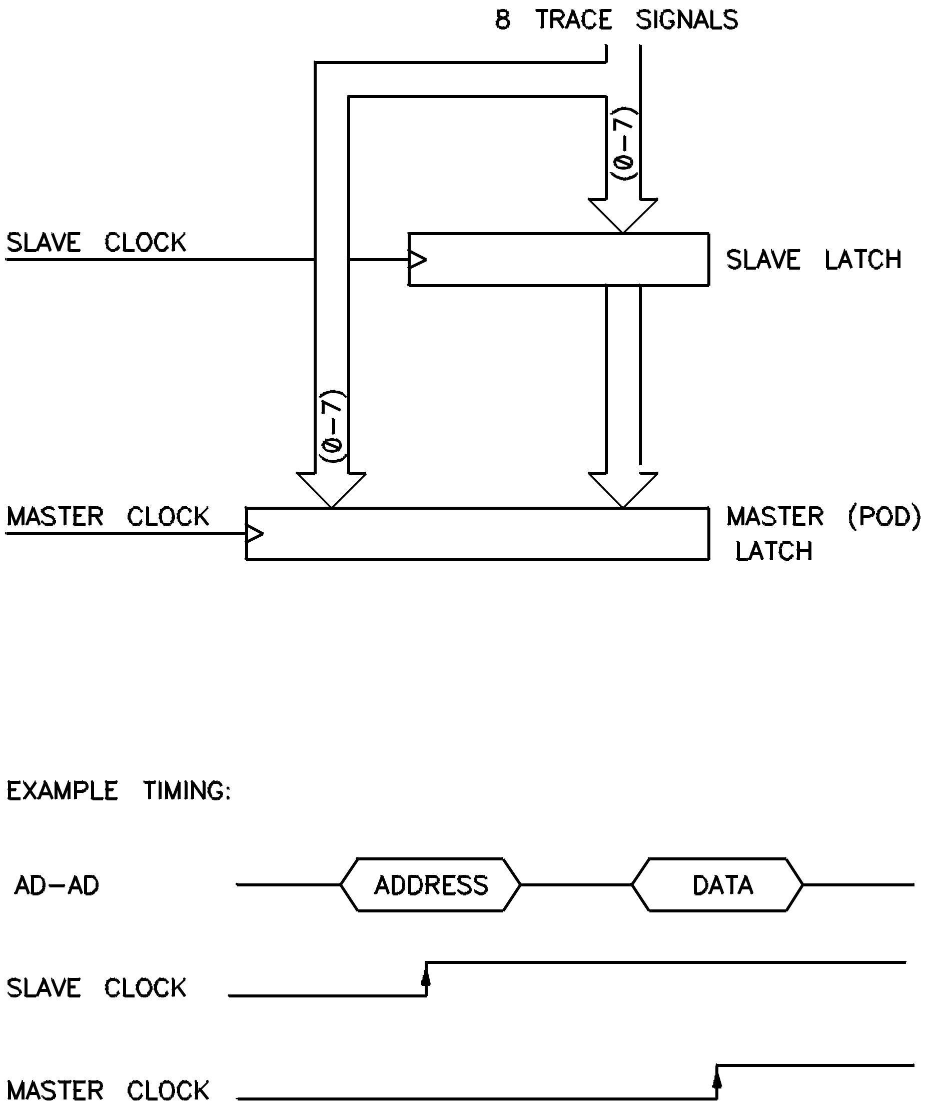
If no slave clock has appeared since the last master clock, the data on the lower 8 bits of the pod will be latched at the same time as the upper 8 bits. If more than one slave clock has appeared since the last master clock, only the first slave data will be available to the analyzer (see the figure below).

Figure 26


MASTER CLOCK
SLAVE CLOCK
DATA LATCHED ON FIRST SLAVE CLOCK AFTER LAST MASTER CLOCK
FOLLOWING SLAVE CLOCKS IGNORED

Choose "demux" to specify the true demultiplexing mode. In this mode, only the lower eight external channels (0-7) are used. The slave clock (as specified by your answers to the next four questions) latches these bits and the emulation clock samples the same channels again. The latched bits show up as bits 0-7 in the trace data, and the second sample shows up as bits 8-15 (see the figure below).

Figure 27



If no slave clock has appeared since the last master clock, the data on the lower 8 bits of the pod will be the same as the upper 8 bits. If more than one slave clock has appeared since the last master clock, only the first slave data will be available to the analyzer.

4 If the "mixed" or "true demultiplexing" slave clock modes are selected, choose "None", "rising", "falling", or "Both" for the "Edges of J (K,L,M) clock used for slave clock" configuration options.

Four configuration options are present when you select either the "mixed" or "demux" slave clock mode. They allow you to define the slave clock. You can specify rising, falling, both, or neither (none) edges of the J, K, L, and M clocks. When several clock edges are specified, any one of the edges clocks the trace.

Clocks J and K are the external clock inputs of the external analyzer probe. The L and M clocks are generated by the emulator. Typically, the L clock is the emulation clock derived by the emulator and the M clock is not used.

To define labels for the external analyzer signals

1 Choose "yes" for the "Should emulation control the external bits" configuration option.

2 For each defined external label (there can be up to 8), choose the "name", "start bit", "width", and "polarity".

You can define up to eight labels for the 16 external data channels in the configuration. These external analyzer labels can be used in trace commands, and the data associated with these labels can be displayed in the trace list. One external analyzer label, "xbits", is defined by the default configuration and is included in the default trace list.

External labels can be defined with bits in the range of 0 through 15. The start bit may be in the range 0 through 15, but the width of the label must not cause the label to extend past bit 15. Thus, the sum of the start bit number plus the width must not exceed 16.

The "polarity" configuration option allows you to specify positive or negative logic for the external bits. In other words, positive means high=1, low=0. Negative means low=1, high=0.

Once external labels are defined, they may be used in trace commands to qualify events (if the emulation controls the external analyzer). Also, you can modify the trace display to include data for the various trace labels.

Note that the Timing Analyzer Interface does not use the external labels defined in the emulator/analyzer interface. You maintain labels for the timing analyzer within the Timing Analyzer Interface itself.

10

Making Coordinated Measurements

Making Coordinated Measurements

When HP 64700 Card Cages are connected together via the Coordinated Measurement Bus (CMB), you can start and stop up to 32 emulators at the same time.

You can use the analyzer in one HP 64700 to arm (activate) the analyzers in other HP 64700 Card Cages or to cause emulator execution in other HP 64700 Card Cages to break into the monitor.

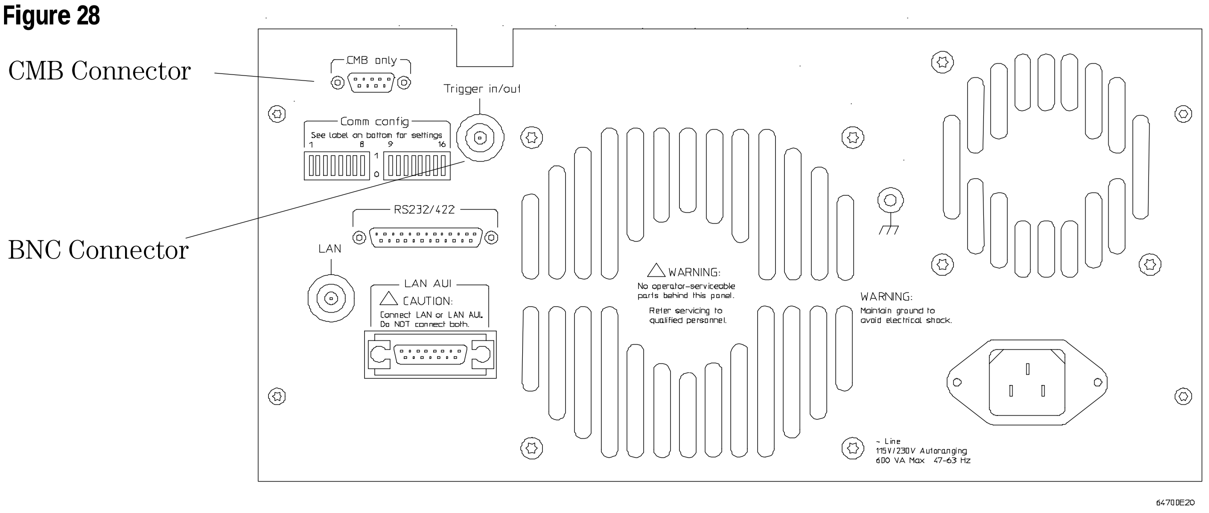
You can use the HP 64700's BNC connector (labeled TRIGGER IN/OUT on the lower left corner of the HP 64700 rear panel) to trigger an external instrument (for example, a logic analyzer or oscilloscope) when the analyzer finds its trigger condition. Also, you can allow an external instrument to arm the analyzer or break emulator execution into the monitor.

The coordinated measurement tasks you can perform are grouped into the following sections:

- Setting up for coordinated measurements.
- Starting and stopping multiple emulators.
- Driving trigger signals to the CMB or BNC.
- Stopping program execution on trigger signals.
- Arming analyzers on trigger signals.

The location of the CMB and BNC connectors on the HP 64700 rear panel is shown in the following figure.

Figure 28



Signal Lines on the CMB

There are three bi-directional signal lines on the CMB connector on the rear panel of the emulator. These CMB signals are:

TRIGGER The CMB TRIGGER line is low true. This signal can be driven or received by any HP 64700 connected to the CMB. This signal can be used to trigger an analyzer. It can be used as a break source for the emulator.

READY The CMB READY line is high true. It is an open collector and performs an ANDing of the ready state of enabled emulators on the CMB. Each emulator on the CMB releases this line when it is ready to run. This line goes true when all enabled emulators are ready to run, providing for a synchronized start.

When CMB is enabled, each emulator is required to break to background when CMB READY goes false, and will wait for CMB READY to go true before returning to the run state. When an enabled emulator breaks, it will drive the CMB READY false and will hold it false until it is ready to resume running. When an emulator is reset, it also drives CMB READY false.

EXECUTE The CMB EXECUTE line is low true. Any HP 64700 on the CMB can drive this line. It serves as a global interrupt and is processed by both the emulator and the analyzer. This signal causes an emulator to run from a specified address when CMB READY returns true.

BNC Trigger Signal

The BNC trigger signal is a positive rising edge TTL level signal. The BNC trigger line can be used to either drive or receive an analyzer trigger, or receive a break request for the emulator.

Comparison Between CMB and BNC Triggers The CMB trigger and BNC trigger lines have the same logical purpose: to provide a means for connecting the internal trigger signals (trig1 and trig2) to external instruments. The CMB and BNC trigger lines are bi-directional. Either signal may be used directly as a break condition.

The CMB trigger is level-sensitive, while the BNC trigger is edge-sensitive. The CMB trigger line puts out a true pulse following receipt of EXECUTE, despite the commands used to configure it. This pulse is internally ignored.

Note that if you use the EXECUTE function, the CMB TRIGGER should not be used to trigger external instruments, because a false trigger will be generated when EXECUTE is activated.

Setting Up for Coordinated Measurements

This section describes how to:

- Connect the Coordinated Measurement Bus.
- Connect the rear panel BNC.

To connect the Coordinated Measurement Bus (CMB)

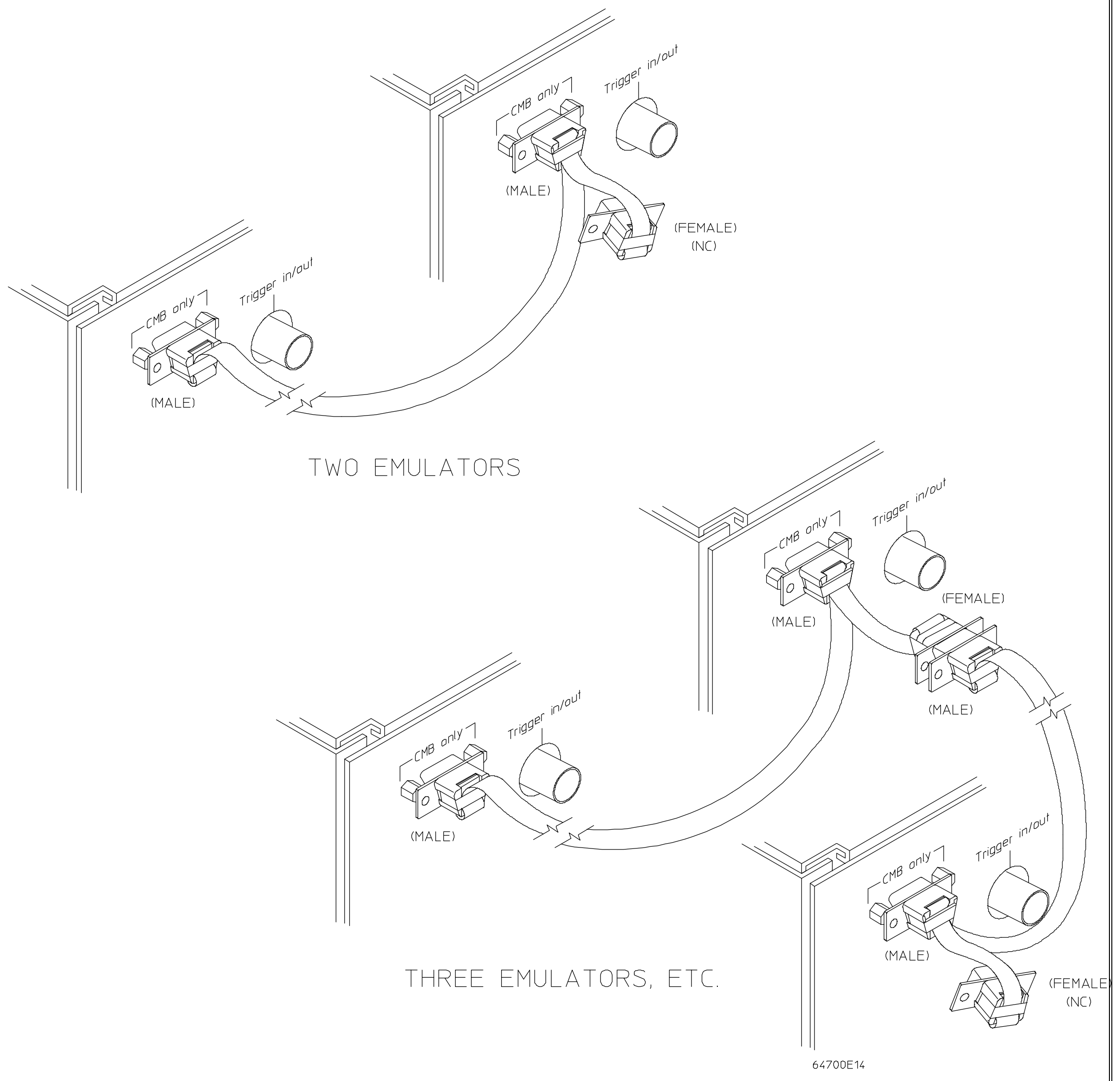
CAUTION Be careful not to confuse the 9-pin connector used for CMB with those used by some computer systems for RS-232-C communications. Applying RS-232-C signals to the CMB connector is likely to result in damage to the HP 64700 Card Cage.

To use the CMB, you will need one CMB cable for the first two emulators and one additional cable for every emulator after the first two. The CMB cable is orderable from HP under product number HP 64023A. The cable is four meters long.

You can build your own compatible CMB cables using standard 9-pin D type subminiature connectors and 26 AWG wire.

Note that Hewlett-Packard does not ensure proper CMB operation if you are using a self-built cable!

1 Connect the cables to the HP 64700 CMB ports.



Number of HP 64700 Series Emulators	Maximum Total Length of Cable	Restrictions on the CMB Connection
2 to 8	100 meters	None.
9 to 16	50 meters	None.
9 to 16	100 meters	Only 8 emulators may have rear panel pullups connected. *
17 to 32	50 meters	Only 16 emulators may have rear panel pullups connected. *

* A modification must be performed by your HP Customer Engineer.

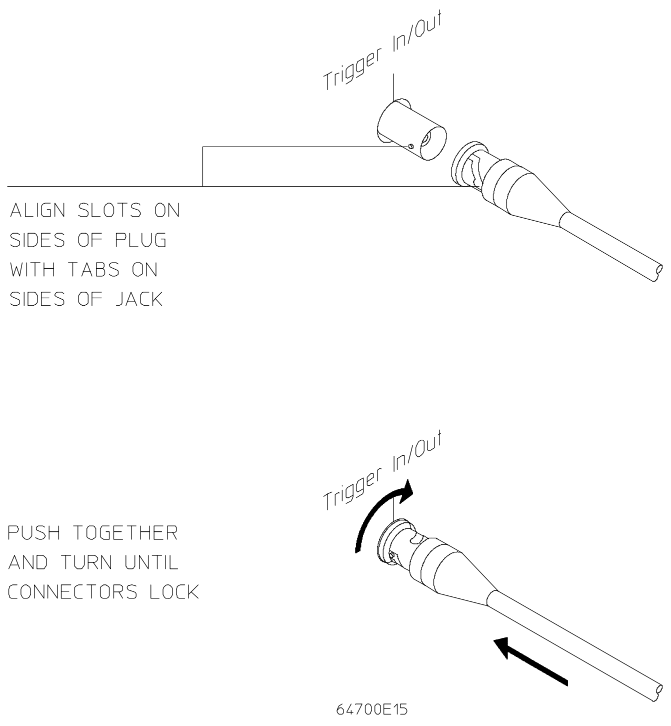
Emulators using the CMB must use background emulation monitors.

At least 3/4 of the HP 64700-Series emulators connected to the CMB must be powered up before proper operation of the entire CMB configuration can be assured.

To connect to the rear panel BNC

CAUTION **The BNC line on the HP 64700 accepts input and output of TTL levels only. (TTL levels should not be less than 0 volts or greater than 5 volts.) Failure to observe these specifications may result in damage to the HP 64700 Card Cage.**

1 Connect one end of a 50-ohm coaxial cable with male BNC connectors to the HP 64700 BNC receptacle and the other end to the appropriate BNC receptacle on the other measuring instrument.


Trigger In/Out
ALIGN SLOTS ON SIDES OF PLUG WITH TABS ON SIDES OF JACK
Trigger In/Out
PUSH TOGETHER AND TURN UNTIL CONNECTORS LOCK
64700E15

The BNC connector is capable of driving TTL level signals into a 50-ohm load. (A positive rising edge is the trigger signal.) It requires a driver that can supply at least 4 mA at 2 volts when used as a receiver. The BNC connector is configured as an open-emitter structure which allows for multiple drivers to be connected. It can be used for cross-triggering between multiple HP 64700Bs when no other cross-measurements are needed. The output of the BNC connector is short-circuit protected and is protected from TTL level signals when the emulator is powered down.

Starting/Stopping Multiple Emulators

When HP 64700 Card Cages are connected together via the Coordinated Measurement Bus (CMB), you can start and stop up to 32 emulators at the same time. These are called synchronous measurements.

This section describes how to:

- Enable synchronous measurements.
- Start synchronous measurements.
- Disable synchronous measurements.

To enable synchronous measurements

- Enter the **specify run** command.

You can enable the emulator's interaction with the CMB by using the ***specify run*** command. When the EXECUTE signal is received, the emulator will run at the current program counter address or the address specified in the specify run command.

Note that when the CMB is being actively controlled by another emulator, the step command does not work correctly. The emulator may end up running in user code (NOT stepping). Disable CMB interaction (see "To disable synchronous measurements" below) while stepping the processor.

Note that enabling CMB interaction does not affect the operation of analyzer cross-triggering.

You can use the ***specify trace*** command to specify that an analyzer measurement begin upon reception of the CMB EXECUTE signal.

The trace measurement defined by the ***specify trace*** command will be started when the EXECUTE signal becomes active. When the trace measurement begins, you will see the message "CMB execute; emulation trace started".

When you enter a normal ***trace*** command, trace at execute is disabled, and the analyzer ignores the CMB EXECUTE signal.

Examples

To enable synchronous measurements:

```
specify run from 1e8h
```

To trace when synchronous execution begins:

```
specify trace after address main
```

To start synchronous measurements

- Enter the **cmb_execute** command.

The ***cmb_execute*** command will cause the EXECUTE line to be pulsed, thereby initiating a synchronous measurement. CMB interaction does not have to be enabled in order to use either of these commands. (When you enable CMB interaction, you only specify how the emulator will react to the CMB EXECUTE signal.)

All emulators whose CMB interaction is enabled will break into the monitor when any one of those emulators breaks into its monitor.

To disable synchronous measurements

- Enter the **specify run disable** command.

You can disable the emulator's interaction with the CMB by using the ***specify run disable*** command. When interaction is disabled, the emulator ignores the CMB EXECUTE and READY lines.

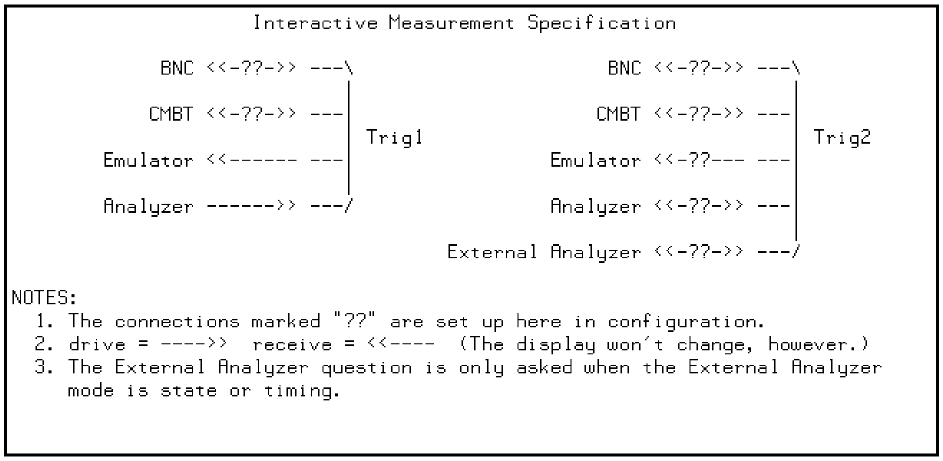
Using Trigger Signals

The HP 64700 contains two internal lines, trig1 and trig2, over which trigger signals can pass from the emulator or analyzer to other HP 64700s on the Coordinated Measurement Bus (CMB) or other instruments connected to the BNC connector.

You can configure the internal lines to make connections between the emulator, analyzer, external analyzer (if it is configured as an independent state or timing analyzer), CMB connector, or BNC connector. Measurements that depend on these connections are called *interactive measurements* or *coordinated measurements*.

This figure below illustrates the possible connections between the internal lines (trig1 and trig2) and the emulator, analyzer, and external devices.

Figure 29



```
                     Interactive Measurement Specification

         BNC <<-??->> ---\                        BNC <<-??->> ---\
                         |                                        |
        CMBT <<-??->> ---|                       CMBT <<-??->> ---|
                         | Trig1                                  | Trig2
    Emulator <<------ ---|                   Emulator <<-??--- ---|
                         |                                        |
    Analyzer ------>> ---/                   Analyzer <<-??->> ---|
                                                                  |
                                    External Analyzer <<-??->> ---/

NOTES:
  1. The connections marked "??" are set up here in configuration.
  2. drive = ---->>  receive = <<----  (The display won't change, however.)
  3. The External Analyzer question is only asked when the External Analyzer
     mode is state or timing.
```

Note that the "External Analyzer" connection for "Trig2" is only available if you have selected "state" or "timing" for the external analyzer mode.

Notice that the analyzer always drives trig1, and the emulator always receives trig1. This provides for the ***break_on_trigger*** syntax of the ***trace*** command.

You can use the trig1 or trig2 line to make a connection between the analyzer and the CMB connector or BNC connector so that, when the analyzer finds its trigger condition, a trigger signal is driven on the HP 64700's Coordinated Measurement Bus (CMB) or BNC connector. This can also be done for the external analyzer when it is configured as an independent state or timing analyzer.

You can use the trig1 or trig2 line to make a connection between the emulator break input and the CMB connector, BNC connector, analyzer, (or external analyzer when configured as an independent state or timing analyzer) so that program execution can break when a trigger signal is received from the CMB, BNC, or analyzer.

You can use the trig2 line to make a connection between the analyzer and the CMB connector or BNC connector so that the analyzer can be armed (that is, enabled) when a trigger signal is received from the CMB or BNC connector. This can also be done for the external analyzer when it is configured as an independent state or timing analyzer.

You can use the trig1 and trig2 lines to make several type of connections at the same time. For example, when the analyzer finds its trigger condition, a signal is driven on the trig1 line. This signal may be used to stop user program execution, but the trigger signal may also be driven on the CMB and BNC connectors.

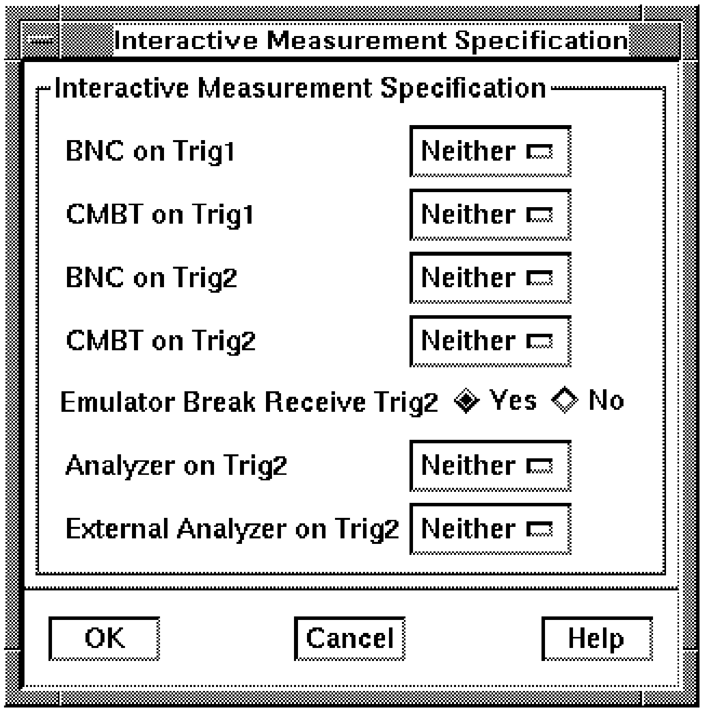
Also, it's possible for signals to be driven and received on the CMB or BNC connectors. So, for example, while the analyzer's trigger signal can be driven on the CMB and BNC connectors, signals can also be received from the CMB and BNC connectors and used to stop user program execution. In this case, the emulator will break into the monitor on either the analyzer trigger or on the reception of a trigger signal from the CMB or BNC.

You can disable connections made by the internal trig1 and trig2 lines by choosing "neither" or "no" to the appropriate interactive measurement configuration options.

In order to modify the interactive measurement specification, you must first start the configuration interface and access the "Interactive

Measurement Specification" configuration section (refer to the "Using the Configuration Interface" section in the "Configuring the Emulator" chapter).

Figure 30



Interactive Measurement Specification

Interactive Measurement Specification

BNC on Trig1	Neither
CMBT on Trig1	Neither
BNC on Trig2	Neither
CMBT on Trig2	Neither
Emulator Break Receive Trig2	[✓ Yes] No
Analyzer on Trig2	Neither
External Analyzer on Trig2	Neither

If you're using the Softkey Interface from a terminal or terminal emulation window, you don't get a dialog box from which to choose configuration sections; however, you have access to the same configuration options through a series of configuration questions. To access the questions in the "Interactive Measurement Specification" section, answer "yes" to the "Modify interactive measurement specification?" question.

This section shows you how to:

- Drive the emulation analyzer trigger signal to the CMB.
- Drive the emulation analyzer trigger signal to the BNC connector.
- Drive the external analyzer trigger signal to the CMB.

- Drive the external analyzer trigger signal to the BNC connector.
- Break emulator execution on signal from CMB.
- Break emulator execution on signal from BNC.
- Break emulator execution on external analyzer trigger.
- Arm the emulation analyzer on signal from CMB.
- Arm the emulation analyzer on signal from BNC.
- Arm the emulation analyzer on external analyzer trigger.
- Arm the external analyzer on signal from CMB.
- Arm the external analyzer on signal from BNC.
- Arm the external analyzer on emulation analyzer trigger.

To drive the emulation analyzer trigger signal to the CMB

- **Choose "receive" for the "Should CMBT drive or receive Trig1" configuration option.**

You could also drive the emulation analyzer trigger to the CMB over the trig2 internal line by specifying that the CMBT should receive trig2 and that the emulation analyzer should drive trig2.

To drive the emulation analyzer trigger signal to the BNC connector

- **Choose "receive" for the "Should BNC drive or receive Trig1" configuration option.**

You could also drive the emulation analyzer trigger to the BNC over the trig2 internal line by specifying that the BNC should receive trig2 and that the emulation analyzer should drive trig2.

To drive the external analyzer trigger signal to the CMB

1 Choose "receive" for the "Should CMBT drive or receive Trig2" configuration option.

2 Choose "drive" for the "Should External Analyzer drive or receive Trig2" configuration option.

To drive the external analyzer trigger signal to the BNC connector

1 Choose "receive" for the "Should BNC drive or receive Trig2" configuration option.

2 Choose "drive" for the "Should External Analyzer drive or receive Trig2" configuration option.

To break emulator execution on signal from CMB

- Choose "drive" for the "Should CMBT drive or receive Trig1" configuration option.

  You could also break emulator execution on a trigger signal from the CMB over the trig2 internal line by specifying that the CMB should drive trig2 and that the emulator break should receive trig2.

To break emulator execution on signal from BNC

- Choose "drive" for the "Should BNC drive or receive Trig1" configuration option.

  You could also break emulator execution on a trigger signal from the BNC over the trig2 internal line by specifying that the BNC should drive trig2 and that the emulator break should receive trig2.

To break emulator execution on external analyzer trigger

1 Choose "yes" for the "Should Emulator break receive Trig2" configuration option.

2 Choose "drive" for the "Should External Analyzer drive or receive Trig2" configuration option.

When an emulator break occurs due to the analyzer trigger, the analyzer will stop driving the internal signal that caused the break. Therefore, if trig2 is used both to break and to drive the CMB TRIGGER (for example), TRIGGER will go true when the trigger is found and then will go false after the emulator breaks. However, if trig1 is used to cause the break and trig2 is used to drive the CMB TRIGGER, TRIGGER will stay true until the trace is halted or until the next trace starts.

To arm the emulation analyzer on signal from CMB

1 Choose "drive" for the "Should CMBT drive or receive Trig2" configuration option.

2 Choose "receive" for the "Should Analyzer drive or receive Trig2" configuration option.

3 Use the **arm_trig2** option to the **trace** command.

To arm the emulation analyzer on signal from BNC

1 Choose "drive" for the "Should BNC drive or receive Trig2" configuration option.

2 Choose "receive" for the "Should Analyzer drive or receive Trig2" configuration option.

3 Use the **arm_trig2** option to the **trace** command.

To arm the emulation analyzer on external analyzer trigger

1 Choose "receive" for the "Should Analyzer drive or receive Trig2" configuration option.
2 Choose "drive" for the "Should External Analyzer drive or receive Trig2" configuration option.
3 Use the **arm_trig2** option to the **trace** command.

To arm the external analyzer on signal from CMB

1 Choose "drive" for the "Should CMBT drive or receive Trig2" configuration option.
2 Choose "receive" for the "Should External Analyzer drive or receive Trig2" configuration option.

To arm the external analyzer on signal from BNC

1 Choose "drive" for the "Should BNC drive or receive Trig2" configuration option.
2 Choose "receive" for the "Should External Analyzer drive or receive Trig2" configuration option.

To arm the external analyzer on emulation analyzer trigger

1 Choose "drive" for the "Should Analyzer drive or receive Trig2" configuration option.
2 Choose "receive" for the "Should External Analyzer drive or receive Trig2" configuration option.

11

Setting X Resources

Setting X Resources

The Graphical User Interface is an X Window System application which means it is a *client* in the X Window System client-server model.

The X server is a program that controls all access to input devices (typically a mouse and a keyboard) and all output devices (typically a display screen). It is an interface between application programs you run on your system and the system input and output devices.

An X *resource* controls an element of appearance or behavior in an X application. For example, in the graphical interface, one resource controls the text in action key pushbuttons as well as the action performed when the pushbutton is clicked.

By modifying resource settings, you can change the appearance or behavior of certain elements in the graphical interface.

When the graphical interface starts up, it reads resource specifications from a set of configuration files. Resources specifications in later files override those in earlier files. Files are read in the following order:

1 The application defaults file. For example, /usr/lib/X11/app-defaults/HP64_Softkey in HP-UX or /usr/openwin/lib/X11/app-defaults/HP64_Softkey in SunOS.

2 The $XAPPLRESDIR/HP64_Softkey file. (The XAPPLRESDIR environment variable defines a directory containing system-wide custom application defaults.)

3 The server's RESOURCE_MANAGER property. (The **xrdb** command loads user-defined resource specifications into the RESOURCE_MANAGER property.)

   If no RESOURCE_MANAGER property exists, user defined resource settings are read from the $HOME/.Xdefaults file.

4 The file named by the XENVIRONMENT environment variable.

   If the XENVIRONMENT variable is not set, the $HOME/.Xdefaults-*host* file (typically containing resource specifications for a specific remote host) is read.

5 Resource specifications included in the command line with the **-xrm** option.
6 System scheme files in directory /usr/hp64000/lib/X11/HP64_schemes.
7 System-wide custom scheme files located in directory $XAPPLRESDIR/HP64_schemes.
8 User-defined scheme files located in directory $HOME/.HP64_schemes (note the dot in the directory name).

Scheme files group resource specifications for different displays, computing environments, and languages.

This chapter shows you how to:

- Modify the Graphical User Interface resources.
- Use customized scheme files.
- Set up custom action keys.
- Set initial recall buffer values.
- Set up demos or tutorials.

Refer to the "X Resources and the Graphical Interface" section in the "Concepts" chapter for more detailed information.

To modify the Graphical User Interface resources

You can customize the appearance of an X Windows application by modifying its X resources. The following tables describe some of the commonly modified application resources.

Application Resources for Schemes		
Resource	Values	Description
HP64_Softkey.platformScheme	HP-UX SunOS (custom)	Names the subdirectory for platform specific schemes. This resource should be set to the platform on which the X server is running (and displaying the Graphical User Interface) if it is different than the platform where the application is running.
HP64_Softkey.colorScheme	BW Color (custom)	Names the color scheme file.
HP64_Softkey.sizeScheme	Small Large (custom)	Names the size scheme file which defines the fonts and the spacing used.
HP64_Softkey.labelScheme	Label $LANG (custom)	Names to use for labels and button text. The default uses the $LANG environment variable if it is set and if a scheme file named Softkey.$LANG exists in one of the directories searched for scheme files; otherwise, the default is Label.
HP64_Softkey.inputScheme	Input (custom)	Specifies mouse and keyboard operation.

Commonly Modified Application Resources		
Resource	Values	Description
HP64_Softkey.lines	24 (min. 18)	Specifies the number of lines in the main display area.
HP64_Softkey.columns	100 (min. 80)	Specifies the number of columns, in characters, in the main display area.
HP64_Softkey.enableCmdline	True False	Specifies whether the command line area is displayed when you initially enter the Graphical User Interface.
*editFile	(example) vi %s	Specifies the command used to edit files.
*editFileLine	(example) vi +%d %s	Specifies the command used to edit a file at a certain line number.
*<proc>*actionKeysSub.keyDefs	(paired list of strings)	Specifies the text that should appear on the action key push buttons and the commands that should be executed in the command line area when the action key is pushed. Refer to the "To set up custom action keys" section for more information.
*<proc>*dirSelectSub.entries	(list of strings)	Specifies the initial values that are placed in the File→Context→Directory pop-up recall buffer. Refer to the "To set initial recall buffer values" section for more information.
*<proc>*recallSub.entries	(list of strings)	Specifies the initial values that are placed in the entry buffer (labeled "():"). Refer to the "To set initial recall buffer values" section for more information.

The following steps show you how to modify the Graphical User Interface's X resources.

1 Copy part or all of the HP64_Softkey application defaults file to a temporary file.

The HP64_Softkey file contains the default definitions for the graphical interface application's X resources.

For example, on an HP 9000 computer you can use the following command to copy the complete HP64_Softkey file to HP64_Softkey.tmp (note that the HP64_Softkey file is several hundred lines long):

```
cp /usr/lib/X11/app-defaults/HP64_Softkey HP64_Softkey.tmp
```

NOTE: The HP64_Softkey application defaults file is re-created each time Graphical User Interface software is installed or updated. You can use the UNIX diff command to check for differences between the new HP64_Softkey application defaults file and the old application defaults file that is saved as /usr/hp64000/lib/X11/HP64_schemes/old/HP64_Softkey.

2 Modify the temporary file.

Modify the resource that defines the behavior or appearance that you wish to change.

For example, to change the number of lines in the main display area to 36:

```
vi HP64_Softkey.tmp
```

Search for the string "HP64_Softkey.lines". You should see lines similar to the following.

```
!-----------------------------------------------------------------------------
! The lines and columns set the vertical and horizontal dimensions of the
! main display area in characters, respectively.  Minimum values are 18 lines
! and 80 columns.  These minimums are silently enforced.
!
! Note: The application cannot be resized by using the window manager.

!HP64_Softkey.lines:    24
!HP64_Softkey.columns:  85
```

Edit the line containing "HP64_Softkey.lines" so that it is uncommented and is set to the new value:

```
!-----------------------------------------------------------------------------
! The lines and columns set the vertical and horizontal dimensions of the
! main display area in characters, respectively.  Minimum values are 18 lines
! and 80 columns.  These minimums are silently enforced.
!
! Note: The application cannot be resized by using the window manager.

HP64_Softkey.lines:    36
!HP64_Softkey.columns:  85
```

Save your changes and exit the editor.

3 If the RESOURCE_MANAGER property exists (as is the case with HP VUE — if you're not sure, you can check by entering the **xrdb -query** command), use the **xrdb** command to add the resources to the RESOURCE_MANAGER property. For example:

xrdb -merge -nocpp HP64_Softkey.tmp

Otherwise, if the RESOURCE_MANAGER property does not exist, append the temporary file to your $HOME/.Xdefaults file. For example:

cat HP64_Softkey.tmp >> $HOME/.Xdefaults

4 Remove the temporary file.

5 Start or restart the Graphical User Interface.

After you have completed the above steps, you must either start, or restart by exiting and starting again, the Graphical User Interface. Starting and exiting the Graphical User Interface is described in the "Starting and Exiting HP 64700 Interfaces" chapter.

To use customized scheme files

Scheme files are used to set platform specific resources that deal with color, fonts and sizes, mouse and keyboard operation, and labels and titles. You can create and use customized scheme files by following these steps.

1 Create the $HOME/.HP64_schemes/<platform> directory.

For example:

```
mkdir $HOME/.HP64_schemes
mkdir $HOME/.HP64_schemes/HP-UX
```

2 Copy the scheme file to be modified to the $HOME/.HP64_schemes/<platform> directory.

Label scheme files are not platform specific; therefore, they should be placed in the $HOME/.HP64_schemes directory. All other scheme files should be placed in the $HOME/.HP64_schemes/<platform> directory.

For example:

```
cp /usr/hp64000/lib/X11/HP64_schemes/HP-UX/Softkey.Color
$HOME/.HP64_schemes/HP-UX/Softkey.MyColor
```

Note that if your custom scheme file has the same name as the default scheme file, the load order requires resources in the custom file to explicitly override resources in the default file.

3 Modify the $HOME/.HP64_schemes/<platform>/Softkey.<scheme> file.

For example, you could modify the "$HOME/.HP64_schemes/HP-UX/Softkey.MyColor" file to change the defined foreground and background colors. Also, since the scheme file name is different than the default, you could comment out various resource settings to cause general foreground and background color definitions to apply to the Graphical User Interface. At least one resource must be defined in your color scheme file for it to be recognized.

4 If your custom scheme file has a different name than the default, you must modify the scheme resource definitions.

The Graphical User Interface application defaults file contains resources that specify which scheme files are used. If your custom scheme files are named differently than the default scheme files, you must modify these resource settings so that your customized scheme files are used instead of the default scheme files.

For example, to use the "$HOME/.HP64_schemes/HP-UX/Softkey.MyColor" color scheme file you would set the "HP64_Softkey.colorScheme" resource to "MyColor":

```
HP64_Softkey.colorScheme: MyColor
```

Refer to the previous "To customize Graphical User Interface resources" section for more detailed information on modifying resources.

To set up custom action keys

- **Modify the "actionKeysSub.keyDefs" resource.**

The "actionKeysSub.keyDefs" resource defines a list of paired strings. The first string defines the text that should appear on the action key pushbutton. The second string defines the command that should be sent to the command line area and executed when the action key is pushed.

A pair of parentheses (with no spaces, that is "()") can be used in the command definition to indicate that text from the entry buffer should replace the parentheses when the command is executed.

Action keys that use the entry buffer should always include the entry buffer symbol, "( )", in the action key label as a visual cue to remind you to place information in the entry buffer before clicking the action key.

Shell commands can be executed by using an exclamation point prefix. A second exclamation point ends the command string and allows additional options on the command line.

Also, command files can be executed by placing the name of the file in the command definition.

Finally, an empty action ("") means to repeat the previous operation, whether it came from a pull-down, a dialog, a pop-up, or another action key.

Examples

To set up custom action keys when the graphical interface is used with 6830x emulators, modify the "*m6830x*actionKeysSub.keyDefs" resource:

```
*m6830x*actionKeysSub.keyDefs: \
   "Make"              "cd /users/project2/6830x; !make! in_browser" \
   "Load Pgm"          "load configuration config.EA; load program2" \
   "Run Pgm"           "run from reset" \
   "Trace after ( )"   "trace after (); display trace" \
   "Step Source"       "set source on; display memory mnemonic; step source" \
   "Again"             ""
```

Refer to the previous "To modify Graphical User Interface resources" section for more detailed information on modifying resources.

To set initial recall buffer values

- **Modify the "entries" resource for the particular recall buffer.**

There are six pop-up recall buffers present in the Graphical User Interface. The resources for these pop-up recall buffers are listed in the following table.

The window manager resource "*transientDecoration" controls the borders around dialog box windows. The most natural setting for this resource is "title."

Popup Recall Buffer Resources		
Recall Popup	Resources	Description
File→Context→Directory ...	*dirSelect.textColumns *dirSelect.listVisibleItemCount *dirSelectSub.entries	The default number of text columns in the pop-up is 50. The default number of visible lines in the pop-up is 12. The "entries" resource is defined as a list of strings (see the following example). Up to 40 unique values are saved in each of the recall buffers (as specified by the resource settings "*XcRecall.maxDepth: 40" and "*XcRecall.onlyUnique: True").
File→Context→Symbols ...	*symSelect.textColumns *symSelect.listVisibleItemCount *symSelectSub.entries	
Trace→Trace Spec ...	*modtrace.textColumns *modtrace.listVisibleItemCount *modtraceSub.entries	
Entry Buffer ():	*recall.textColumns *recall.listVisibleItemCount *recallSub.entries	
Command Line command recall	*recallCmd.textColumns *recallCmd.listVisibleItemCount *recallCmdSub.entries	
Command Line pod/simio recall	*recallKbd.textColumns *recallKbd.listVisibleItemCount *recallKbdSub.entries	

Examples

To set the initial values for the directory selection dialog box when the Graphical User Interface is used with 6830x emulators, modify the "*m6830x*dirSelectSub.entries" resource:

```
*m6830x*dirSelectSub.entries: \
   "$HOME" \
   ".." \
   "/users/project1" \
   "/users/project2/6830x"
```

Refer to the previous "To modify the Graphical User Interface resources" section for more detailed information on modifying resources.

To set up demos or tutorials

You can add demos or tutorials to the Graphical User Interface by modifying the resources described in the following tables.

Demo Related Component Resources		
Resource	Value	Description
*enableDemo	False True	Specifies whether Help→Demo appears in the pull-down menu.
*demoPopupSub.indexFile	./Xdemo/Index-topics	Specifies the file containing the list of topic and file pairs.
*demoPopup.textColumns	30	Specifies the width, in characters, of the of the demo topic list pop-up.
*demoPopup.listVisibleItemCount	10	Specifies the length, in lines, of the demo topic list pop-up.
*demoTopic	About demos	Specifies the default topic in the demo pop-up selection buffer.

Tutorial Related Component Resources		
Resource	Value	Description
*enableTutorial	False True	Specifies whether Help→Tutorial appears in the pull-down menu.
*tutorialPopupSub.indexFile	./Xtutorial/Index-topics	Specifies the file containing the list of topic and file pairs.
*tutorialPopup.textColumns	30	Specifies the width, in characters, of the of the tutorial topic list pop-up.
*tutorialPopup.listVisibleItemCount	10	Specifies the length, in lines, of the tutorial topic list pop-up.
*tutorialTopic	About tutorials	Specifies the default topic in the tutorial pop-up selection buffer.

The mechanism for providing demos and tutorials in the graphical interface is identical. The following steps show you how to set up demos or tutorials in the Graphical User Interface.

1 Create the demo or tutorial topic files and the associated command files.

Topic files are simply ASCII text files. You can use "\I" to produce inverse video in the text, "\U" to produce underlining in the text, and "\N" to restore normal text.

Command files are executed when the "Press to perform demo (or tutorial)" button (in the topic pop-up dialog) is pushed. A command file must have the same name as the topic file with ".cmd" appended. Also, a command file must be in the same directory as the associated topic file.

2 Create the demo or tutorial index file.

Each line in the index file contains first a quoted string that is the name of the topic which appears in the index pop-up and second the name of the file that is raised when the topic is selected. For example:

```
"About demos"       /users/guest/gui_demos/general
"Loading programs"  /users/guest/gui_demos/loadprog
"Running programs"  /users/guest/gui_demos/runprog
```

You can use absolute paths (for example, /users/guest/topic1), paths relative to the directory in which the interface was started (for example, mydir/topic2), or paths relative to the product directory (for example, ./Xdemo/general where the product directory is something like /usr/hp64000/inst/emul/64742A).

3 **Set the "*enableDemo" or "*enableTutorial" resource to "True".**

4 **Define the demo index file by setting the "*demoPopupSub.indexFile" or "*tutorialPopupSub.indexFile" resource.**

For example:

```
*demoPopupSub.indexFile:  /users/guest/gui_demos/index
```

You can use absolute paths (for example, /users/guest/Index), paths relative to the directory in which the interface was started (for example, mydir/indexfile), or paths relative to the product directory (for example, ./Xdemo/Index-topics where the product directory is something like /usr/hp64000/inst/emul/64742A).

5 **If you wish to define a default topic to be selected, set the "*demoTopic" or "*tutorialTopic" resource to the topic string.**

For example:

```
*demoTopic:  "About demos"
```

Refer to the previous "To customize Graphical User Interface resources" section for more detailed information on modifying resources.

Part 3

Reference

Part 3

Descriptions of the product in a dictionary or encyclopedia format.

12

Emulator/Analyzer Interface Commands

Emulator/Analyzer Interface Commands

This chapter describes the emulator/analyzer interface commands in alphabetical order. First, the syntax conventions are described and the commands are summarized.

How Pull-down Menus Map to the Command Line

Pull-down menu items and corresponding softkey commands are shown below.

Pull-down	Command Line
File→Context→Directory	cd
File→Context→Symbols	cws
File→Load→Emulator Config	load configuration
File→Load→Executable	load <abs_file>
File→Load→Program Only	load <abs_file> nosymbols
File→Load→Symbols Only	load symbols
File→Load→Trace Data	load trace <FILE>
File→Load→Trace Spec	load trace <FILE>
File→Store→Trace Data	store trace
File→Store→Trace Spec	store trace_spec
File→Store→BBA Data	bbaunload
File→Copy→Display	copy display to
File→Copy→Memory	copy memory to
File→Copy→Data Values	copy data to
File→Copy→Configuration Info	copy configuration_info to
File→Copy→Trace	copy trace to
File→Copy→Registers	copy registers to
File→Copy→Breakpoints	copy software_breakpoints to
File→Copy→Status	copy status to
File→Copy→Global Symbols	copy global_symbols to
File→Copy→Local Symbols ()	copy local_symbols_in --SYMB-- to
File→Copy→Pod Commands	copy pod_command to
File→Copy→Error Log	copy error_log to
File→Copy→Event Log	copy event_log to

Pull-down	Command Line
File→Log→Playback	<command file>
File→Log→Record	log_commands to
File→Log→Stop	log_commands off
File→Emul700→Performance Analyzer	N/A
File→Emul700→Emulator/Analyzer	N/A
File→Emul700→Timing Analyzer	N/A
File→Emul700→SPA for VRTXsa	N/A
File→Emul700→SPA for VRTX	N/A
File→Edit→File	! vi <file> ! no_prompt_before_exit
File→Edit→At () Location	! vi +<line> <file> ! no_prompt_before_exit
File→Edit→At PC Location	! vi +<line> <file> ! no_prompt_before_exit
File→Term	!
File→Exit→Window (save session)	end
File→Exit→Locked (all windows, save session)	end locked
File→Exit→Released (all windows, release emulator)	end release_system
Display→Context	pwd, pws
Display→Memory	display memory
Display→Memory→Mnemonic ()	display memory --EXPR-- mnemonic
Display→Memory→Mnemonic at PC	display memory mnemonic at_pc
Display→Memory→Mnemonic Previous	display memory mnemonic previous_display
Display→Memory→Hex ()→bytes	display memory --EXPR-- blocked bytes
Display→Memory→Hex ()→words	display memory --EXPR-- blocked words
Display→Memory→Hex ()→long	display memory --EXPR-- blocked long
Display→Memory→Real ()→short	display memory --EXPR-- real short
Display→Memory→Real ()→long	display memory --EXPR-- real long
Display→Memory→At ()	display memory --EXPR--
Display→Memory→Repetitively	display memory repetitively
Display→Data Values	display data
Display→Data Values→New ()→<type>	display data --EXPR-- <type>
Display→Data Values→Add ()→<type>	display data, --EXPR-- <type>

Pull-down	Command Line
Display→Configuration Info	display configuration_info
Display→Configuration Info→Diagnostics	display configuration_info diagnostics
Display→Configuration Info→Chip Selects (SIM)	display configuration_info sim_chip_selects
Display→Configuration Info→Chip Selects (Emulator SIM)	display configuration_info emsim_chip_selects
Display→Configuration Info→Bus Interface Ports (SIM)	display configuration_info bus_interface_ports
Display→Configuration Info→Bus Interface Ports (Emulator SIM)	display configuration_info embus_interface_ports
Display→Configuration Info→Memory Map	display configuration_info memory_map
Display→Configuration Info→Reset Mode Value	display configuration_info reset_mode
Display→Configuration Info→Initialization Source Code	display configuration_info init_source_code
Display→SIM Register Differences	sync_sim_registers difference
Display→Trace	display trace
Display→Registers	display registers
Display→Breakpoints	display software_breakpoints
Display→Status	display status
Display→Simulated IO	display simulated_io
Display→Global Symbols	display global_symbols
Display→Local Symbols ()	display local_symbols_in --SYMB--
Display→Pod Commands	display pod_command
Display→Error Log	display error_log
Display→Event Log	display event_log
Modify→Emulator Config	modify configuration
Modify→Memory	modify memory
Modify→Memory at ()	modify memory --EXPR--
Modify→Register	modify register
Modify→SIM Registers→Copy Processor SIM to Emulator SIM	sync_sim_registers from_6830x_to_config
Modify→SIM Registers→Copy Emulator SIM to Processor SIM	sync_sim_registers to_6830x_from_config
Modify→SIM Registers→Default Emulator SIM to Reset Values	sync_sym_registers default_emsim

Pull-down	Command Line
Execution→Run→from PC	run
Execution→Run→from ()	run from --EXPR--
Execution→Run→from Transfer Address	run from transfer_address
Execution→Run→from Reset	run from reset
Execution→Run→until ()	run until --EXPR--
Execution→Step Source→from PC	step source
Execution→Step Source→from ()	step source from --EXPR--
Execution→Step Source→from Transfer Address	step source from transfer_address
Execution→Step Instruction→from PC	step
Execution→Step Instruction→from ()	step from --EXPR--
Execution→Step Instruction→from Transfer Address	step from transfer_address
Execution→Break	break
Execution→Reset	reset
Breakpoints→Display	display software_breakpoints
Breakpoints→Enable	modify software_breakpoints enable/disable
Breakpoints→Permanent ()	modify software_breakpoints set --EXPR-- permanent
Breakpoints→Temporary ()	modify software_breakpoints set --EXPR-- temporary
Breakpoints→Set All	modify software_breakpoints set
Breakpoints→Clear ()	modify software_breakpoints clear --EXPR--
Breakpoints→Clear All	modify software_breakpoints clear

Pull-down	Command Line
Trace→Display	display trace
Trace→Display Options	display trace
Trace→Trace Spec	N/A (browses recall buffer for trace commands)
Trace→After ()	trace after STATE
Trace→Before ()	trace before STATE
Trace→About ()	trace about STATE
Trace→Only ()	trace only STATE
Trace→Only () Prestore	trace only STATE prestore anything
Trace→Again	trace again
Trace→Repetitively	<previous trace spec> repetitively
Trace→Everything	trace
Trace→Until ()	trace before STATE break_on_trigger
Trace→Until Stop	trace on_halt
Trace→Stop	stop_trace
Settings→Source/Symbol Modes→Absolute	set source off symbols off
Settings→Source/Symbol Modes→Symbols	set source off symbols on
Settings→Source/Symbol Modes→Source Mixed	set source on inverse_video on symbols on
Settings→Source/Symbol Modes→Source Only	set source only inverse_video off symbols on
Settings→Display Modes	set
Settings→Pod Command Keyboard	display pod_command; pod_command keyboard
Settings→Simulated IO Keyboard	display simulated_io; modify keyboard_to_simio
Settings→Command Line	N/A (toggles the command line)

How Pop-up Menus Map to the Command Line

The following tables show the items available in the pop-up menus and the command line commands to which they map.

Mnemonic Memory Display Pop-up	Command Line
Set/Clear Software Breakpoint	modify software_breakpoints set/clear --EXPR--
Edit Source	! vi +<line> <file> ! no_prompt_before_exit
Run Until	run until --EXPR--
Trace After	trace after STATE
Trace Before	trace before STATE
Trace About	trace about STATE
Trace Until	trace before STATE break_on_trigger

Breakpoints Display Pop-up	Command Line
Set/Inactivate Breakpoint	modify software_breakpoints set/deactivate --EXPR--
Clear (delete) Breakpoint	modify software_breakpoints clear --EXPR--
Enable/Disable Software Breakpoints	modify software_breakpoints enable/disable
Set All Breakpoints	modify software_breakpoints set
Clear (delete) All Breakpoints	modify software_breakpoints clear

Symbols Display Pop-up	Command Line
Display Local Symbols	display local_symbols_in --SYMB--
Display Parent Symbols	display local_symbols_in --SYMB--, display global_symbols
Cut Full Symbol Name	N/A
Edit File Defining Symbol	! vi +<line> <file> ! no_prompt_before_exit

Status Line Pop-up	Command Line
Remove Temporary Message	N/A
Command Line On/Off	(toggles command line)
Display Error Log	display error_log
Display Event Log	display event_log

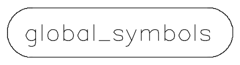
Command Line Pop-up	Command Line
Position Cursor, Replace Mode	<INSERT CHAR> key (when in insert mode)
Position Cursor, Insert Mode	<INSERT CHAR> key
Execute Command	<RETURN> key
Clear to End of Line	<CTRL>e
Clear Entire Line	<CTRL>u
Command Line Off	(toggles command line)

Syntax Conventions

Conventions used in the command syntax diagrams are defined below.

Oval-shaped Symbols

Oval-shaped symbols show options available on the softkeys and other commands that are available, but do not appear on softkeys (such as log_commands and wait). These appear in the syntax diagrams as:

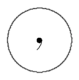


Rectangular-shaped Symbols

Rectangular-shaped symbols contain prompts or references to other syntax diagrams. Prompts are enclosed with angle brackets (< and >). References to other diagrams are shown in all capital letters. Also, references to expressions are shown in all capital letters, for example --EXPR-- and --SYMB-- (see those syntax diagrams). These appear in the following syntax diagrams as:

Circles

Circles indicate operators and delimiters used in expressions and on the command line as you enter commands. These appear in the syntax diagrams as:



The -NORMAL- Key

The softkey labeled -NORMAL- allows you exit the --SYMB-- definition, and access softkeys that are not displayed when defining expressions. You can press this key after you have defined an expression to view other available options.

Commands

Emulator/analyzer Softkey Interface commands are summarized in the table below and described in the following pages.

!UNIX_COMMAND	display event_log	modify memory[4]
bbaunload	display global_symbols	modify register[1]
break	display local_symbols_in	modify software_breakpoints[1]
cd (change directory)[3]	display memory[4]	modify tags
cmb_execute	display pod_command	name_of_module[3]
<command file>[3]	display registers[1]	performance_measurement_end
copy configuration_info	display simulated_io[2]	performance_measurement_init
copy data[4]	display software_breakpoints	performance_measurement_run
copy display	display status	pod_command
copy error_log	display trace	pwd (print working directory)[3]
copy event_log	end	pws (print working symbol)[3]
copy global_symbols	forward[3]	reset
copy help	help[3]	run
copy local_symbols_in	load <absolute_file>	set
copy memory[4]	load configuration	specify
copy pod_command	load emul_mem	step
copy registers[1]	load fg_mon	stop_trace
copy software_breakpoints	load trace	store memory
copy status	load trace_spec	store trace
copy trace	load user_memory	store trace_spec
cws(change working symbol)[3]	log_commands[3]	sync_sim_registers[1]
display configuration_info	modify configuration	trace
display data[4]	modify keyboard_to_simio[2]	wait[3]
display error_log		

[1] This option is not available in real-time mode.
[2] This is only available when simulated I/O is defined.
[3] These commands are not displayed on softkeys.
[4] This option is not available in real-time mode if addresses are in user memory.

bbaunld

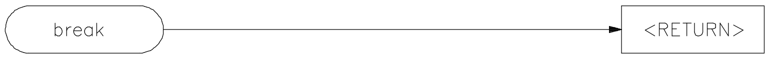
This command is available when the HP Branch Validator product is installed. This basis branch analyzer (BBA) product is used to analyze the testing of your programs, create more complete test suites, and quantify your level of testing.

The HP Branch Validator records branches executed in a program and generates reports that provide information about program execution during testing. It uses a special C preprocessor to add statements that write to a data array when program branches are taken. After running the program in the emulator (using test input), you can use the bbaunload command to store the BBA information to a file. Then, you can generate reports based on the stored information.

See Also

Refer to the *HP Branch Validator (BBA) User's Guide* for complete details on the ***bbaunload*** command syntax.

break



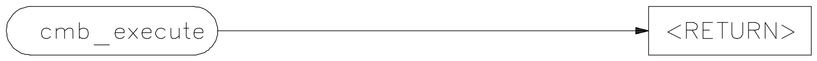
This command causes the emulator to leave user program execution and begin executing in the monitor.

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.
reset	Break releases the processor from reset, and diverts execution to the monitor.
running in monitor	The break command does not perform any operation while the emulator is executing in the monitor.

See Also The ***reset***, ***run***, and ***step*** commands.

cmb_execute

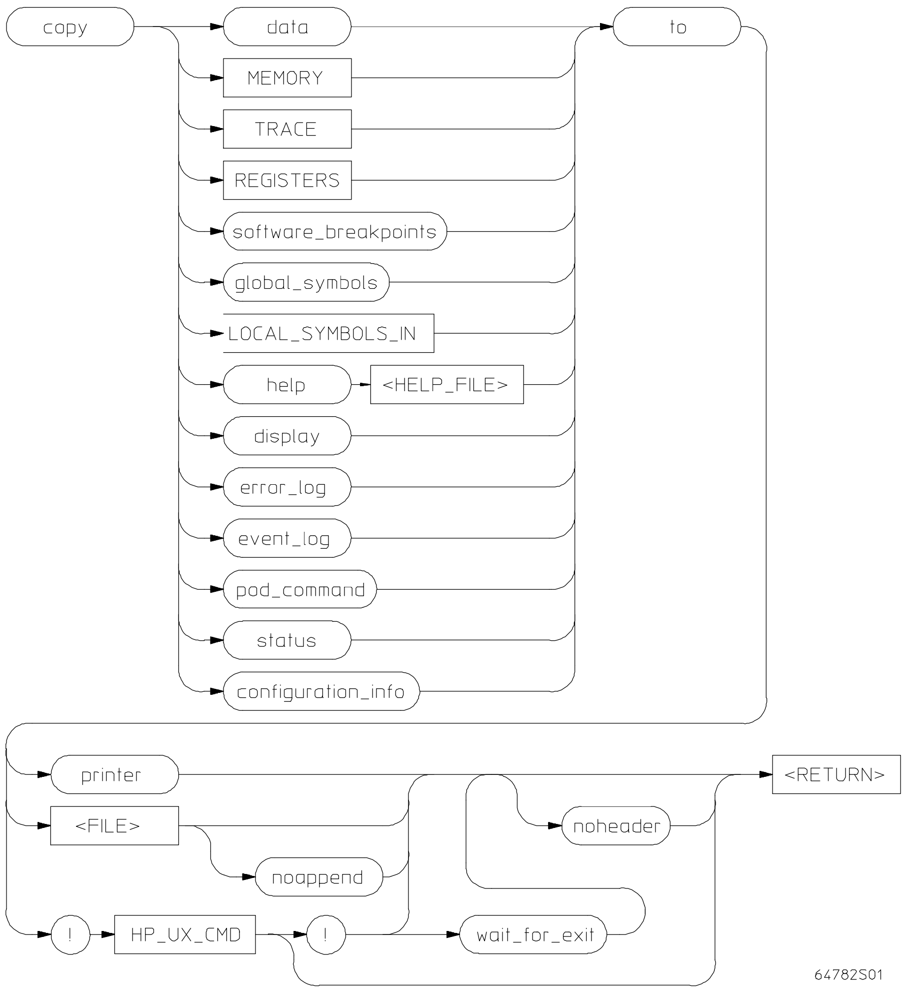


The cmb_execute command causes the emulator to emit an EXECUTE pulse on its rear panel Coordinated Measurement Bus (CMB) connector. All emulators connected to the CMB (including the one sending the CMB EXECUTE pulse) and configured to respond to this signal will take part in the measurement.

See Also

The ***specify run*** and ***specify trace*** commands.

copy



Use this command with various parameters to save or print emulation and analysis information.

The ***copy*** command copies selected information to your system printer or listing file, or directs it to an UNIX process.

Depending on the information you choose to copy, default values may be options selected for the previous execution of the display command. For example, if you display memory locations 10h through 20h, then issue a copy memory to myfile command, myfile will list only memory locations 10h through 20h.

The parameters are as follows:

configuration_info	Copies the last configuration information display.
data	Copies a list of memory contents formatted in various data types (see display data).
display	Copies the display to a selected destination.
error_log	Copies the most recent errors that have occurred.
event_log	Copies the most recent events that have occurred.
<FILE>	This prompts you for the name of a file where you want the specified information to be copied. If you want to specify a file name that begins with a number, you must precede the file name with a backslash. For example: ***copy display to*** \12.10.
global_symbols	Copies a list of global symbols to the selected destination.
help	Copies the contents of the emulation help files to the selected destination.
<HELP_FILE>	This represents the name of the help file to be copied. Available help file names are displayed on the softkey labels.
UNIX CMD	This represents an UNIX filter or pipe where you want to route the output of the copy command. UNIX commands must be preceded by an exclamation point (!). An exclamation point following the UNIX command continues command line execution after the UNIX command executes. Emulation is not affected when using an UNIX command that is a shell intrinsic.
local_symbols_in	Copies all the children of a given symbol to the selected destination. See the --SYMB-- syntax page and the *Symbolic Retrieval Utilities User's Guide* for information on symbol hierarchy.
memory	Copies a list of the contents of memory to the selected destination.
noappend	This causes any copied information to overwrite an existing file with the same name specified by <FILE>. If this option is not selected, the default operation is to append the copied information to the end of an existing file with the same name that you specify.
noheader	Copies the information into a file without headings.
pod_command	This allows you to copy the most recent commands sent to the HP 64700 Series emulator/analyzer.

printer	This option specifies your system printer as the destination device for the copy command. Before you can specify the printer as the destination device, you must define PRINTER as a shell variable. For example, you could enter the text shown below after the "$" symbol:

```
$ PRINTER=lp
$ export PRINTER
```

If you don’t want the print message to overwrite the command line, execute:

```
$ set PRINTER = "lp -s"
```

registers	Copies a list of the contents of the emulation processor registers to the selected destination.
software _breakpoints	Copies a list of the current software breakpoints to a selected destination.
status	Copies emulation and analysis status information.
to	This allows you to specify a destination for the copied information.
trace	Copies the current trace listing to the selected destination.
wait_for_exit	Waits for the UNIX command to complete before returning.
!	An exclamation point specifies the delimiter for UNIX commands. An exclamation point must precede all UNIX commands. A trailing exclamation point should be used if you want to return to the command line and specify noheader. Otherwise, the trailing exclamation point is optional. If an exclamation point is part of the UNIX command, a backslash (\) must precede the exclamation point.

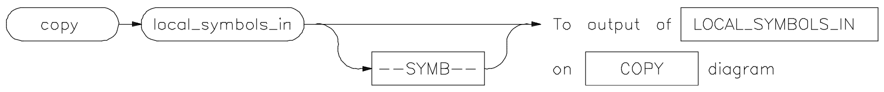
Examples

See the following pages on various ***copy*** syntax diagrams.

See Also

See the following pages on various ***copy*** syntax diagrams.

copy local_symbols_in



This command lets you copy local symbols contained in a source file and relative segments (program, data, or common) to the selected destination.

Local symbols are symbols that are children of the particular file or symbol defined by --SYMB--, that is, they are defined in that file or scope.

For additional information on symbols, refer to the --SYMB-- syntax pages and the *Symbolic Retrieval Utilities User's Guide*.

--SYMB-- is the current working symbol.

The parameters are as follows:

--SYMB-- This option represents the symbol whose children are to be listed. See the --SYMB-- syntax diagram and the *Symbolic Retrieval Utilities User's Guide* for information on symbol hierarchy.

Examples

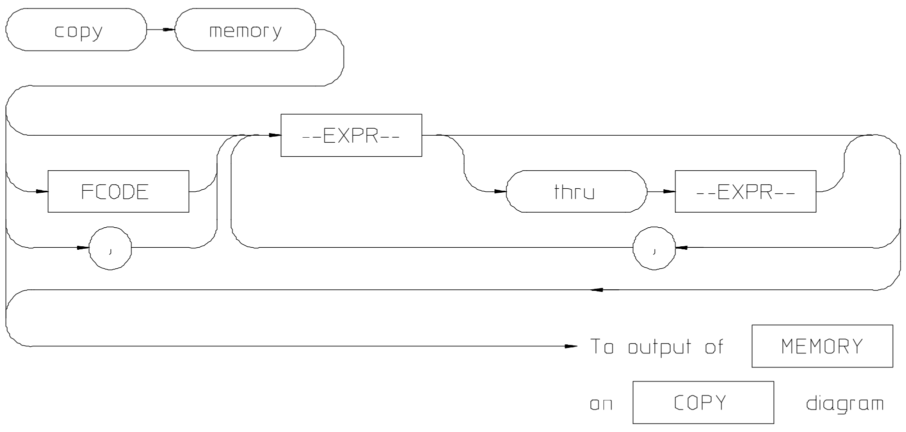
copy local_symbols_in `mod_name` ***to printer***

copy local_symbols_in `mod_name:` ***to*** `linenumfile`

See Also

The ***display local_symbols_in*** command.

copy memory



This command copies the contents of a memory location or series of locations to the specified output.

The memory contents are copied in the same format as specified in the last display memory command.

Contents of memory can be displayed if program runs are not restricted to real-time. Memory contents are listed as an asterisk (*) under the following conditions:

- The address refers to guarded memory.
- Runs are restricted to real-time, the emulator is running a user program, and the address is located in user memory.

Values in emulation memory can always be displayed.

Initial values are the same as those specified by the command ***display memory*** 0 ***blocked bytes offset_by*** 0.

Defaults are to values specified in the previous ***display memory*** command.

The parameters are as follows:

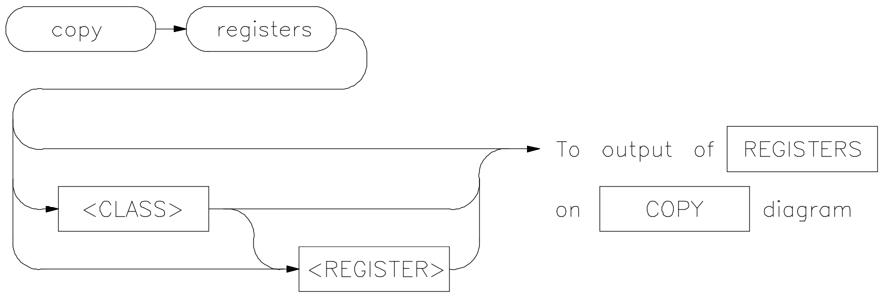
--EXPR--	An expression is a combination of numeric values, symbols, operators, and parentheses, specifying a memory address or offset value. See the EXPR syntax diagram.
FCODE	The function code used to define the address space being referenced. See the syntax diagram for FCODE to see a list of the function codes available and for an explanation of those codes.
,	A comma used immediately after memory in the command line appends the current copy memory command to the preceding display memory command. The data specified in both commands is copied to the destination specified in the current command. Data is formatted as specified in the current command. The comma is also used as a delimiter between values when specifying multiple memory addresses.

Examples

```
copy memory start to printer

copy memory 0 thru 100h , start thru +5 , 500H ,
target2 to memlist

copy memory 2000h thru 204fh to memlist
```

See Also The ***display memory***, ***modify memory***, and ***store memory*** commands.

copy registers



This command copies the contents of the processor registers to a file or printer.

The copy register process does not occur in real-time. The emulation system must be configured for nonreal-time operation to list the registers while the processor is running.

With no options specified, the basic register class is copied. This includes the local and global registers.

The parameters are as follows:

<CLASS>
<REGISTER>
Specifies a particular class of the emulator registers.

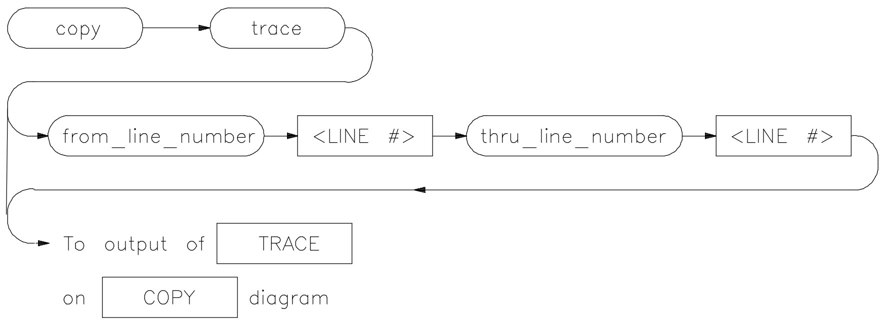
Examples

```
copy registers global to printer

copy registers to reglist
```

See Also

The ***display registers*** and ***modify registers*** commands.

copy trace



This command copies the contents of the trace buffer to a file or to the printer.

Trace information is copied in the same format as specified in the last display trace command.

Initial values are the same as specified by the last ***display trace*** command.

The parameters are as follows:

from_line_number	This specifies the trace list line number from which copying will begin.
<LINE#>	Use this with from_line_number and thru_line_number to specify the starting and ending trace list lines to be copied.
thru_line_number	Specifies the last line number of the trace list to include in the copied range.

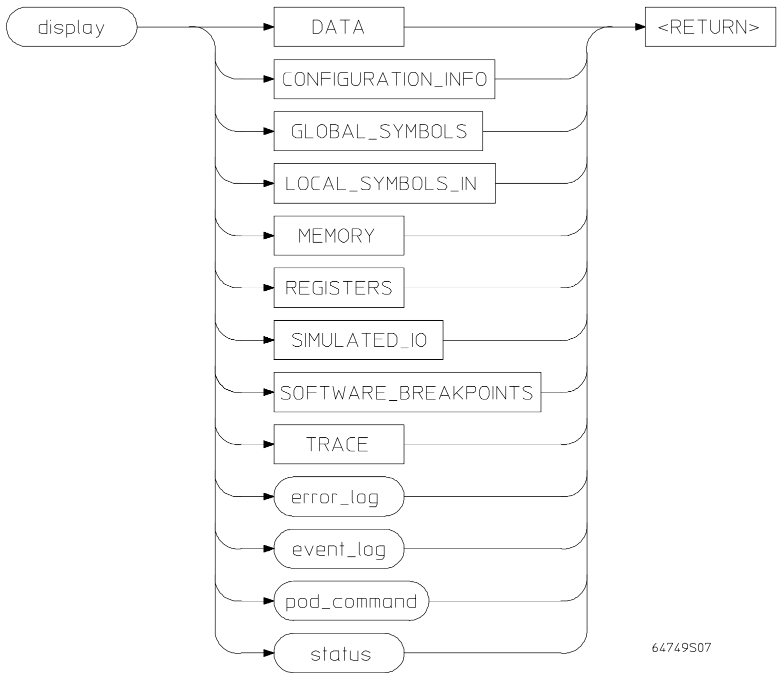
Examples

```
copy trace to tlist

copy trace from_line_number 0 thru_line_number 5
to longtrac
```

See Also

The ***display trace*** and ***store trace*** commands.

display



This command displays selected information on your screen.

You can use the <Up arrow>, <Down arrow>, <PREV>, and <NEXT> keys to view the displayed information. For software_breakpoints, data, memory, and trace displays you can use the <CTRL>g and <CTRL>f keys to scroll left and right if the information goes past the edge of the screen.

Depending on the information you select, defaults may be the options selected for the previous execution of the ***display*** command.

The parameters are as follows:

data	This allows you to display a list of memory contents formatted in various data types (see the display data pages for details).
error_log	This option displays the recorded list of error messages that occurred during the emulation session.
event_log	This option displays the recorded list of events.
global_symbols	This option lets you display a list of all global symbols in memory.

local_symbols_in	This option lets you display all the children of a given symbol. See the --SYMB-- syntax page and the *Symbolic Retrieval Utilities User's Guide* for details on symbol hierarchy.
memory	This option allows you to display the contents of memory.
pod_command	This option lets you display the output of previously executed emulator pod commands.
registers	This allows you to display the contents of emulation processor registers.
simulated_io	This lets you display data written to the simulated I/O display buffer after you have enabled polling for simulated I/O in the emulation configuration.
software _breakpoints	This option lets you display the current list of software breakpoints.
status	This displays the emulator and trace status.
trace	This displays the current trace list.

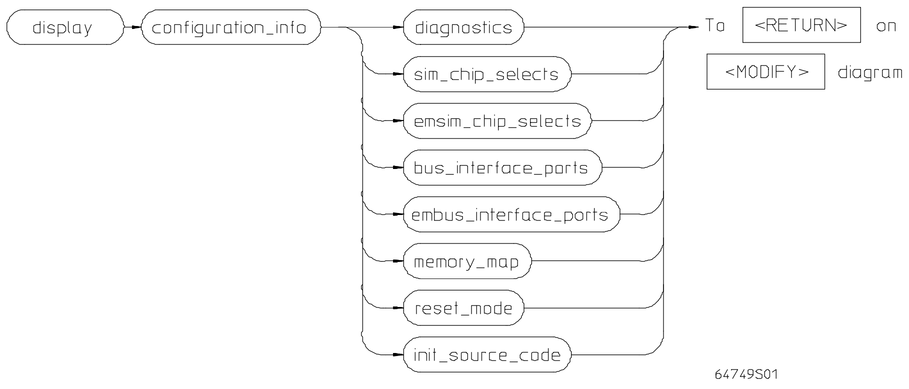
Examples

```
display event_log

display local_symbols_in mod_name
```

See Also

The ***copy*** command description and the following pages which describe the various ***display*** commands.

display configuration_info



This command displays information about emulator configuration and processor SIM programming. You can also display diagnostic information about inconsistencies found in the emulator configuration.

The parameters are as follows:

diagnostics	Checks all parts of the emulator configuration and reports any inconsistencies. It identifies errors that result from inconsistencies between related configuration values. These errors should be resolved in order for the emulator to operate correctly. This option primarily checks for inconsistencies between the mapper and the EMSIM registers, but it also provides status messages about expectations and limitations of the emulator of which you should be aware. (These checks are primarily between the reset mode configuration value and the EMSIM registers.) If no messages are returned, no inconsistencies are found in the emulator configuration.
sim_chip_selects emsim_chip _selects	Display chip select information from the sim (processor) register set or the emsim (emulator) register set. The resulting display shows: How the chip select is assigned. The base address. The block size. Other information from the option register.
bus_interface_ports embus_interface _ports	Display bus interface information from the sim (processor) register set or the emsim (emulator) register set. The resulting display shows the pin assignments for the available ports.
memory_map	When in the memory map section of the emulator configuration, the ranges of memory that have been mapped are displayed. The memory map configuration information shows detailed information about the memory map and how actual mapper resources are allocated due to the current programming of the chip selects in the EMSIM register sets.
reset_mode	Displays information about the reset mode configuration value, whether it is generated internally by the emulator or externally by the target system.
init_source_code	Displays the assembly language program that will initialize the processor as defined by the current EMSIM register contents.

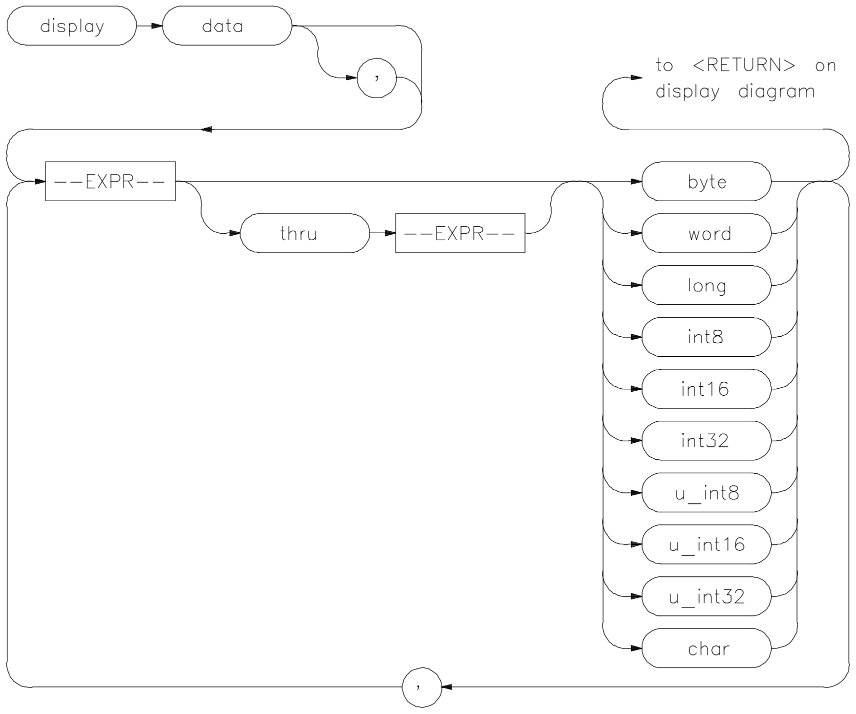
Examples

```
display configuration_info diagnostics

display configuration_info memory_map
```

See Also

The ***sync_sim_registers*** and ***modify configuration*** commands. Also, see the "Verifying the Emulator Configuration" section in the "Configuring the Emulator" chapter.

display data



This command can display the values of simple data types in your program. Using this command can save you time; otherwise, you would need to search through memory displays for the location and value of a particular variable.

The address, identifier, and data value of each symbol may be displayed. You must issue the command set symbols on to see the symbol names displayed.

In the first ***display data*** command after you begin an emulation session, you must supply at least one expression specifying the data item(s) to display.

Thereafter, the ***display data*** command defaults to the expressions specified in the last ***display data*** command, unless new expressions are supplied or appended (with a leading comma).

Symbols are normally set off until you give the command ***set symbols on***. Otherwise, only the address, data type, and value of the data item will be displayed.

The parameters are as follows:

,	A leading comma allows you to append additional expressions to the previous display data command. Commas between expression/data type specifications allow you to specify multiple variables and types for display with the current command.
--EXPR--	Prompts you for an expression specifying the data item to display. The expression can include various math operators and program symbols. See the --EXPR-- and --SYMB-- syntax pages for more information.
thru --EXPR--	Allows you to specify a range of addresses for which you want data display. Typically, you use this to display the contents of an array. You can display both single-dimensioned and multi-dimensioned arrays. Arrays are displayed in the order specified by the language definition, typically row major order for most Algol-like languages.
<TYPE>	Specifies the format in which to display the information. (Data type information is not available from the symbol database, so you must specify.)
byte	Hex display of one 8 bit location.
word	Hex display of one 16 bit location.
long	Hex display of one 32 bit location. Note that byte ordering in word and long displays is determined by the conventions of the processor in use.
int8	Display of one 8 bit location as a signed integer using two's complement notation.
int16	Display of two bytes as a signed integer using two's complement notation.
int32	Display of four bytes as a signed integer using two's complement notation.
u_int8	Display of one byte as an unsigned positive integer.
u_int16	Display of two bytes as an unsigned positive integer.
u_int32	Display of four bytes as an unsigned positive integer.
char	Displays one byte as an ASCII character in the range 0 through 127. Control characters and values in the range 128 through 255 are displayed as a period (.).

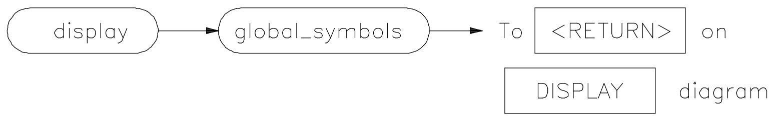
Examples

```
display data Msg_A thru +17 char, Stack long

set symbols on

set width label 30

display data , Msg_B thru +17 char, Msg_Dest thru
+17 char
```

See Also

The ***copy data*** and ***set*** commands.

display global_symbols

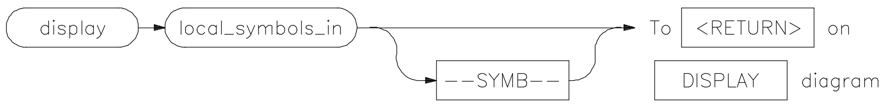


This command displays the global symbols defined for the current absolute file.

Global symbols are symbols declared as global in the source file. They include procedure names, variables, constants, and file names. When the ***display global_symbols command*** is used, the listing will include the symbol name and its logical address.

See Also The ***copy global_symbols*** command.

display local_symbols_in



This command displays the local symbols in a specified source file and their relative segment (program, data, or common).

Local symbols of --SYMB-- are the ones which are children of the file and/or scope specified by --SYMB--. That is, they are defined in that file or scope.

See the --SYMB-- syntax pages and the *Symbolic Retrieval Utilities User's Guide* for further explanation of symbols.

Displaying the local symbols sets the current working symbol to the one specified.

The parameters are as follows:

--SYMB-- This option represents the symbol whose children are to be listed. See the --SYMB-- syntax diagram and the *Symbolic Retrieval Utilities User's Guide* for more information on symbol hierarchy and representation.

Examples

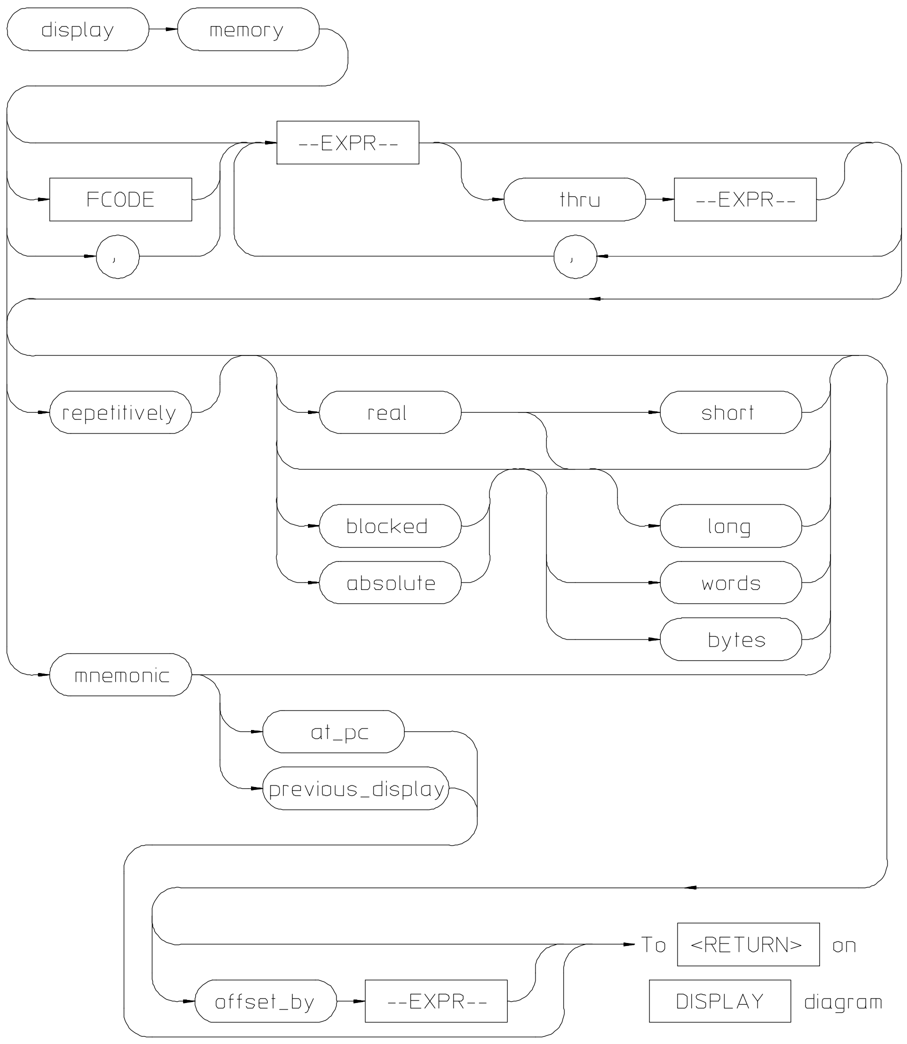
display local_symbols_in `mod_name`

display local_symbols_in `mod_name:main`

See Also

The ***copy local_symbols_in*** command.

display memory



This command displays the contents of the specified memory location or series of locations.

The memory contents can be displayed in mnemonic, hexadecimal, or real number format. In addition, the memory addresses can be listed offset by a value, which allows the information to be easily compared to the program listing.

When displaying memory mnemonic and stepping, the next instruction that will step is highlighted. The memory mnemonic display autopages to the new address if the next PC goes outside the currently displayed address range. This feature works even if stepping is performed in a different emulation window than the one displaying memory mnemonic.

Pending software breakpoints are shown in the memory mnemonic display by an asterisk (*) in the leftmost column of the assembly instruction or source line that has a pending breakpoint.

A label column (symbols) may be displayed for all memory displays except blocked mode. Memory mnemonic may be displayed with source and assembly code intermixed, or with source code only. Symbols also can be displayed in the memory mnemonic string. (See the ***set*** command.)

Initial values are the same as specified by the command:

```
display memory 0 blocked bytes offset_by 0
```

Defaults are values specified in a previous ***display memory*** command.

The symbols and source defaults are:

```
set source off symbols off
```

The parameters are as follows:

absolute	Formats the memory listing in a single column.
at_pc	Displays the memory at the address pointed to by the current program counter value.
blocked	Formats the memory listing in multiple columns.
bytes	Displays the absolute or blocked memory listing as byte values.
--EXPR--	An expression is a combination of numeric values, symbols, operators, and parentheses, specifying a memory address or memory offset value. See the EXPR syntax diagram.
FCODE	The function code used to define the address space being referenced. See the syntax diagram for FCODE to see a list of the function codes available and for an explanation of those codes.
long	Displays memory in a 64-bit real number format or 32-bit long words when preceded by blocked or absolute.

mnemonic	This causes the memory listing to be formatted in assembly language instruction mnemonics with associated operands. When specifying mnemonic format, you should include a starting address that corresponds to the first byte of an operand to ensure that the listed mnemonics are correct. If set source only is on, you will see only the high level language statements and corresponding line numbers.
offset_by	This option lets you specify an offset that is subtracted from each of the absolute addresses before the addresses and corresponding memory contents are listed. You might select the offset value so that each module appears to start at address 0000H. The memory contents listing will then appear similar to the assembler or compiler listing. This option is also useful for displaying symbols and source lines in dynamically relocated programs.
previous_display	Returns to display associated with the previous mnemonic memory display command.
real	Formats memory values in the listing as real numbers. (NaN in the display list means "Not a Number.")
repetitively	Updates the memory listing display continuously. You should only use this to monitor memory while running user code, since it is very CPU intensive. To allow updates to the current memory display whenever memory is modified, a file is loaded, software breakpoint is set, etc., use the set update command.
short	Formats the memory list as 32-bit real numbers.
thru	This option lets you specify a range of memory locations to be displayed. Use the <Up arrow>, <Down arrow>, <NEXT>, and <PREV> keys to view additional memory locations.
words	Displays the absolute or blocked memory listing as 16-bit word values.
,	A comma after memory in the command line appends the current display memory command to the preceding display memory command. The data specified in both commands is displayed. The data will be formatted as specified in the current command. The comma is also a delimiter between values when specifying multiple addresses.

Examples

You can display memory in real number and mnemonic formats:

```
display memory 2000h thru 202fh , 2100h real long
```

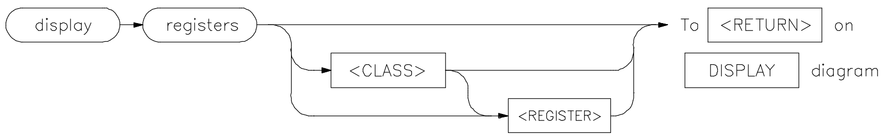
```
display memory 400h mnemonic
```

```
set symbols on
set source on
display memory main mnemonic
```

See Also

The ***copy memory***, ***modify memory***, ***set***, and ***store memory*** commands.

display registers



This command displays the current contents of the emulation processor registers.

If a ***step*** command just executed, the mnemonic representation of the last instruction is also displayed, if the current display is the register display. This process does not occur in real-time. The emulation system must be configured for nonreal-time operation to display registers while the processor is running. Symbols also may be displayed in the register step mnemonic string (see ***set symbols***).

With no options specified, the basic register class is displayed as the default. This includes the local and global registers.

The parameters are as follows:

<CLASS>	This allows you to display a particular class of emulation processor registers.
<REGISTER>	This displays an individual register or control register field.

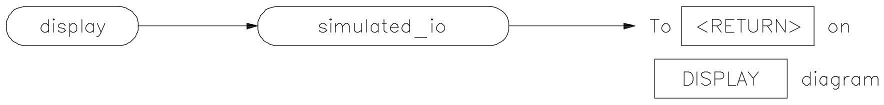
Examples

```
display registers

display registers BASIC D2
```

See Also

The ***copy registers***, ***modify registers***, ***set***, and ***step*** commands.

display simulated_io



This command displays information written to the simulated I/O display buffer.

After you have enabled polling for simulated I/O during the emulation configuration process, six simulated I/O addresses can be defined. You then define files used for standard input, standard output, and standard error.

For details about setting up simulated I/O, refer to the *Simulated I/O User's Guide*.

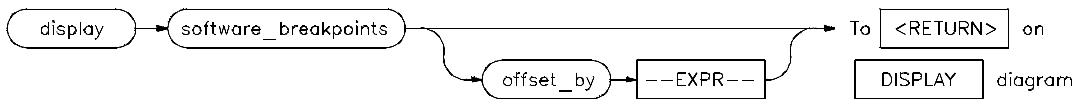
Examples

```
display simulated_io
```

See Also

The ***modify configuration*** and ***modify keyboard_to_simio*** commands.

display software_breakpoints



This command displays the currently defined software breakpoints and their status.

If the emulation session is continued from a previous session, the listing will include any previously defined breakpoints. The column marked "status" shows whether the breakpoint is pending, inactivated, or unknown.

A pending breakpoint causes the processor to enter the emulation monitor upon execution of that breakpoint. Executed breakpoints are listed as inactivated. Entries that show an inactive status can be reactivated by executing the ***modify software_breakpoints set*** command.

A label column also may be displayed for addresses that correspond to a symbol. See the ***set*** command for details.

The parameters are as follows:

--EXPR-- An expression is a combination of numeric values, symbols, operators, and parentheses, specifying an offset value for the breakpoint address. See the --EXPR-- syntax diagram.

offset_by This option allows you to offset the listed software breakpoint address value from the actual address of the breakpoint. By subtracting the offset value from the breakpoint address, the system can cause the listed address to match that given in the assembler or compiler listing.

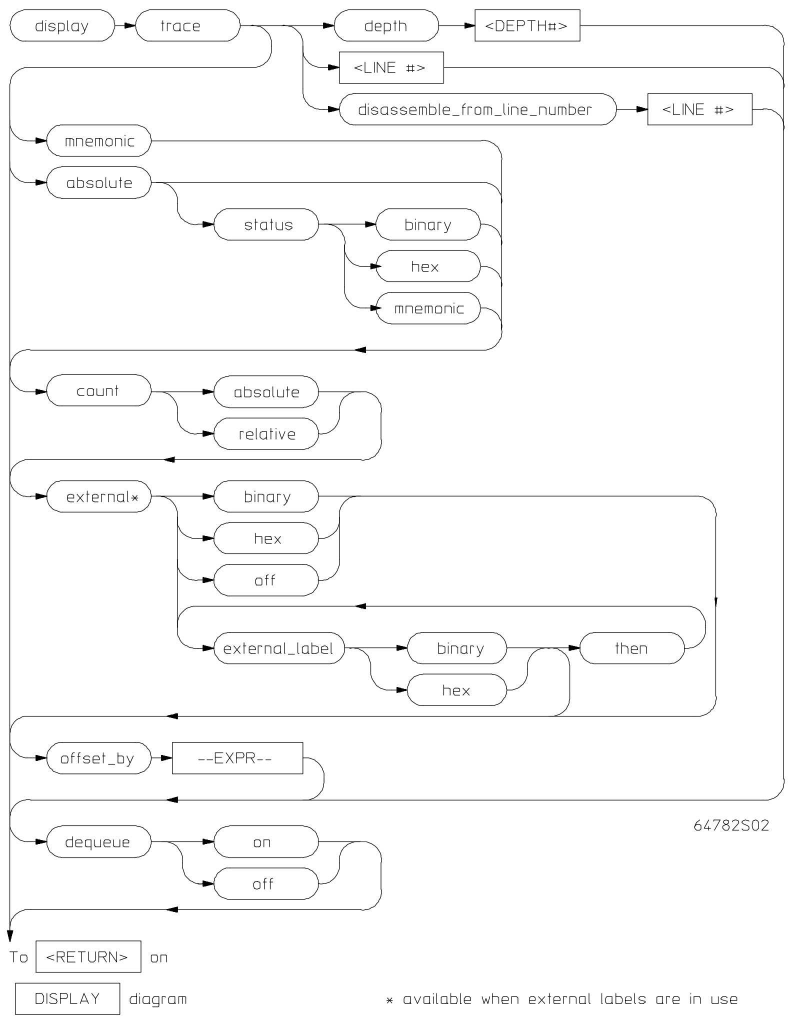
Examples

```
display software_breakpoints

display software_breakpoints offset_by 1000H
```

See Also

The ***copy software_breakpoints***, ***modify software_breakpoints***, and ***set*** commands.

display trace



This command displays the contents of the trace buffer.

Captured information can be presented as absolute hexadecimal values or in mnemonic form. The processor status values captured by the analyzer can be listed mnemonically or in hexadecimal or binary form.

Addresses captured by the analyzer are physical addresses.

The offset_by option subtracts the specified offset from the addresses of the executed instructions before listing the trace. With an appropriate entry for offset, each instruction in the listed trace will appear as it does in the assembled or compiled program listing.

The count parameter lists the time associated with a trace event either relative to the previous event in the trace list or as an absolute count measured from the trigger event.

The source parameter allows display of source program lines in the trace listing, enabling you to quickly correlate the trace list with your source program.

Initial values are the same as specified by the command:

```
display trace mnemonic count relative offset_by 0
```

The parameters are as follows:

absolute	Lists trace information in hexadecimal format, rather than mnemonic opcodes.
count	
absolute	This lists the time count for each event of the trace as the total time measured from the trigger event.
relative	This lists the time count for each event of the trace as the time measured relative to the previous event.
depth	
<DEPTH#>	This defines the number of states to be uploaded by the interface. Note that after you have changed the trace depth, execute the command wait measurement_complete before displaying the trace. Otherwise the new trace states will not be available.
dequeue	Displays the trace list with or without dequeuing. A dequeued trace list is available through the disassembly options. In a dequeued trace list, unused instruction prefetch cycles are discarded, and operand cycles are placed immediately following the corresponding instruction fetch. If you choose a non-dequeued trace list, instruction and operand fetches are shown exactly as captured by the analyzer.
disassemble _from_line _number	Displays the trace at a certain line number and disassembles instruction opcodes.

--EXPR--	An expression is a combination of numeric values, symbols, operators, and parentheses, specifying an offset value to be subtracted from the addresses traced by the emulation analyzer. See the EXPR syntax diagram.
external	
binary	Displays the external analyzer trace list in binary format.
<external _label>	This option displays a defined external analyzer label.
hex	Displays the external analyzer trace list in hexadecimal format.
off	Use this option to turn off the external trace list display.
then	This allows you to display multiple external analysis labels. This option appears when more than one external analyzer label is in use.
<LINE#>	This prompts you for the trace list line number to be centered in the display. Also, you can use <LINE#> with disassemble_from_line_number. <LINE#> prompts you for the line number from which the inverse assembler attempts to disassemble data in the trace list.
mnemonic	Lists trace information with opcodes in mnemonic format.
offset_by	This option allows you to offset the listed address value from the address of the instruction. By subtracting the offset value from the physical address of the instruction, the system makes the listed address match that given in the assembler or compiler listing.

This option is also useful for displaying symbols and source lines in dynamically relocated programs.

Note that when using the set source only command, the analyzer may operate more slowly than when using the set source on command. This is an operating characteristic of the analyzer:

> When you use the command set source on, and are executing only assembly language code (not high-level language code), no source lines are displayed. The trace list will then fill immediately with the captured assembly language instructions.
>
> When using set source only, no inverse assembled code is displayed. Therefore, the emulation software will try to fill the display with high-level source code. This requires the emulation software to search for any captured analysis data generated by a high-level language statement.

In conclusion, you should not set the trace list to set source only when tracing assembly code. This will result in optimum analyzer performance.

status

binary	Lists absolute status information in binary form.
hex	Lists absolute status information in hexadecimal form.
mnemonic	Lists absolute status information in mnemonic form.

Examples

`display trace count absolute`

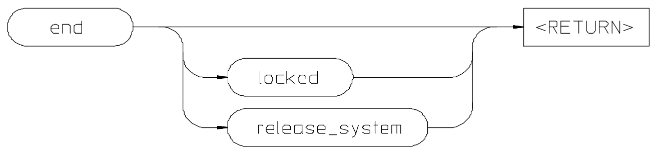
`display trace absolute status binary`

`display trace mnemonic`

See Also

The ***copy trace***, ***store trace***, and ***set*** commands.

end



This command terminates the current emulation session.

You can end the emulation session and keep the emulator in a locked state. The current emulation configuration is stored, so that you can continue the emulation session on reentry to the emulator. You also can release the emulation system when ending the session so that others may use the emulator.

Note that pressing <CTRL>d performs the same operation as pressing ***end*** <RETURN>. Pressing <CTRL>\ or <CTRL>| performs the same as ***end release_system*** <RETURN>.

When the emulation session ends, control returns to the UNIX shell without releasing the emulator.

The parameters are as follows:

locked	This option allows you to stop all active instances of an emulator/analyzer interface session in one or more windows and/or terminals. This option is not available when operating the emulator in the measurement system.
release_system	This option stops all instances of the emulator/analyzer interface in one or more windows or terminals. The emulation system is released for other users. If you do not release the emulation system when ending, others cannot access it.

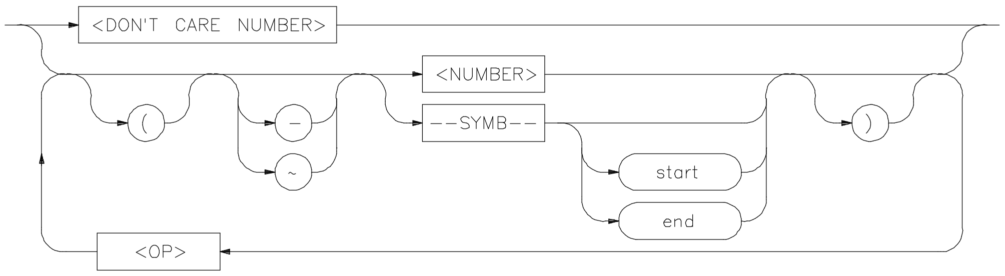
Examples

```
end

end release_system
```

See Also

The "Exiting the Emulator/Analyzer Interface" section in the "Starting and Exiting HP 64700 Interfaces" chapter.

--EXPR--



An expression is a combination of numeric values, symbols, operators, and parentheses used to specify address, data, status, executed address, or any other value used in the emulation commands.

The function of an expression (--EXPR--) is to let you define the address, data, status, or executed address expression that fits your needs. You can combine multiple values to define the expression.

Certain emulation commands will allow the option of <+EXPR> after pressing a thru softkey. This allows you to enter a range without retyping the original base address or symbol. For example, you could specify the address range

disp_buf ***thru*** disp_buf + 25

as

disp_buf ***thru*** +25

The parameters are as follows:

<DON'T CARE NUMBER>
You can include "don't care numbers" in expressions. These are indicated by a number containing an "x." These numbers may be defined as binary, octal, decimal, or hexadecimal. For example: 1fxxh, 17x7o, and 011xxx10b are valid.

Note that "Don't care numbers" are not valid for all commands.

--NORMAL--
This appears as a softkey label to enable you to return to the --EXPR-- key. The --NORMAL-- label can be accessed whenever defining an expression, but

	is only valid when "C" appears on the status line, which indicates a valid expression has been defined.
<NUMBER>	This can be an integer in any base (binary, octal, decimal, or hexadecimal), or can be a string of characters enclosed with quotation marks.
<OP>	This represents an algebraic or logical operand and may be any of the following (in order of precedence):

mod	modulo
*	multiplication
/	division
&	logical AND
+	addition
-	subtraction
|	logical OR

--SYMB--	This allows you to define symbolic information for an address, range of addresses, or a file. See the --SYMB-- syntax pages and the *Symbolic Retrieval Utilities User's Guide* for more information on symbols.
end	This displays the last location where the symbol information may be located. For example, if a particular symbol is associated with a range of addresses, end will represent the last address in that range.
start	This displays first memory location where the symbol you specify may be located. For example, if a particular symbol is associated with a range of addresses, start will represent the first address in that range.
<UNARY>	This defines either the algebraic negation (minus) sign (-) or the logical negation (NOT) sign (~).
( )	Parentheses may be used in expressions to enclose numbers. For every opening parenthesis, a closing parenthesis must exist.

Note that when "C" appears on the right side of the status line, a valid expression exists. The --NORMAL-- key can be accessed at any time, but is only valid when "C" is on the command line.

Note that when a thru softkey has been entered, a <+ EXPR> prompt appears. This saves you from tedious repeated entry of long symbols and expressions. For example:

```
disp_buf thru +25
```

is the same as

```
disp_buf thru disp_buf + 25
```

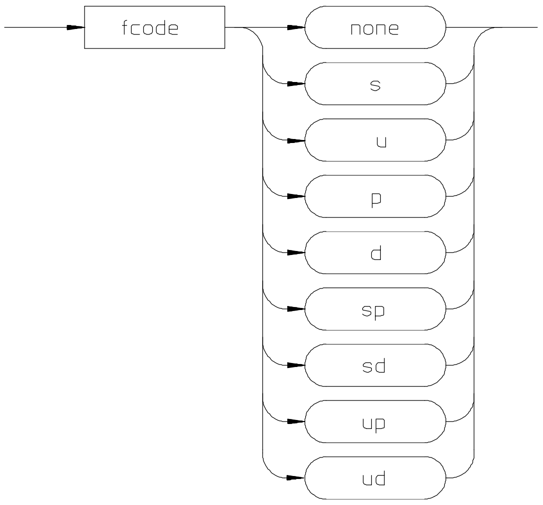
Examples

```
05fxh

0ffffh

disp_buf + 5

symb_tbl + (offset / 2)

start

mod_name: line 15 end
```

See Also

The SYMB syntax description.

FCODE



The function code is used to define the address space being referenced. Select the appropriate function code from those listed below.

d	Data space.
none	Causes the emulator to ignore the function code bits.
p	Program space.
s	Supervisor space.
sd	Supervisor data space.
sp	Supervisor program space.
u	User space.
ud	User data space.
up	User program space.

Examples

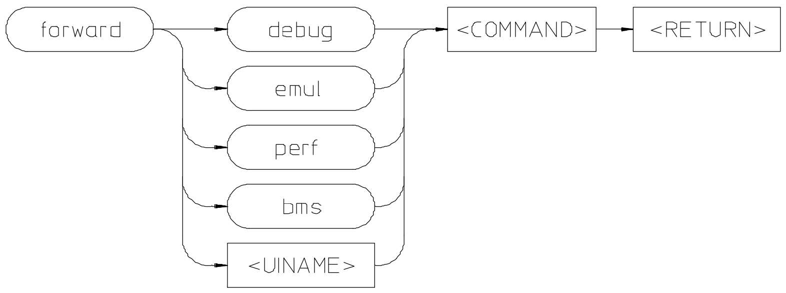
To copy a portion of user data memory to a file:

copy memory fcode ud 1000H ***thru*** 1fffH ***to*** mymem

To modify a location in program memory:

modify memory fcode p 5000h ***long to*** 12345678h

forward



This command lets you forward commands to other HP 64700 interfaces that use the "emul700dmn" daemon process to coordinate actions between the interfaces.

bms	Sends messages to the Broadcast Message Server or BMS.
<COMMAND>	An ASCII string, enclosed in quotes, that is the command to be forwarded to the named interface.
emul	Forwards command to the emulator/analyzer interface.
perf	Forwards commands to the software performance analyzer interface.
<UINAME>	Forwards commands to a user interface name other than those available on the softkeys.

Examples

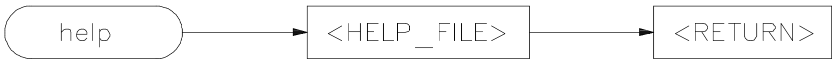
To send the "profile" command to the software performance analyzer:

```
forward perf "profile"
```

See Also

The *User's Guide* for the interface to which you are forwarding commands.

help



Displays information about system and emulation features during an emulation session.

Typing help or ? displays softkey labels that list the options on which you may receive help. When you select an option, the system will list the information to the screen.

The help command is not displayed on the softkeys. You must enter it into the keyboard. You may use a question mark in place of help to access the help information.

The parameters are as follows:

<HELP_FILE> This represents one of the available options on the softkey labels. You can either press a softkey representing the help file, or type in the help file name. If you are typing in the help file name, make sure you use the complete syntax. Not all of the softkey labels reflect the complete file name.

Examples

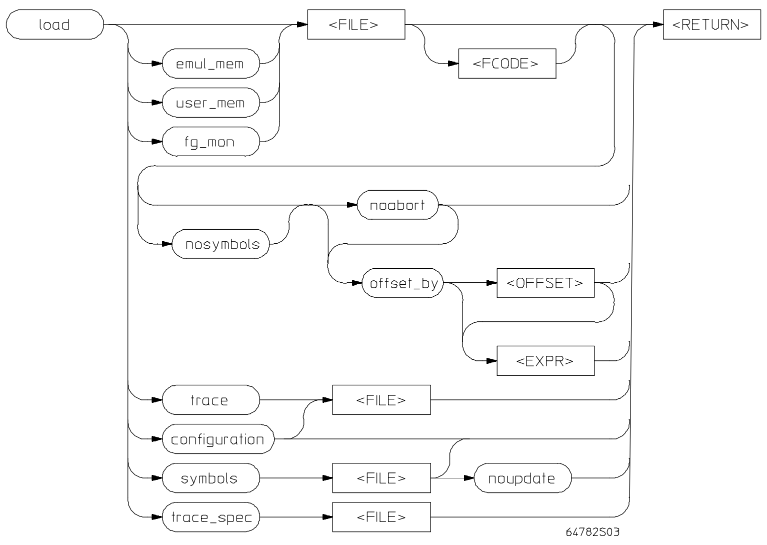
```
help system_commands

? run
```

This is a summary of the commands that appear on the softkey labels when you type help or press ?:

system_commands
run
trace
step
break
display
modify
load
store
copy
reset

stop_trace
end
software_breakpoints
registers
expressions (--EXPR--)
symbols (--SYMB--)
specify
cmb
cmb_execute
map
set
wait
pod_command
bbaunload
coverage
performance_measurement_initialize
performance_measurement_run
performance_measurement_end

load



This command transfers absolute files from the host computer into emulation or target system RAM. With other parameters, the ***load*** command can load emulator configuration files, trace records, trace specifications, or symbol files.

The absolute file contains information about where the file is stored. The memory map specifies that the locations of the file are in user (target system) memory or emulation memory. This command also allows you to access and display previously stored trace data, load a previously created configuration file, and load absolute files with symbols.

Note that any file specified by <FILE> cannot be named "configuration", "emul_mem", "user_mem", "symbols", "trace", or "trace_spec" because these are reserved words, and are not recognized by the emulator/analyzer interface as ordinary file names.

The absolute file is loaded into emulation memory by default.

The parameters are as follows:

configuration

This option specifies that a previously created emulation configuration file will be loaded into the emulator. You can follow this option with a file name. Otherwise the previously loaded configuration will be reloaded.

emul_mem	Loads only those portions of the absolute file that reside in memory ranges mapped as emulation memory.
FCODE	Specifies the address space where the file will be loaded.
fg_mon	Loads a foreground monitor.
<FILE>	This represents the absolute file to be loaded into either target system memory, emulation memory (.X files are assumed), or the trace memory (.TR files are assumed).
offset_by <OFFSET>	Specifies an offset value that is subtracted from the address before the foreground monitor file is loaded.
noabort	This option allows you to load a file even if part of the file is located at memory mapped as "guarded" or "target ROM" (trom).
nosymbols	This option causes the file specified to be loaded without symbols.
noupdate	This option suppresses rebuilding of the symbol data base when you load an absolute file. If you load an absolute file, end emulation, then modify the file (and relink it), the symbol database will not be updated upon reentering emulation and reloading the file. The default is to rebuild the database.
symbols	This option causes the file specified to be loaded with symbols.
trace	This option allows you to load a previously generated trace file.
trace_spec	This option allows you to load a previously generated trace specification. Note that the current trace specification will be modified, but a new trace will not be started. To start a trace with the newly loaded trace specification, enter trace again or specify trace again (not trace). If you specify trace, a new trace will begin with the default trace specification, not the one you loaded.
user_mem	Loads only those portions of the absolute file that reside in memory ranges mapped as target memory.

Examples

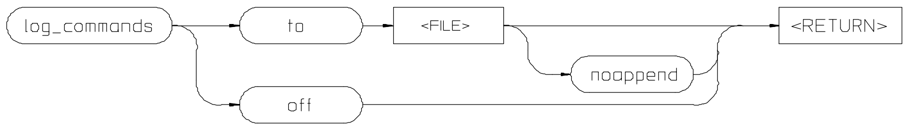
load `sort1`

load configuration `config3`

See Also

The ***display trace*** command.

log_commands



This command allows you to record commands that are executed during an emulation session.

Commands executed during an emulation session are stored in a file until this feature is turned off. This is a handy method for creating command files.

To execute the saved commands after the file is closed, type the filename on the command line.

The parameters are as follows:

<FILE>	This represents the file where you want to store commands that are executed during an emulation session.
noappend	If the named file is an existing file, this option causes the new commands to overwrite any information present in the file. If this option is not specified, new commands are appended to the existing contents of the file.
off	This option turns off the capability to log commands.
to	This allows you to specify a file for the logging of commands.

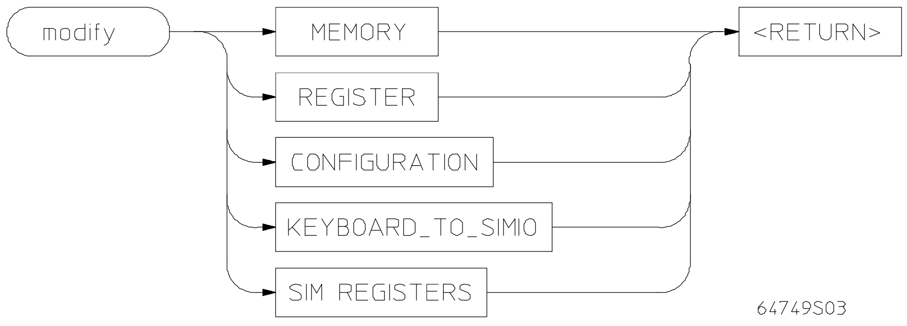
Examples

```
log_commands to logfile

log_commands off
```

See Also

The ***wait*** command.

modify



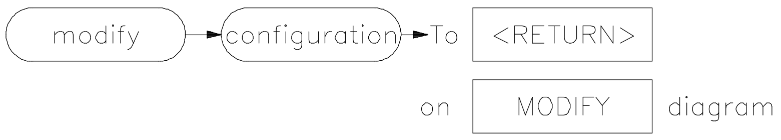
This command allows you to observe or change information specific to the emulator.

The ***modify*** command is used to:

- Modify contents of memory (as integers, strings, or real numbers).
- Modify the contents of the processor registers.
- View or edit the current emulation configuration.
- Modify the simulated I/O keyboard settings.
- Modify the SIM registers.

The following pages contain detailed information about the various modify syntax diagrams.

modify configuration



This command allows you to view and edit the current emulation configuration items.

The configuration questions are presented in sequence with either the default response, or the previously entered response. You can select the currently displayed response by pressing <RETURN>. Otherwise, you can modify the response as you desire, then press <RETURN>.

The default responses defined on powerup are displayed.

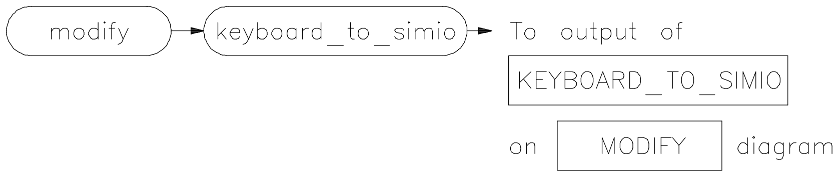
Examples

modify configuration

See Also

The ***load configuration*** command.

modify keyboard_to_simio



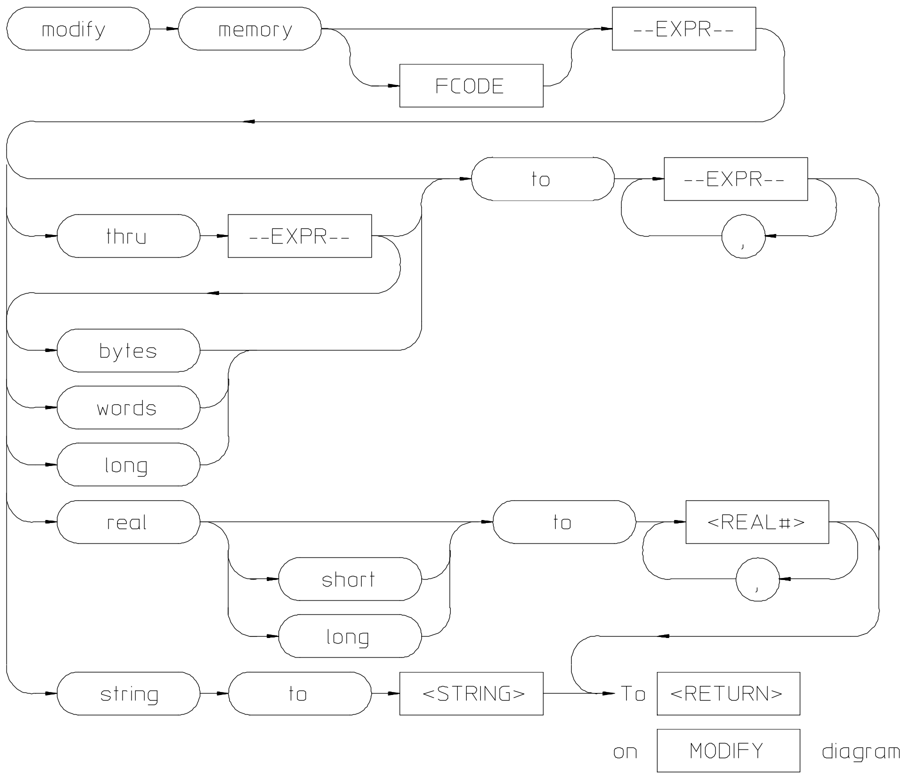
This command allows the keyboard to interact with your program through the simulated I/O software.

When the keyboard is activated for simulated I/O, its normal interaction with emulation is disabled. The emulation softkeys are blank and the ***suspend*** softkey is displayed on your screen. Pressing ***suspend*** <RETURN> will deactivate keyboard simulated I/O and return the keyboard to normal emulation mode. For details about setting up simulated I/O, refer to the *Simulated I/O User's Guide*.

See Also

The ***display simulated_io*** command.

modify memory



This command lets you modify the contents of selected memory locations.

You can modify the contents of individual memory locations to individual values. Or, you can modify a range of memory to a single value or a sequence of values.

Modify a series of memory locations by specifying the address of the first location in the series to be modified, and the values to which the contents of that location and successive locations are to be changed. The first value listed will replace the contents of the first memory location. The second value replaces the contents of the next memory location in the series, and so on, until the list is exhausted. When more than one value is listed, the value representations must be separated by commas. (See the examples for more information.)

A range of memory can be modified such that the content of each location in the range is changed to the single specified value, or to a single or repeated sequence. This type of memory modification is done by entering the limits of the memory range to be modified (--EXPR-- thru --EXPR--) and the value or list of values (--EXPR--, ... , --EXPR--) to which the contents of all locations in the range are to be changed.

Note that if the specified address range is not large enough to contain the new data, only the specified addresses are modified.

If the address range contains an odd number of bytes and a word operation is being executed, the last word of the address range will be modified. Thus the memory modification will stop one byte after the end of the specified address range.

If an error occurs in writing to memory (to guarded memory or target memory with no monitor) the modification is aborted at the address where the error occurred.

For integer memory modifications, the default is to the current display memory mode, if one is in effect. Otherwise the default is to "byte."

For real memory modifications, the default is to the current display memory mode, if one is in effect. Otherwise the default is "word."

The parameters are as follows:

bytes	Modify memory in byte values.
--EXPR--	An expression is a combination of numeric values, symbols, operators, and parentheses, specifying a memory address. See the EXPR syntax diagram.
FCODE	The function code used to define the address space being referenced. See the syntax diagram for FCODE to see a list of the function codes available and for an explanation of those codes.
long	Modify memory values as 32-bit long word values or 64-bit real values when preceded by real.
real	Modify memory as real number values.
<REAL#>	This prompts you to enter a real number as the value.
short	Modify memory values as 32-bit real numbers.
words	Modify memory values as 16-bit values.
string	Modify memory values to the ASCII character string given by <STRING>.
<STRING>	Quoted ASCII string including special characters as follows:

null	\0
newline	\n
horizontal tab	\t

backspace	\b
carriage return	\r
form feed	\f
backslash	\\
single quote	\’
bit pattern	\ooo (where ooo is an octal number)

thru	This option lets you specify a range of memory locations to be modified.
to	This lets you specify values to which the selected memory locations will be changed.
words	Modify memory locations as 32-bit values.
,	A comma is used as a delimiter between values when modifying multiple memory addresses.

Examples

modify memory `data1` ***bytes to*** `0E3H , 01H , 08H`

modify memory `data1` ***thru*** `DATA100` ***to*** `0FFFFH`

modify memory `0675H` ***real to*** `-1.303`

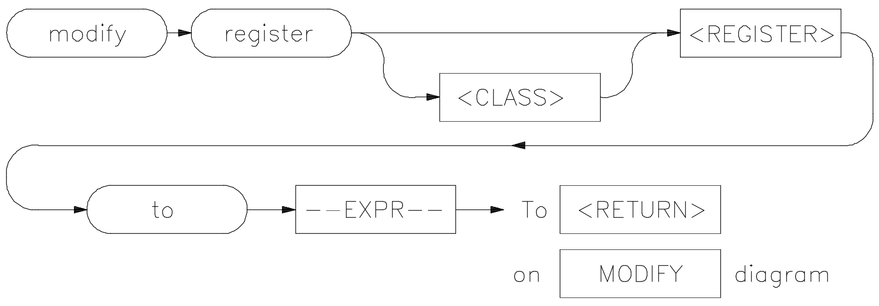
modify memory `temp` ***real long to*** `0.5532E-8`

modify memory `buffer` ***string to*** `"Test \n\0"`

See Also

The ***copy memory***, ***display memory***, and ***store memory*** commands.

modify register



This command allows you to modify the contents of the emulation processor internal registers.

The entry you specify for <REGISTER> determines which register is modified. Individual fields of control registers may be modified.

Register modification cannot be performed during real-time operation of the emulation processor. A ***break*** command or condition must occur before you can modify the registers.

The parameters are as follows:

--EXPR-- An expression is a combination of numeric values, symbols, operators, and parentheses, specifying a register value. For the floating-point registers, the value is interpreted as a decimal real number. See the --EXPR-- description.

<REGISTER> This represents the name of a register.

to Allows you to specify the values to which the selected registers will be changed.

Examples

modify register D2 ***to*** 41H

See Also

The ***copy registers***, ***display registers***, and ***modify registers*** commands.

modify SIM registers

The 6830x SIM is configured through the registers in the SIM register class; these registers control how the 6830x uses external signal lines.

The emulator's hardware is configured through the registers in the EMSIM register class.

Normally, the SIM and EMSIM registers should be programmed with the same values so they will be working together.

The default programming of the emulator hardware (EMSIM) matches the reset values of the 6830x SIM (refer to the appropriate Motorola *MC6830x User's Manual* for specific values).

If desired, the programming of the emulator hardware (EMSIM) can be transferred into the 6830x SIM with the ***sync_sim_registers to_6830x_from_config*** command. This happens automatically each time a break to the monitor from emulation reset occurs. This ensures that the 6830x is prepared to properly access memory when a program is downloaded to the emulator.

Alternatively, the emulator hardware (EMSIM) can be programmed from the 6830x SIM with the ***sync_sim_registers from_6830x_to_config*** command. This is useful if initialization code that configures the 6830x SIM exists, but you don't know what its values are. In this case, you can use the default configuration, run from reset to execute the initialization code, and use the ***sync_sim_registers from_6830x_to_config*** command to configure the emulator to match the 6830x SIM.

At any time, you can verify if the SIM and EMSIM are programmed the same with the ***sync_sim_registers difference*** command. Any differences between the two register sets will be listed.

It should be noted that the emulator hardware is programmed solely from the EMSIM register set and is therefore static with respect to the application program. No attempt is made to update the programming of the emulator hardware by tracking instructions that will program the 6830x SIM.

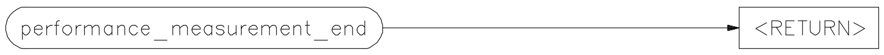
Examples

```
sync_sim_registers from_6830x_to_config

sync_sim_registers to_6830x_from_config
sync_sim_registers default_emsim
```

See Also

The ***sync_sim_registers*** command and the "Concepts" chapter.

performance_measurement_end



This command stores data previously generated by the ***performance_measurement_run*** command, in a file named "perf.out" in the current working directory.

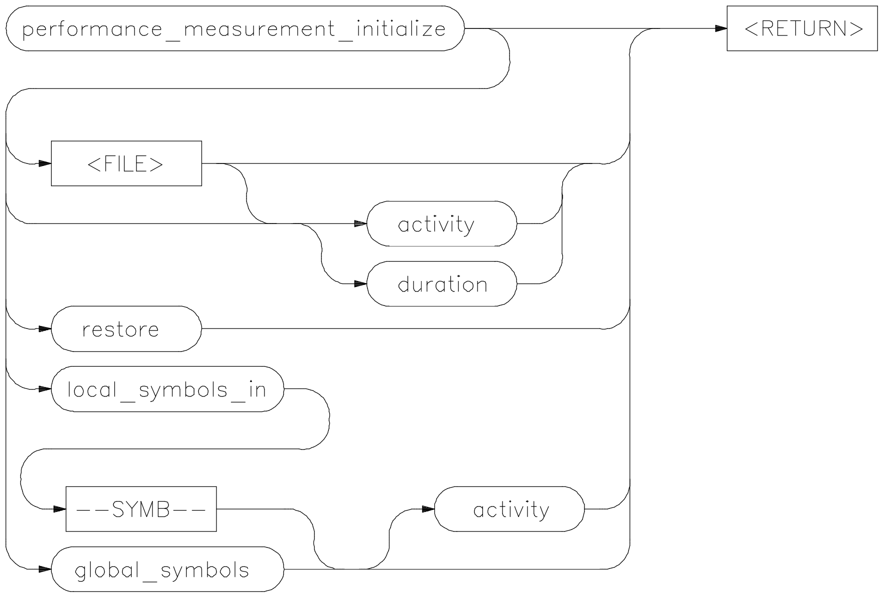
The file named "perf.out" is overwritten each time this command is executed. Current measurement data existing in the emulation system is not altered by this command.

Examples ***`performance_measurement_end`***

See Also The ***performance_measurement_initialize*** and ***performance_measurement_run*** commands.

Refer to the "Making Software Performance Measurements" chapter for examples of performance measurement specification and use.

performance_measurement_initialize



This command sets up performance measurements.

The emulation system will verify whether a symbolic database has been loaded. If a symbolic database has been loaded, the performance measurement is set up with the addresses of all global procedures and static symbols. If a valid database has not been loaded, the system will default to a predetermined set of addresses, which covers the entire emulation processor address range.

The measurement will default to "activity" mode.

Default values will vary, depending on the type of operation selected, and whether symbols have been loaded.

The parameters are as follows:

activity	This option causes the performance measurement process to operate as though an option is not specified.
duration	This option sets the measurement mode to "duration." Time ranges will default to a predetermined set (unless a user-defined file of time ranges is specified).
<FILE>	This represents a file you specify to supply user-defined address or time ranges to the emulator.
global_symbols	This option specifies that the performance measurement will be set up with the addresses of all global symbols and procedures in the source program.
local_symbols_in	This causes addresses of the local symbols to be used as the default ranges for the measurement.
restore	This option restores old measurement data so that a measurement can be continued when using the same trace command as previously used.
--SYMB--	This represents the source file that contains the local symbols to be listed. This also can be a program symbol name, in which case all symbols that are local to a function or procedure are used. See the SYMB syntax diagram.

Examples

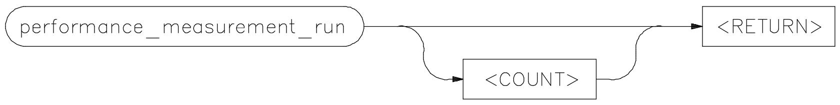
```
performance_measurement_initialize

performance_measurement_initialize duration

performance_measurement_initialize
local_symbols_in mod_name
```

See Also

The ***performance_measurement_run*** and ***performance_measurement_end*** commands.

Refer to the "Making Software Performance Measurements" chapter for examples of performance measurement specification and use.

performance_measurement_run



This command begins a performance measurement.

This command causes the emulation system to reduce trace data contained in the emulation analyzer, which will then be used for analysis by the performance measurement software.

The default is to process data presently contained in the analyzer.

The parameters are as follows:

<COUNT> This represents the number of consecutive traces you specify. The emulation system will execute the trace command, process the resulting data, and combine it with existing data. This sequence will be repeated the number of times specified by the COUNT option.

Note that the trace command must be set up correctly for the requested measurement. For an activity measurement, you can use the default trace command (trace).

For a duration measurement, you must set up the trace specification to store only the points of interest. To do this, for example, you could enter:

trace only <symbol_entry> ***or*** <symbol_exit>

Examples

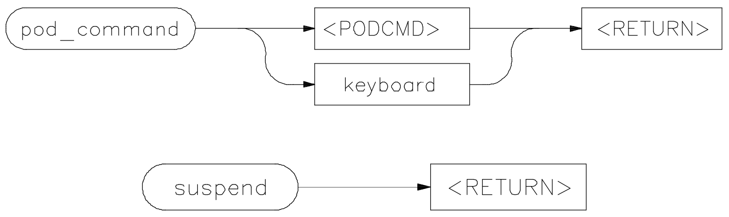
```
performance_measurement_run 10

performance_measurement_run
```

See Also

The ***performance_measurement_end*** and ***performance_measurement_initialize*** commands.

Refer to the "Making Software Performance Measurements" chapter for examples of performance measurement specification and use.

pod_command



Allows you to control the emulator through the direct HP 64700 Terminal Interface.

The HP 64700 Card Cage contains a low-level Terminal Interface, which allows you to control the emulator's functions directly. You can access this interface using ***pod_command***. The options to ***pod_command*** allow you to supply only one command at a time. Or, you can select a keyboard mode which gives you interactive access to the Terminal Interface.

There are certain commands that you should avoid while using the Terminal Interface through ***pod_command***.

stty, po, xp	Do not use. These commands will change the operation of the communications channel, and are likely to hang the Softkey Interface and the channel.
echo, mac	Using these may confuse the communications protocols in use on the channel.
wait	Do not use. The pod will enter a wait state, blocking access by the emulator/analyzer interface.
init, pv	These will reset the emulator pod and force an end release_system command.
t	Do not use. The trace status polling and unload will become confused.

To see the results of a particular ***pod_command*** (the information returned by the emulator pod), you use ***display pod_command***.

Refer to the *6830x Installation/Service/Terminal Interface User's Guide* for information on using the Terminal Interface to control the emulator.

The parameters are as follows:

keyboard	Enters an interactive mode where you can simply type Terminal Interface commands (unquoted) on the command line. Use display pod_command to see the results returned from the emulator.
<POD_CMD>	Prompts you for a Terminal Interface command as a quoted string. Enter the command in quotes and press <RETURN>.
suspend	This command is displayed once you have entered keyboard mode. Select it to stop interactive access to the Terminal Interface and return to the Graphical User Interface or Softkey Interface.

Examples

This example shows a simple interactive session with the Terminal Interface.

```
display pod_command

pod_command keyboard

cf

tsq

tcq
```

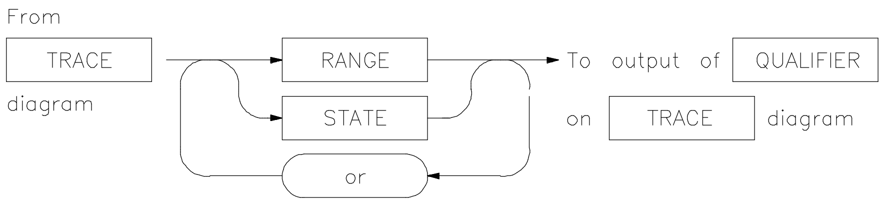
Click ***suspend*** to return to the Graphical User Interface or Softkey Interface.

See Also

The ***display pod_command*** command.

Also see the *6830x Installation/Service/Terminal Interface User's Guide* and the Terminal Interface online help information.

QUALIFIER



The QUALIFIER parameter is used with trace only, trace prestore, and TRIGGER to specify states captured during the trace measurement.

You may specify a range of states (RANGE) or specific states (STATE) to be captured. You can continue to "or" states until the analyzer resources are depleted. You can use only one RANGE statement in the entire trace command.

You can include "don't care numbers." These contain an "x" preceded and/or followed by a number. Some examples include 1fxxh, 17x7o, and 011xxx10b. "Don't care numbers" may be entered in binary, octal, or hexadecimal base.

The default is to qualify on all states.

The parameters are as follows:

or	This option allows you to specify multiple states (STATE) to be captured during a trace measurement. See the STATE syntax diagram.
RANGE	This allows you to specify a range of states to be captured during a trace measurement. See the RANGE syntax diagram.
STATE	This represents a unique state that can be a combination of address, data, status, and executed address values. See the STATE syntax diagram.

Examples

trace only address `mod_name:read_input`

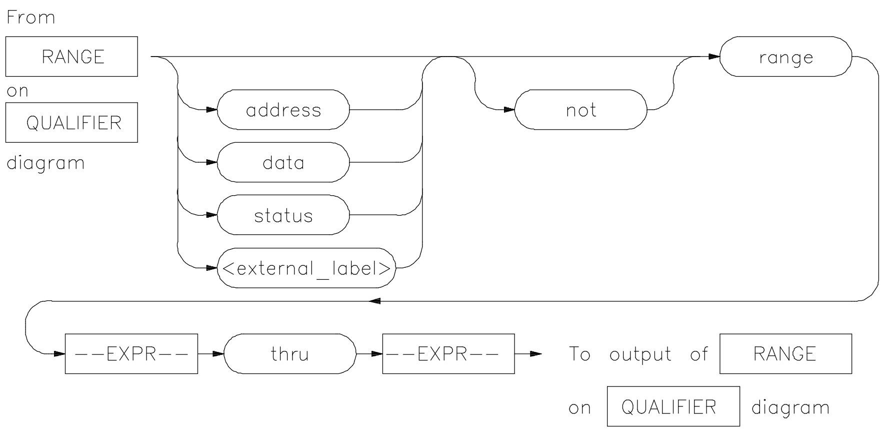
trace only address range `mod_name:read_input` ***thru*** `output`

trace only address range `mod_name:clear` ***thru*** `read_input`

See Also

The ***trace*** command.

RANGE



The RANGE parameter allows you to specify a condition for the trace measurement, made up of one or more values.

The range option can be used for state qualifier labels. Range can only be used once in a trace measurement.

Refer to the "Qualifying Trigger and Store Conditions" section in the "Using the Emulation Analyzer" chapter for a list of the predefined values that can be assigned to the status state qualifiers.

Expression types are "address" when none is chosen.

The parameters are as follows:

address	The value following this softkey is searched for on the lines that monitor the emulation processor's address bus.
data	The value following this softkey is searched for on the lines that monitor the emulation processor's data bus.
--EXPR--	An expression is a combination of numeric values, symbols, operators, and parentheses, specifying an address, data, status, or executed address value. See the EXPR syntax diagram for details.
<external_label>	This represents a defined external analyzer label.
not	This specifies that the analyzer search for the logical "not" of the specified range (this includes any addresses not in the specified range).

range	This indicates a range of addresses to be specified (--EXPR-- thru --EXPR--).
status	The value following this softkey is searched for on the lines that monitor other emulation processor signals.
thru	This indicates that the following address expression is the upper address in a range.

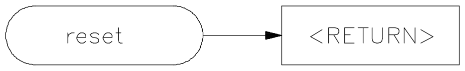
Examples

See the ***trace*** command examples.

See Also

The ***trace*** command and the QUALIFIER syntax description.

reset



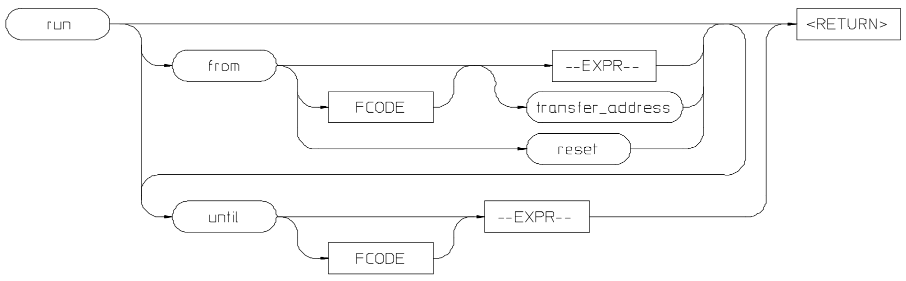
This command suspends target system operation and re-establishes initial emulator operating parameters, such as reloading control registers.

The reset signal is latched when the ***reset*** command is executed and released by either the run or break command.

See Also

The ***break*** and ***run*** commands.

run



This command causes the emulator to execute a program.

If the processor is in a reset state, ***run*** will cause the reset to be released.

If the emulator is configured to run directly into user code out of reset, the monitor will not be entered and part of your debug environment may be temporarily disabled. A subsequent break into the monitor will restore it. See the "Enter monitor from reset?" question in the configuration menu for more information.

If the from parameter and an address are specified, the processor will start running your program at that address. Otherwise, the run will occur from the address currently stored in the processor's program counter.

A ***run from reset*** command will reset the processor and then allow it to run. It is equivalent to entering a ***reset*** command followed by a ***run*** command.

If the emulator is configured to participate in the READY signal on the CMB, then this emulator will release the READY signal so that it will go TRUE if all other HP 64700 emulators participating on that signal are also ready. See the ***cmb_execute*** command description.

Qualifying a ***run*** command with an until parameter causes a software breakpoint to be set before the program is run.

If you omit the address option (--EXPR--), the emulator begins program execution at the current address specified by the emulation processor program counter. If an absolute file containing a transfer address has just been loaded, execution starts at that address.

The parameters are as follows:

address	Specifies an address for a temporary register breakpoint that will be programmed into one of the processor's two breakpoint registers. Up to two addresses may be specified.
--EXPR--	An expression is a combination of numeric values, symbols, operators, and parentheses, specifying a memory address. See the EXPR syntax diagram.
FCODE	The function code used to define the address space being referenced. See the syntax diagram for FCODE to see a list of the function codes available and for an explanation of those codes.
from	This specifies the address from which program execution is to begin.
reset	This option resets the processor prior to running.
transfer_address	This represents the starting address of the program loaded into emulation or target memory. The transfer address is defined in the linker map and is part of the symbol database associated with the absolute file.
until	Causes a software breakpoint to be set at the specified address before the program is run.

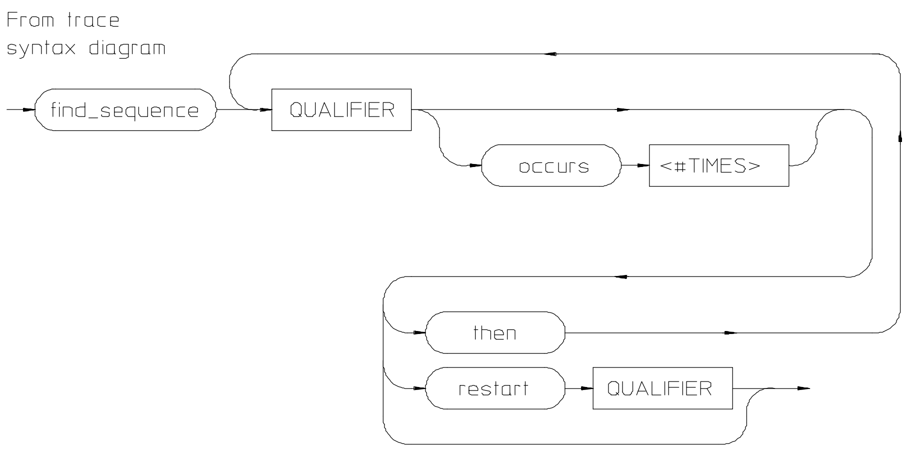
Examples

```
run

run from 810H

run from COLD_START

run from transfer_address until 910H
```

See Also

The ***step*** command.

SEQUENCING



Lets you specify complex branching activity that must be satisfied to trigger the analyzer.

Sequencing provides you with parameters for the ***trace*** command that let you define branching conditions for the analyzer trigger.

You are limited to a total of seven sequence terms, including the trigger, if no windowing specification is given. If windowing is selected, you are limited to a total of four sequence terms.

The analyzer default is no sequencing terms. If you select the sequencer using the find_sequence parameter, you must specify at least one qualifying sequence term.

The parameters are as follows:

find_sequence	Specifies that you want to use the analysis sequencer. You must enter at least one qualifier.
QUALIFIER	Specifies the address, data, status, or executed address value or value range that will satisfy this sequence term if looking for a sequence (find_sequence), or will restart at the beginning of the sequence (restart). See the QUALIFIER syntax pages for further information.

occurs	Selects the number of times a particular qualifier must be found before the analyzer proceeds to the next sequence term or the trigger term. This option is not available when trace windowing is in use. See the WINDOW syntax pages.
<#TIMES>	Prompts you for the number of times a qualifier must be found.
then	Allows you to add multiple sequence terms, each with its own qualifier and occurrence count.
restart	Selects global restart. If the analyzer finds the restart qualifier while searching for a sequence term, the sequencer is reset and searching begins for the first sequence term.

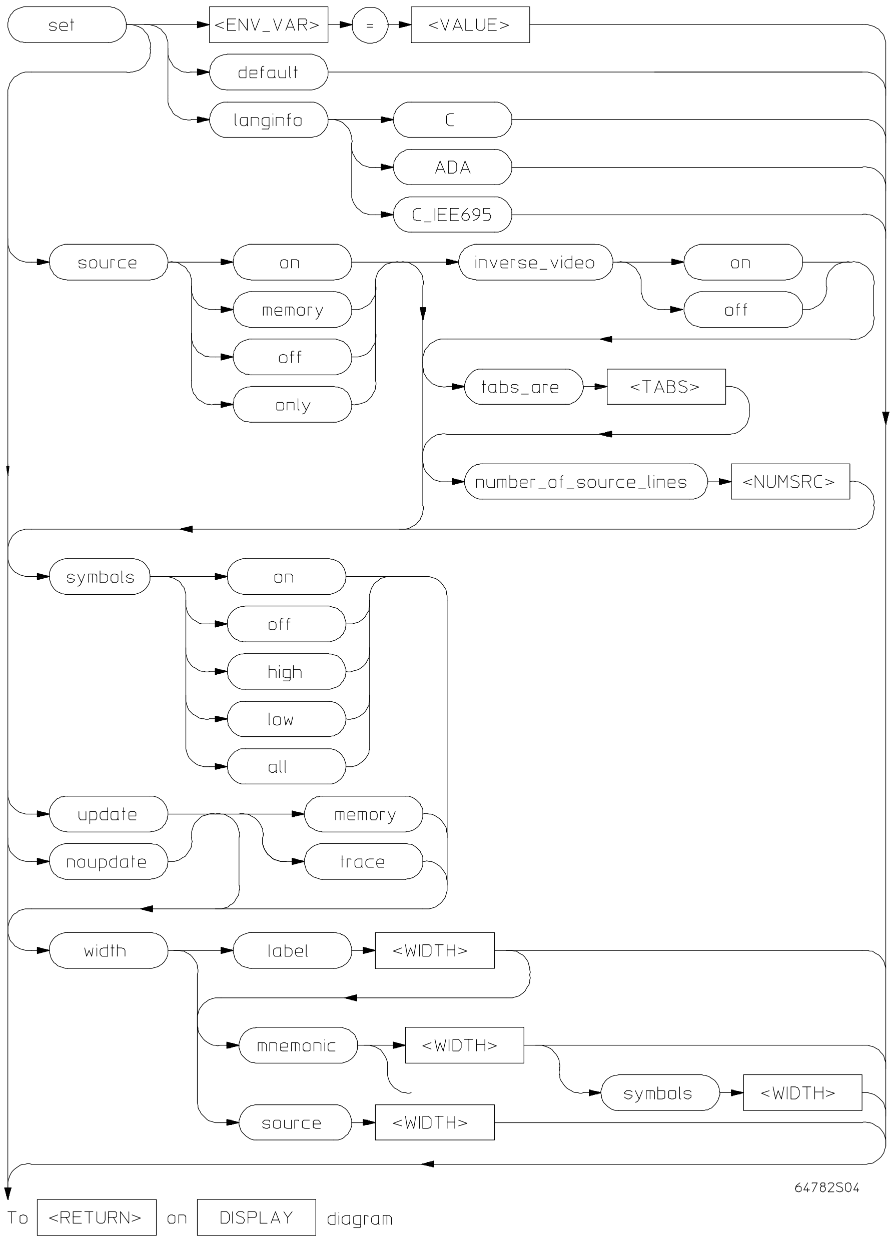
Examples

```
trace find_sequence Caller_3 then Write_Num
restart anly."anly.c": line 57 trigger after
Results+0c4h
```

See Also

The ***trace*** command and the QUALIFIER and WINDOW syntax descriptions.

set


set
<ENV_VAR>
=
<VALUE>
default
langinfo
C
ADA
C_IEE695
source
on
memory
off
only
inverse_video
on
off
tabs_are
<TABS>
number_of_source_lines
<NUMSRC>
symbols
on
off
high
low
all
update
noupdate
memory
trace
width
label
<WIDTH>
mnemonic
<WIDTH>
symbols
<WIDTH>
source
<WIDTH>
64782S04
To <RETURN> on DISPLAY diagram

Controls the display format for the data, memory, register, software breakpoint, and trace displays.With the ***set*** command, you can adjust the display format results for various measurements, making them easier to read and interpret. Formatting of source lines, symbol display selection and width, and update after measurement can be defined to your needs.

The display command uses the set command specifications to format measurement results for the display window. Another option to the ***set*** command, <ENV_VAR> = <VALUE>, allows you to set and export system variables to the UNIX environment.

The default display format parameters are the same as those set by the commands:

`set update`

`set source off symbols off`

You can return the display format to this state by entering:

`set default`

The parameters are as follows:

default	This option restores all the set options to their default settings.
<ENV_VAR>	Specifies the name of a UNIX environment variable to be set.
=	The equals sign is used to equate the <ENV_VAR> parameter to a particular value represented by <VALUE>.
inverse video	
`off`	This displays source lines in normal video.
`on`	This highlights the source lines on the screen (dark characters on light background) to differentiate the source lines from other data on the screen.

langinfo

In certain languages, you may have symbols with the same names but different types. For example, in IEEE695, you may have a file named main.c and a procedure named main. SRU would identify these as main(module) and main(procedure). The command display local_symbols_in main would cause an error message to appear (Ambiguous symbol: main(procedure, module)). Users of C tend to think the procedure is important and users of ADA tend to think the module is important. By entering "langinfo" and "C", SRU will interpret the above command to be main(procedure). With langinfo ADA, SRU will interpret the above command to be main(module).

`C` Identifies ANSI C as the language so SRU can use the C hierarchy to disambiguate symbols.

`ADA` Identifies ADA as the language so SRU can use the ADA hierarchy to disambiguate symbols.

`C_IEE695` Identifies C_IEEE-695 as the language so SRU can use the C_IEEE-695 hierarchy to disambiguate symbols.

An alternate method for making the langinfo specification is to use the environment variable, HP64SYMORDER. By making the following entry in your **.profile**, the langinfo setting will always be C, for example.

```
$ HP64SYMORDER=C   # I want to use the C disambiguating
                   # hierarchy
$ export HP64SYMORDER   # let children processes know
                        # about it
```

memory

Sets update option for memory displays only.

noupdate

When using multiple windows or terminals, and specifying this option, the display buffer in that window or terminal will not update when a new measurement completes. Displays showing memory contents are not updated when a command executes that could have caused the values in memory to change (modify memory, load, etc.).

number_of_
source_lines

This allows you to specify the number of source lines displayed for the actual processor instructions with which they correlate. Only source lines up to the previous actual source line will be displayed. Using this option, you can specify how many comment lines are displayed preceding the actual source line. The default value is 5.

`<NUMSRC>`	This prompts you for the number of source lines to be displayed. Values in the range 1 through 50 may be entered.
source	
`memory`	This displays source lines of memory.
`off`	This option prevents inclusion of source lines in the trace and memory mnemonic display lists.
`on`	This option displays source program lines preceding actual processor instructions with which they correlate. This enables you to correlate processor instructions with your source program code. The option works for both the trace list and memory mnemonic displays.
`only`	This option displays only source lines. Processor instructions are only displayed in memory mnemonic if no source lines correspond to the instructions. Processor instructions are never displayed in the trace list.
symbols	
`off`	This prevents symbol display.
`on`	This displays symbols. This option works for the trace list, memory, software breakpoints, and register step mnemonics.
`high`	Displays only high level symbols, such as those available from a compiler. See the *Symbolic Retrieval Utilities User's Guide* for a detailed discussion of symbols.
`low`	Displays only low level symbols, such as those generated internally by a compiler, or an assembly symbol.
`all`	Displays all symbols.
`tabs_are`	This option allows you to define the number of spaces inserted for tab characters in the source listing.
`<TABS>`	Prompts you for the number of spaces to use in replacing the tab character. Values in the range of 2 through 15 may be entered.
trace	Sets update option for trace displays only.
update	When using multiple windows or terminals, and specifying this option, the display buffer in that window or terminal will be updated when a new measurement completes. This is the default. Note that for displays that show memory contents, the values will be updated when a command executes that changes memory contents (such as modify memory, load, and so on).

<VALUE> Specifies the logical value to which a particular UNIX environment variable is to be set.

width

`source` This allows you to specify the width (in columns) of the source lines in the memory mnemonic display. To adjust the width of the source lines in the trace display, increase the widths of the label and/or mnemonic fields.

`label` This lets you specify the address width (in columns) of the address field in the trace list or label (symbols) field in any of the other displays.

`mnemonic` This lets you specify the width (in columns) of the mnemonic field in memory mnemonics, trace list and register step mnemonics displays. It also changes the width of the status field in the trace list.

`<WIDTH>` This prompts you for the column width of the source, label, mnemonic, or symbols field.

Note that <CTRL>f and <CTRL>g may be used to shift the display left or right to display information which is off the screen.

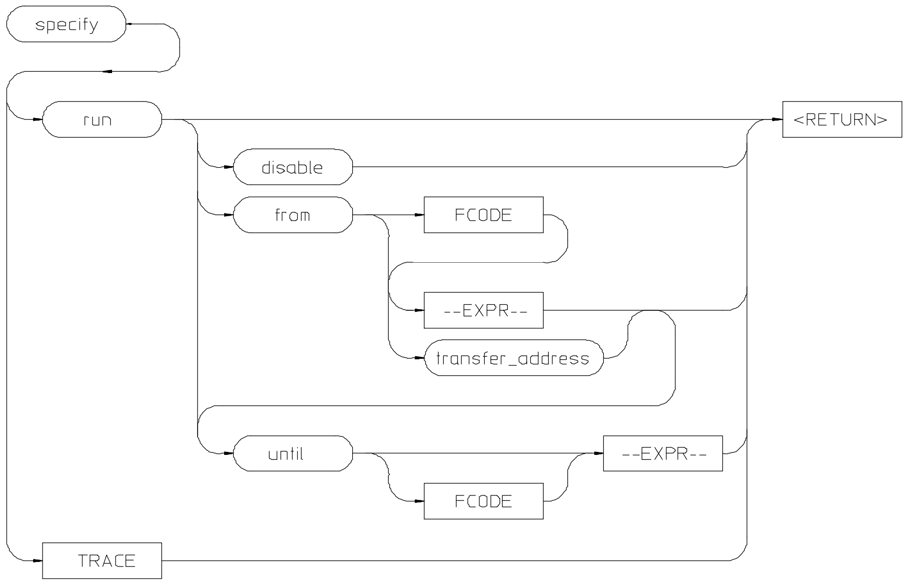
Examples

```
set source on inverse_video on tabs_are 2

set symbols on width label 30 mnemonic 20

set PRINTER = "lp -s"

set HP64KSYMBPATH=".file1:proc1
.file2:proc2:code_block_1"
```

See Also

The ***display data***, ***display memory***, ***display software_breakpoints***, and ***display trace*** commands.

specify



This command prepares a command for execution, and is used with the ***cmb_execute*** command.

When you precede a ***run*** or ***trace*** command with ***specify***, the system does not execute your command immediately. Instead, it waits until until an EXECUTE signal is received from the Coordinated Measurement Bus or until you enter a ***cmb_execute*** command.

If the processor is reset and no address is specified, a ***cmb_execute*** command will run the processor from the "reset" condition.

Note that the run specification is active until you enter specify ***run disable***. The trace specification is active until you enter another ***trace*** command without the specify prefix.

The emulator will run from the current program counter address if no address is specified in the command.

The parameters are as follows:

disable	This option turns off the specify condition of the run process.

from	
--EXPR--	This is used with the specify run from command. An expression is a combination of numeric values, symbols, operators, and parentheses, specifying a memory address. See the EXPR syntax diagram.
FCODE	The function code used to define the address space being referenced. See the syntax diagram for FCODE to see a list of the function codes available and for an explanation of those codes.
transfer_address	This is used with the specify run from command, and represents the address from which the program will begin running.
run	This option specifies that the emulator will run from either an expression or from the transfer address when a CMB EXECUTE signal is received.
TRACE	This option specifies that a trace measurement will be taken when a CMB EXECUTE signal is received.
until	Specifies an address where program execution is to stop. The emulator will set a software breakpoint at this address and stop execution of your program when it reaches this address and enter the monitor.

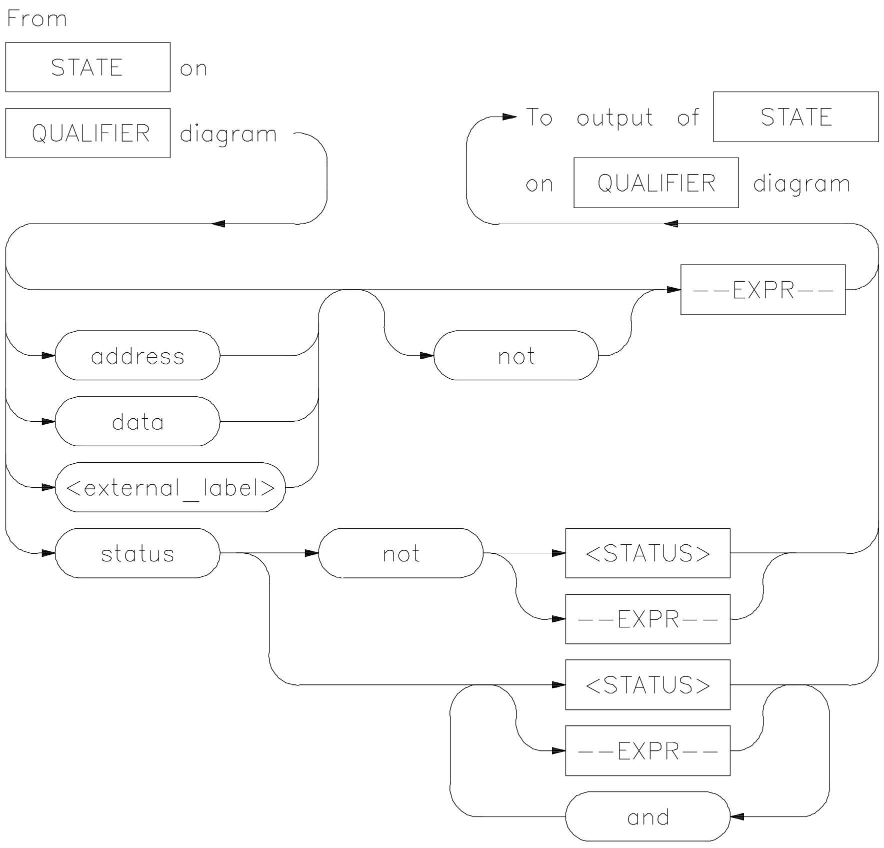
Examples

```
specify run from START

specify trace after address 1234H
```

See Also

The ***cmb_execute*** command.

STATE



This parameter lets you specify a trigger condition as a unique combination of address, data, status, and executed address values.

The STATE option is part of the QUALIFIER parameter to the trace command, and allows you to specify a condition for the trace measurement.

Refer to the "Qualifying Trigger and Store Conditions" section in the "Using the Emulation Analyzer" chapter for a list of the predefined values that can be assigned to the status state qualifiers.

The default STATE expression type is address.

The parameters are as follows:

address	This specifies that the expression following is an address value. This is the default, and is therefore not required on the command line when specifying an address expression.
and	This lets you specify a combination of status and expression values when status is specified in the state specification.
data	The value following this softkey is searched for on the lines that monitor the emulation processor's data bus.
--EXPR--	An expression is a combination of numeric values, symbols, operators, and parentheses, specifying an address, data, status, or executed address value. See the EXPR syntax diagram.
<external_label>	This represents a defined external analyzer label.
not	This specifies that the analyzer will search for the logical "not" of a specified state (this includes any address that is not in the specified state).
status	The value following this softkey is searched for on the lines that monitor other emulation processor signals.
<STATUS>	This prompts you to enter a status value in the command line. Status values can be entered from softkeys or typed into the keyboard. Numeric values may be entered using symbols, operators, and parentheses to specify a status value. See the EXPR syntax diagram.

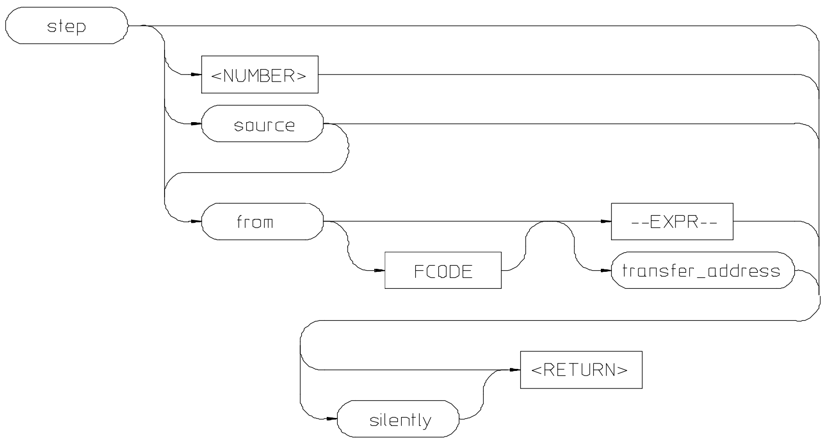
Examples

```
trace before status write

trace about address 1000H status write
```

See Also

The ***trace*** command and the QUALIFIER syntax description, and the trace command examples.

step



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

You can display the contents of the processor registers, trace memory, and emulation or target memory after each ***step*** command.

Source line stepping is implemented by single stepping assembly instructions until the next PC is beyond the address range of the current source line. When attempting source line stepping on assembly code (with no associated source line), stepping will complete when a source line is found. Therefore, stepping only assembly code may step forever. To abort stepping, press <CTRL>c.

When displaying memory mnemonic and stepping, the next instruction that will step is highlighted. The memory mnemonic display autopages to the new address if the next PC goes outside of the currently displayed address range. This feature works even if stepping is performed in a different emulation window than one displaying memory mnemonic.

If no value is entered for <NUMBER> times, only one step instruction is executed each time you press <RETURN>. Multiple instructions can be executed by holding down the <RETURN> key. Also, the default step is for assembly code lines, not source code lines.

If the from address option (defined by --EXPR-- or transfer_address) is omitted, stepping begins at the next program counter address.

The parameters are as follows:

--EXPR--	An expression is a combination of numeric values, symbols, operators, and parentheses specifying a memory address. See the EXPR syntax diagram.
FCODE	The function code used to define the address space being referenced. See the syntax diagram for FCODE to see a list of the function codes available and for an explanation of those codes.
from	Use this option to specify the address from which program stepping begins.
<NUMBER>	This defines the number of instructions that will be executed by the step command. The number of instructions to be executed can be entered in binary (B), octal (O or Q), decimal (D), or hexadecimal (H) notation.
silently	When you specify a number of steps, this option updates the register step mnemonic only after stepping is complete. This will speed up stepping of many instructions. The default is to update the register step mnemonic after each assembly instruction (or source line) executes (if stepping is performed in the same window as the register display).
transfer_address	This represents the starting address of the program you loaded into emulation or target memory. The transfer_address is defined in the linker map.
source	This option performs stepping on source lines.

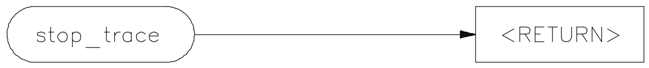
Examples

```
step

step from 810H

step 5 source

step 20 silently

step 4 from main
```

See Also

The ***display registers***, ***display memory mnemonic***, and ***set symbols*** commands.

stop_trace

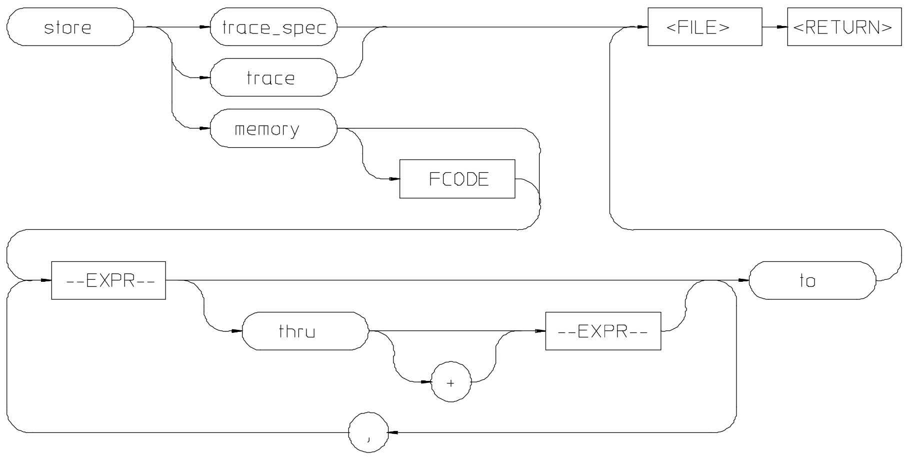


This command terminates the current trace and stops execution of the current measurement.

The analyzer stops searching for trigger and trace states. If trace memory is empty (no states acquired), nothing will be displayed.

See Also The ***trace*** command.

store



This command lets you save the contents of specific memory locations in an absolute file. You also can save trace memory contents in a trace file.

The ***store*** command creates a new file with the name you specify, if there is not already an absolute file with the same name. If a file represented by <FILE> already exists, you must decide whether to keep or delete the old file. If you respond with yes to the prompt, the new file replaces the old one. If you respond with no, the store command is canceled and no data is stored.

The transfer address of the absolute file is set to zero.

The parameters are as follows:

--EXPR--	This is a combination of numeric values, symbols, operators, and parentheses, specifying a memory address. See the EXPR syntax diagram.
FCODE	The function code used to define the address space being referenced. See the syntax diagram for FCODE to see a list of the function codes available and for an explanation of those codes.
<FILE>	This represents a file name you specify for the absolute file identifier or trace file where data is to be stored. If you want to name a file beginning with a number, you must precede the file name with a backslash (\) so the system will recognize it as a file name.
memory	This causes selected memory locations to be stored in the specified HP64000 format file with a .X extension.

thru	This allows you to specify that ranges of memory be stored.
to	Use this in the store memory command to separate memory locations from the file identifier.
trace	This option causes the current trace data to be stored in the specified file with a .TR extension.
trace_spec	This option stores the current trace specification in the specified file with a .TS extension.
,	A comma separates memory expressions in the command line.

Examples

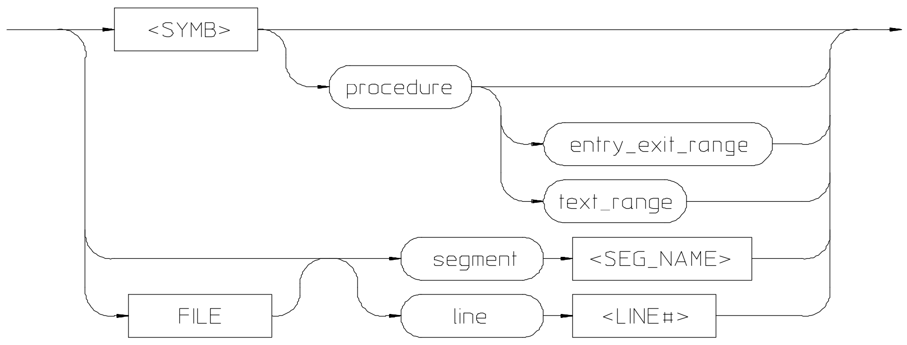
```
store memory 800H thru 20FFH to TEMP2

store memory EXEC thru DONE to \12.10

store trace TRACE

store trace_spec TRACE
```

See Also

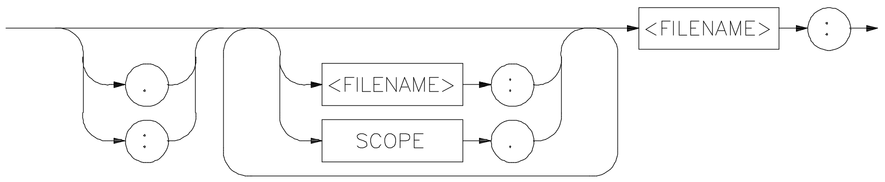
The ***display memory***, ***display trace***, and ***load*** commands.

--SYMB--

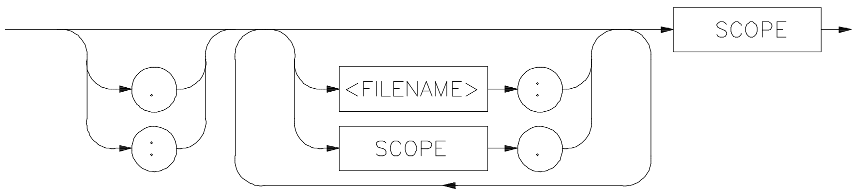
--SYMB--



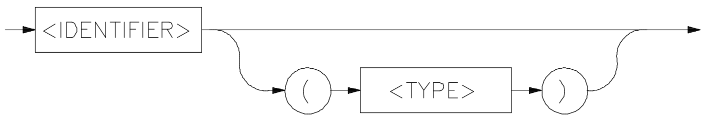
FILE



<SYMB>



SCOPE



This parameter is a symbolic reference to an address, address range, file, or other value.

Note that if no default file was defined by executing the command ***display local_symbols_in*** --SYMB--, or with the cws command, a source file name (<FILE>) must be specified with each local symbol in a command line.

Symbols may be:

- Combinations of paths, filenames, and identifiers defining a scope, or referencing a particular identifier or location (including procedure entry and exit points).
- Combinations of paths, filenames, and line numbers referencing a particular source line.
- Combinations of paths, filenames, and segment identifiers identifying a particular PROG, DATA or COMN segment or a user-defined segment.

The Symbolic Retrieval Utilities (SRU) handle symbol scoping and referencing. These utilities build trees to identify unique symbol scopes.

If you use the SRU utilities to build a symbol database before entering the emulation environment, the measurements involving a particular symbol request will occur immediately. If you then change a module and re-enter the emulation environment without rebuilding the symbol database, the emulation software rebuilds the changed portions of the database in increments as necessary.

Further information regarding the SRU and symbol handling is available in the *Symbolic Retrieval Utilities User's Guide*. Also refer to that manual for information on the HP64KSYMBPATH environment variable.

The last symbol specified in a ***display local_symbols_in*** --SYMB-- command, or with the cws command, is the default symbol scope. The default is "none" if no current working symbol was set in the current emulation session.

You also can specify the current working symbol by typing the cws command on the command line and following it with a symbol name. The pws command displays the current working symbol on the status line.

Display memory mnemonic also can modify the current working symbol.

The parameters are as follows:

<FILENAME>	This is an UNIX path specifying a source file. If no file is specified, and the identifier referenced is not a global symbol in the executable file that was loaded, then the default file is assumed (the last absolute file specified by a display local_symbols_in command). A default file is only assumed when other parameters (such as line) in the --SYMB-- specification expect a file.
line	This specifies that the following numeric value references a line number in the specified source file.
<LINE#>	Prompts you for the line number of the source file.
<IDENTIFIER>	Identifier is the name of an identifier as declared in the source file.
SCOPE	Scope is the name of the portion of the program where the specified identifier is defined or active (such as a procedure block).
segment	This indicates that the following string specifies a standard segment (such as PROG, DATA, or COMN) or a user-defined segment in the source file.
<SEG_NAME>	Prompts you for entry of the segment name.
(<TYPE>)	When two identifier names are identical and have the same scope, you can distinguish between them by entering the type (in parentheses). Do not type a space between the identifier name and the type specification. The type will be one of the following:
filename	Specifies that the identifier is a source file.
module	These refer to module symbols. For Ada, they are packages. Other language systems may allow user-defined module names.
procedure	Any procedure or function symbol. For languages that allow a change of scope without explicit naming, SRU assigns an identifier and tags it with type procedure.
static	Static symbols, which includes global variables. The logical address of these symbols will not change.
task	Task symbols, which are specifically defined by the processor and language system in use.
:	A colon is used to specify the UNIX file path from the line, segment, or symbol specifier. When following the file name with a line or segment selection, there must be a space after the colon. For a symbol, there must not be a space after the colon.

Examples

The following short C code example should help illustrate how symbols are maintained by SRU and referenced in your emulation commands.

File /users/dave/control.c:

```
int *port_one;
main () {
int port_value;

  port_ptr = port_one;
  port_value = 10;

  process_port (port_ptr, port_value);
} /* end main */
```

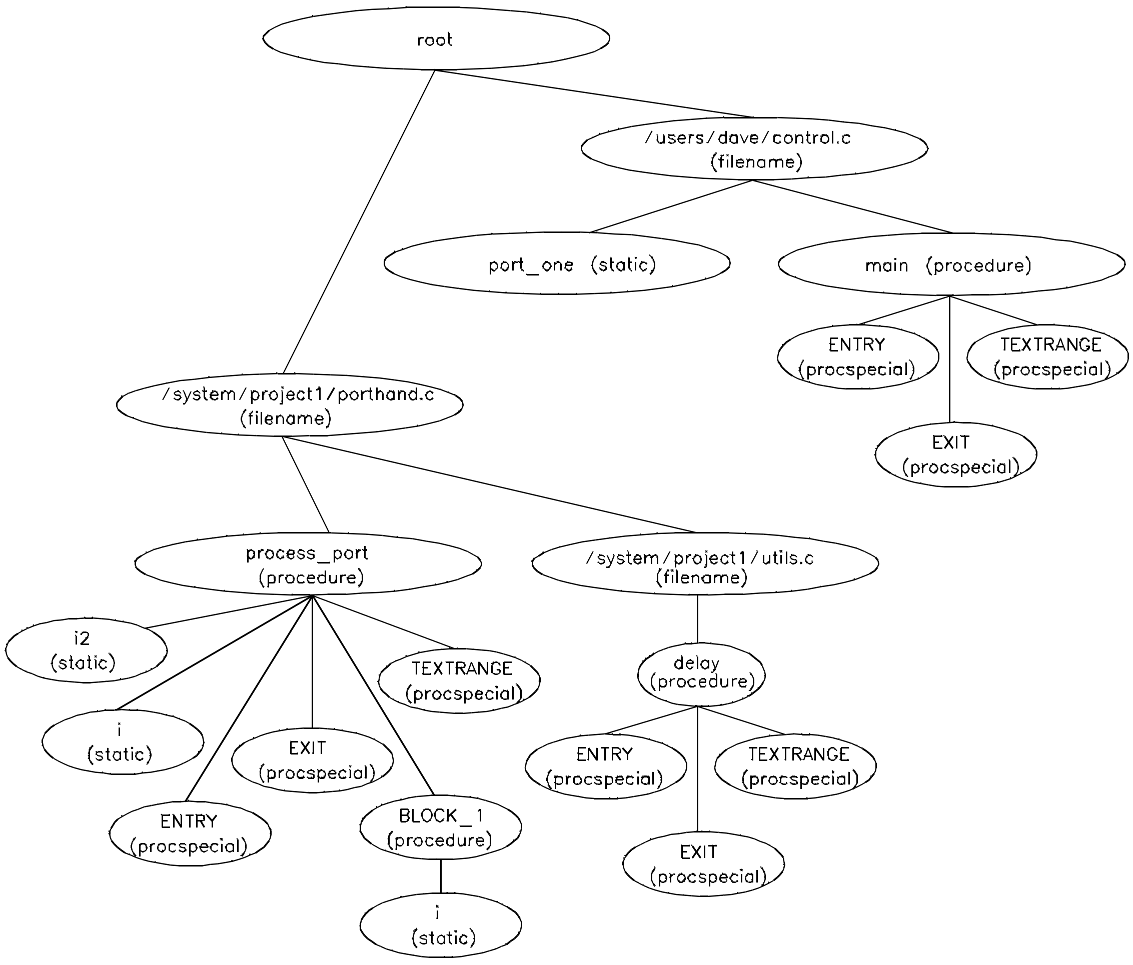
File /system/project1/porthand.c:

```
#include "utils.c"

void process_port (int *port_num, int port_data) {
static int i;
static int i2;

  for (i = 0; i <= 64; i++) {
    i2 = i * 2;
    *port_num = port_data + i2;
    delay();
      {
      static int i;
      i = 3;
      port_data = port_data + i;
      }
    }
} /* end of process_port */
```

File /system/project1/utils.c:

```
delay() {
int i,j;
int waste_time;

  for (i = 0; i <= 256000; i++)
    for (j = 0; j <= 256000; j++)
      waste_time = 0;
} /* end delay */
```

The symbol tree as built by SRU might appear as follows, depending on the object module format and compiler used:



Note that SRU does not build tree nodes for variables that are dynamically allocated on the stack at run-time, such as i and j within the delay () procedure. SRU has no way of knowing where these variables will be at run time and therefore cannot build a corresponding symbol tree entry with run time address.

Here are some examples of referencing different symbols in the above programs:

```
control.c:main
```

```
control.c:port_one
```

```
porthand.c:utils.c:delay
```

The last example above only works with IEEE-695 object module format; the HP object module format does not support referencing of include files that generate program code.

```
porthand.c:process_port.i
```

```
porthand.c:process_port.BLOCK_1.i
```

Notice how you can reference different variables with matching identifiers by specifying the complete scope. You also can save typing by specifying a scope with cws. For example, if you are making many measurements involving symbols in the file porthand.c, you could specify:

```
cws porthand.c:process_port
```

Then:

```
i
```

```
BLOCK_1.i
```

are prefixed with porthand.c: process_port before the database lookup.

If a symbol search with the current working symbol prefix is unsuccessful, the last scope on the current working symbol is stripped. The symbol you specified is then retested with the modified current working symbol. Note that this does not change the actual current working symbol.

For example, if you set the current working symbol as

```
cws porthand.c:process_port.BLOCK_1
```

and made a reference to symbol i2, the retrieval utilities attempt to find a symbol called

```
porthand.c:process_port.BLOCK_1.i2
```

which would not be found. The symbol utilities would then strip BLOCK_1 from the current working symbol, yielding

```
porthand.c:process_port.i2
```

which is a valid symbol.

You also can specify the symbol type if conflicts arise. Although not shown in the tree, assume that a procedure called port_one is also defined in control.c. This would conflict with the identifier port_one which declares an integer pointer. SRU can resolve the difference. You must specify:

```
control.c:port_one(static)
```

to reference the variable, and

```
control.c:port_one(procedure)
```

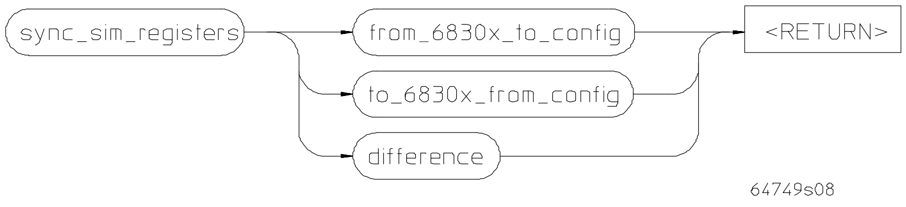
to reference the procedure address.

See Also

The copy local_symbols_in and display local_symbols_in commands.

Also refer to the *Symbolic Retrieval Utilities User's Guide* for further information on symbols.

sync_sim_registers



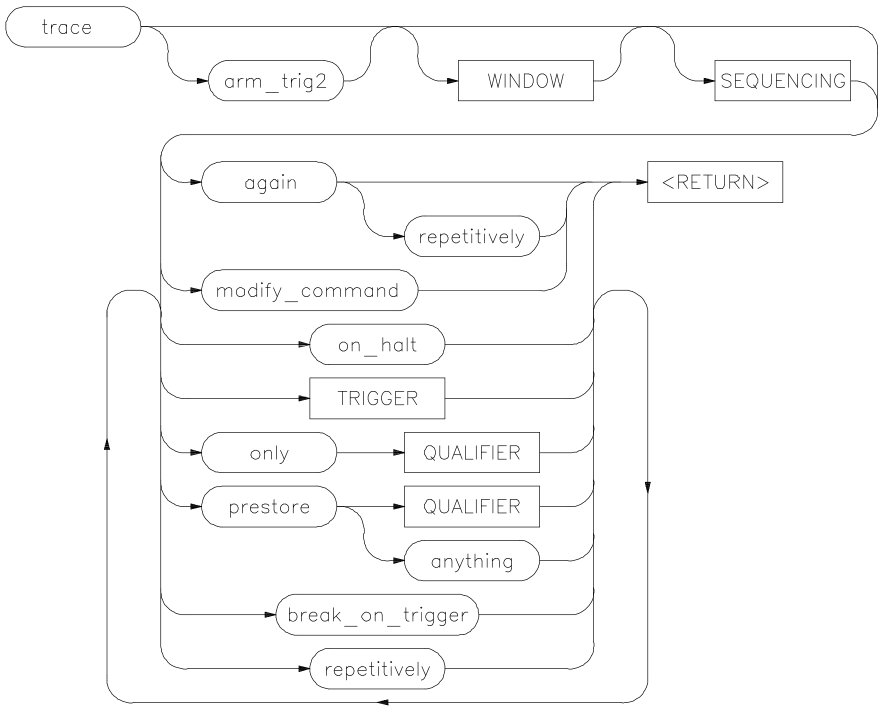
This command synchronizes the 6830x's system integration module (SIM) registers to the emulator's EMSIM registers.

The parameters are as follows:

from_6830x _to_config	Copies the microprocessor's SIM registers into the emulator's EMSIM registers.
to_6830x _from_config	Copies the emulator's EMSIM registers into the microprocessor's SIM registers.
difference	Displays the differences between the microprocessor's SIM registers and the emulator's EMSIM registers.

See Also The ***modify register*** commands and the "Concepts" chapter.

trace



This command allows you to trace program execution using the emulation analyzer.

Note that the options shown can be executed once for each ***trace*** command. Refer to the TRIGGER and QUALIFIER diagrams for details on setting up a trace.

You can perform analysis tasks either by starting a program run and then specifying the trace parameters, or by specifying the trace parameters first and then initiating the program run. Once a trace begins, the analyzer monitors the system busses of the emulation processor to detect the states specified in the trace command.

When the trace specification is satisfied and trace memory is filled, a message will appear on the status line indicating the trace is complete. You can then use ***display trace*** to display the contents of the trace memory. If a previous trace list is on screen, the current trace automatically updates the display. If the trace memory contents exceed the page size of the display, the <NEXT>, <PREV>, <Up arrow>, or <Down arrow> keys may be used to display all the trace memory contents. You also can press <CTRL>f and <CTRL>g to move the display left and right.

You can set up trigger and storage qualifications using the ***specify trace*** command. The analyzers will begin tracing when a ***cmb_execute*** command executes, which causes an EXECUTE signal on the Coordinated Measurement Bus.

The analyzer will trace any state by default.

The parameters are as follows:

again	This option repeats the previous trace measurement. It also begins a trace measurement with a newly loaded trace specification. (Using trace without the again parameter will start a trace with the default specification rather than the loaded specification.)
anything	This causes the analyzer to capture any type of information.
arm_trig2	This option allows you to specify the external trigger as a trace qualifier, for coordinating measurements between multiple HP 64700s, or an HP 64700 and another instrument. Before arm_trig2 can appear as an option, you must modify the emulation configuration interactive measurement specification. When doing this, you must specify that either BNC or CMBT drive trig2, and that the analyzer receive trig2. See the chapter on "Making Coordinated Measurements" for more information.
break_on_trigger	This stops target system program execution when the trigger is found. The emulator begins execution in the emulation monitor. When using this option, the on_halt option cannot be included in the command.
modify_command	This recalls the last trace command that was executed.
on_halt	When using this option, the analyzer will continue to capture states until the emulation processor halts or until a stop_trace command is executed. When this option is used, the break_on_trigger, repetitively, and TRIGGER options cannot be included in the command.
only	This option allows you to qualify the states that are stored, as defined by QUALIFIER.
prestore	This option instructs the analyzer to save specific states that occur prior to states that are stored (as specified with the "only" option).
QUALIFIER	This determines which of the traced states will be stored or prestored in the trace memory for display upon completion of the trace. Events can be

selectively saved by using trace only to enter the specific events to be saved. When this is used, only the indicated states are stored in the trace memory. See the QUALIFIER syntax.

repetitively	This initiates a new trace after the results of the previous trace are displayed. The trace will continue until a stop_trace or a new trace command is issued. When using this option, you cannot use the on_halt option.
SEQUENCING	Allows you to specify up to seven sequence terms including the trigger. The analyzer must find each of these terms in the given order before searching for the trigger. You are limited to four sequence terms if windowing is enabled. See the SEQUENCING syntax pages for more details.
TRIGGER	This represents the event on the emulation bus to be used as the starting, ending, or centering event for the trace. See the TRIGGER syntax diagram. When using this option, you cannot include the on_halt option.
WINDOW	Selectively enables and disables analyzer operation based upon independent enable and disable terms. This can be used as a simple storage qualifier. Or, you may use it to further qualify complex trigger specifications. See the WINDOW syntax pages for details.

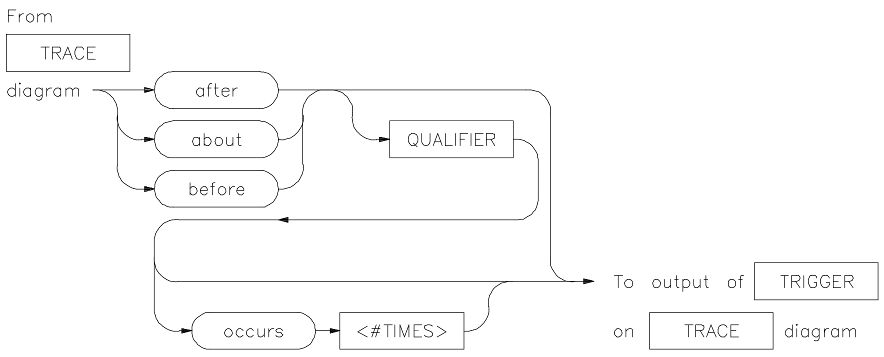
Examples

```
trace after 1000H

trace only address range 1000H thru 1004H

trace after address 1000H occurs 2 only address
range 1000H thru 1004H break_on_trigger
```

See Also

The ***copy trace***, ***display trace***, ***load trace***, ***load trace_spec***, ***specify trace***, ***store trace***, and ***store trace_spec*** commands.

TRIGGER



This parameter lets you define where the analyzer will begin tracing program information during a trace measurement.

A trigger is a QUALIFIER. When you include the occurs option, you can specify the trigger to be a specific number of occurrences of a QUALIFIER (see the QUALIFIER syntax diagram).

The default is to trace after any state occurs once.

The parameters are as follows:

about	This option captures trace data leading to and following the trigger qualifier. The trigger is centered in the trace listing.
after	Trace data is acquired after the trigger qualifier is found.
before	Trace data is acquired prior to the trigger qualifier.
occurs	This specifies a number of qualifier occurrences of a range or state on which the analyzer is to trigger.
QUALIFIER	This determines which of the traced states will be stored in trace memory.
<#TIMES>	This prompts you to enter a number of qualifier occurrences.

Examples

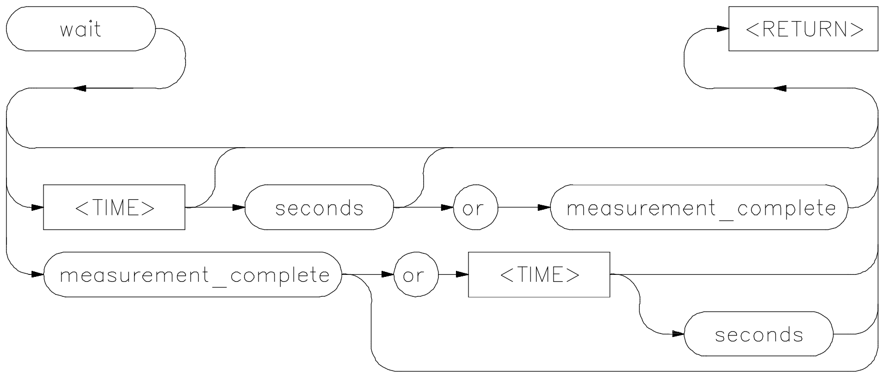
trace after `MAIN`

trace after `1000H` ***then data*** `5`

See Also

The ***trace*** command and examples.
Also, refer to the "Making Coordinated Measurements" chapter.

wait



This command allows you to present delays to the system.

The wait command can be an enhancement to a command file, or to normal operation at the main emulation level. Delays allow the emulation system and target processor time to reach a certain condition or state before executing the next emulation command.

The wait command does not appear on the softkey labels. You must type the wait command into the keyboard. After you type wait, the command parameters will be accessible through the softkeys.

The system will pause until it receives a <CTRL>c signal.

Note that if set intr <CTRL>c was not executed on your system, <CTRL>c normally defaults to the backspace key. See your UNIX system administrator for more details regarding keyboard definitions.

The parameters are as follows:

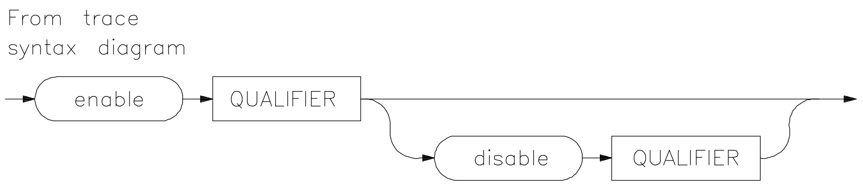
measurement _complete	This causes the system to pause until a pending measurement completes (a trace data upload process completes), or until a <CTRL>c signal is received. If a measurement is not in progress, the wait command will complete immediately.
or	This causes the system to wait for a <CTRL>c signal or for a pending measurement to complete. Whichever occurs first will satisfy the condition.
seconds	This causes the system to pause for a specific number of seconds.
<TIME>	This prompts you for the number of seconds to insert for the delay.

Note that a wait command in a command file will cause execution of the command file to pause until a <CTRL>c signal is received, if <CTRL>c is defined as the interrupt signal. Subsequent commands in the command file will not execute while the command file is paused. You can verify whether the interrupt signal is defined as <CTRL>c by typing set at the system prompt.

Examples

```
wait

wait 5; wait measurement_complete
```

WINDOW



Lets you select which states are stored by the analyzer.

WINDOW allows you to selectively toggle analyzer operation. When enabled, the analyzer will recognize sequence terms, trigger terms, and will store states. When disabled, the analyzer is effectively off, and only looks for a particular enable term.

You specify windowing by selecting an enable qualifier term; the analyzer will trigger or store all states after this term is satisfied. If the disable term occurs after the analyzer is enabled, the analyzer will then stop storing states, and will not recognize trigger or sequence terms. You may specify only one enable term and one disable term.

The analyzer defaults to recognizing all states. If you specify enable, you must supply a qualifier term. If you then specify disable, you must specify a qualifier term.

The parameters are as follows:

disable	Allows you to specify the term which will stop the analyzer from recognizing states once the enable term has been found.
enable	Allows you to specify the term which will enable the analyzer to begin monitoring states.
QUALIFIER	Specifies the actual address, data, status value or range of values that cause the analyzer to enable or disable recognition of states. Note that the enable qualifier can be different from the disable qualifier. Refer to the QUALIFIER syntax pages for further details on analyzer qualifier specification.

Examples

```
trace enable _rand disable 0ecch
```

See Also

The ***trace*** command and the SEQUENCING and QUALIFIER syntax descriptions.

13

Error Messages

Error Messages

This chapter alphabetically lists and describes the error messages displayed on the interface status line and in the error log. The *error log* records error messages received during the emulation session. You may want to display the error log to view the error messages. Sometimes several messages will be displayed for a single error to help you locate a problem quickly. To prevent overrun, the error log purges the oldest messages to make room for the new ones.

To display the error log, enter:

```
display error_log
```

Error messages are listed alphabetically by the first letter of the first word of the error messages. In the error log, some messages may be preceded by a number. Messages in this chapter that have an error message number associated with them have the number listed in parentheses following the message. These are ordered alphabetically by the first letter of the first word.

<message> (622)

Cause: Monitor specific message.

<CONFIGURATION FILENAME> does not exist

Cause: The configuration file you are trying to load does not exist.

Action: Try the load configuration command again using a valid configuration file name.

<LOGICAL NAME>: End, continuing

Cause: This is a status message. The emulation session is being exited with the end command. When you restart the emulation session later, it will continue using the same settings as in the session you just ended. The emulator logical name is located in the /usr/hp64000/etc/64700tab.net (or 64700tab) file.

<LOGICAL NAME>: End, released

Cause: This is a status message. The emulation session is being exited with the end release_system command. When the session has ended, the emulator is released, meaning that others can access and use it. When you restart the emulation session later, the new session will use all default settings. The emulator logical name is located in the /usr/hp64000/etc/64700tab.net (or 64700tab) file.

64700 command aborted (10371)

Cause: User abort occurred due when emulator is monopolized by another command.

Action: Don't issue an abort.

Address range too small for request - request truncated

Cause: Too small of an address range is specified in a modify memory command.

Action: Specify a larger memory range.

Adjust PC failed during break (600)

Cause: System failure or target condition.

Action: Run performance verification (Terminal Interface pv command), and check target system.

Ambiguous address: <address> (312)

Cause: Certain emulators support segmentation or function code information in addressing. The emulator is unable to determine which of two or more address ranges you are referring to, based upon the information you entered.

Action: Re-enter the command and fully specify the address, including segmentation or function code information.

Analyzer Break (613)

Cause: Status message.

Analyzer limitation; all range resources in use
Analyzer limitation; all pattern resources in use
Analyzer limitation; all expression resources in use (10360)

Cause: Your trace specification would use more than the maximum number of resources available to the analyzer.

Action: Simplify the trace specification.

Analyzer trace running (1304)

Cause: This error occurs when you attempt to change the external analyzer mode while a trace is in progress.

Action: Halt the trace before changing the external analyzer mode.

BNC trigger break (616)

Cause: This status message will be displayed if you have configured the emulator to break on a BNC trigger signal and the BNC trigger line is activated during a program run. The emulator is broken to the monitor.

Break caused by CMB not ready (611)

Cause: This status message is printed during coordinated measurements if the CMB READY line goes false. The emulator breaks to the monitor. When CMB READY is false, it indicates that one or more of the instruments participating in the measurement is running in the monitor.

Action: None, information only.

Break condition configuration aborted (653)

Cause: Occurs when <CTRL>c is entered during the configuration of break conditions.

Break due to cause other than step (689)

Cause: An activity other than a step command caused the emulator to break. This could include any of the break conditions or a <CTRL>c break.

Break failed (602)

Cause: The break command was unable to break the emulator to the monitor.

Action: Determine why the break failed, then correct the condition and retry the command. See message 608.

Breakpoint code already exists: <breakpoint_address> (667)

Cause: You attempted to insert a breakpoint; however, there was already a software breakpoint instruction at that location which was not already in the breakpoint table.

Action: Your program code is apparently using the same instructions as used by the software breakpoints feature. Remove the breakpoint instructions from your program code and use the modify software_breakpoints set command to insert them.

Breakpoint disable aborted (671)

Cause: Occurs when <CTRL>c is entered when disabling software breakpoints.

Breakpoint enable aborted (670)

Cause: Occurs when <CTRL>c is entered when setting software breakpoints.

Breakpoint list full; not added: <address> (664)

Cause: The software breakpoint table is already reached the maximum of 32 breakpoints. The breakpoint you just requested, with address <address>, was not inserted.

Action: Clear breakpoints that are no longer in use. Then, set the new breakpoint.

Breakpoint not added: <address> (668)

Cause: You tried to insert a breakpoint in a memory location which was not mapped or was mapped as guarded memory.

Action: Insert breakpoints only within memory ranges mapped to emulation or target RAM or ROM.

Breakpoint remove aborted (669)

Cause: Occurs when <CTRL>c is entered when clearing a software breakpoint.

Cannot create module file:

Cause: Insufficient disk space for the module file.

Action: Check disk space under /usr/hp64000.

Cannot default emulator; already in use (10332)

Cause: You tried to start an emulator interface, but your attempt failed because the emulator is already in use by someone else.

Action: Current user must release the emulator.

Cannot initialize PC and SSP (154)

Cause: When the emultor breaks to the monitor from reset, the emulator may try to initialize SSP as PC by reading memory reset vactor location (0,4). If a memory access fails, then this message will occur.

Note that you may specify the initial value of SSP and PC through the configuration menu.

Action: Initialize SSP and PC before running and/or use the configuation menu to changed the method of setting up SSP and PC.

Cannot interpret emulator output (10350)

Cause: There may be characters dropped in the information returned from the emulator.

Action: Ignore this message unless it becomes frequent. If it becomes frequent, you may have a fatal error; call your HP 64700 representative.

Cannot lock emulator; failure in obtaining the accessid
Cannot lock emulator; failure in <ERRNO MSG>

Cannot modify program counter to an odd value (164)

Cause: The emulator will not allow you to modify the content of the program counter to an odd value.

Cannot modify stack pointer to an odd value (163)

Cause: The emulator will not allow you to modify the stack pointer to an odd value.

Cannot start. Ending previous session, try again

Cause: The host system could not start a new emulation session, and is ending the previous session.

Action: After the previous session has ended, try starting a new emulation session. If that fails, try "emul700 -u <logical name>" to unlock the emulator and cycle power, if needed.

Cannot start. Pod initialization failed

Cause: The host system could not start a new emulation session because it could not initialize the emulator.

Action: Cycle power on the emulator; verify that there are no red lights on the front of the emulator. You may need to run the Terminal Interface "pv" command to verify that the emulator is functioning properly before starting a new session.

Cannot unlock emulator; emulator in use by user: <USER NAME> (10328)

Cause: The emulator is already in use by the named user.

Action: Current user must release the emulator.

Cannot unlock emulator; emulator not locked (10328)

Cause: You have issued a command to unlock an emulator that is not locked.

Action: The emulator is available now. You can start the interface.

Cannot unlock emulator; lock file missing
Cannot unlock emulator; semaphore missing (10328)

Cause: Lock semaphore missing.

Action: Verify existence and permissions of /usr/hp64000 directory. Cycle emulator power and use emul700 -u <logical name>.

Clock (CLKO) drive must be normal (159)

Cause: Bits 15 or 14 have been set to nonzero.

Action: Reset the bits.

CMB execute; emulation trace started (1305)

Cause: This status message informs you that an emulation trace measurement has started as a result of a CMB execute signal (as specified by the specify trace command).

CMB execute; run started (693)

Cause: This status message is displayed when you are making coordinated measurements. The CMB /EXECUTE pulse has been received; the emulation processor started running at the address specified by the specify run command.

Action: None; information only.

CMB execute break (623)

Cause: This message occurs when coordinated measurements are enabled and an EXECUTE pulse causes the emulator to run; the emulator must break before running.

Action: This is a status message; no action is required.

CMB trigger break (617)

Cause: This status message will be displayed if you have configured the emulator to break on a CMB trigger signal and the CMB trigger line is activated during a program run. The emulator is broken to the monitor.

Configuration aborted (624)

Cause: Occurs when a <CTRL>c is entered while emulator configuration items are being set.

Configuration failed; setting unknown: <item>=<setting> (626)

Cause: Target condition or system failure.

Action: Check target system, and run performance verification (Terminal Interface pv command).

Configuration not valid, restoring previous configuration
Configuration not valid, restoring default configuration

Cause: The modifications you tried to make to the emulator configuration are not valid, so the host system restored the previous configuration.

Action: See the "Configuring the Emulator" chapter for more information about the emulator configuration items and their settings.

Configuration process QUIT

Cause: The configuration process ended because <CTRL> "\" (SIGQUIT signal) was encountered. This is an easy way to exit configuration without saving any changes.

Action: Try starting the emulation session again. If the problem persists, you may need to cycle power on the emulator.

Connecting to <LOGICAL NAME>

Cause: This is a status message. The host system is making a communication connection to the emulator whose logical name is defined in /usr/hp64000/etc/64700tab.net or /usr/hp64000/etc/64700tab.

Continue load failed

Cause: The host system could not continue the previous emulation session because it could not load the continue file.

Action: Try again. If the failure continues, call your HP Service Representative.

Continuing previous session, continue file loaded

Cause: This is a status message. An emulation session which was ended earlier with the end command has been restarted. The host system reported that the session was continued (using settings from the previous session) and that the continue file loaded properly.

Continuing previous session, user interface defaulted

Cause: The previous emulation session was continued and the Softkey Interface was set to the default state.

Could not access emulation memory (161)

Cause: The cycle is hung, or CLKO has been turned off, or there is no clock processor.

Action: Reset the processor.

Could not create <CONFIGURATION BINARY FILENAME>

Cause: The system could not create a binary emulation configuration file (file.EB).

Action: Check the file.EB write permission and verify that the specified directory exists and is writeable.

Could not create default configuration

Cause: The host system could not create a default configuration for the emulation session.

Action: Check disk space under /usr/hp64000 and verify proper software installation.

Could not exec configuration process

Cause: The host system could not fork the configuration process or could not execute the configuration process.

Action: Make sure that the host system is operating properly, and that all Softkey Interface files were loaded properly during the installation process. Try starting the emulation session again.

Could not load default configuration

Cause: The host system could not load the default configuration into the emulator.

Action: Cycle power on the emulator and run the Terminal Interface "pv" (performance verification) command on the emulator to verify that it is functioning properly. Also, verify proper software installation. If loading default configuration still fails, then call your HP 64000 representative.

Count out of bounds: <value> (318)

Cause: You specified an occurrence count less than 1 or greater than 65535.

Action: Re-enter the command, specifying a count value from 1 to 65535.

Coverage not supported (180)

Cause: The memory coverage command cannot be used in this emulator because there is no supporting hardware.

DEMO/TEST BOARD not present or powered up (190)

Cause: Performance verification was attempted without the demo/test board present.

Action: Connect the demo/test board to the emulator.

Disable breakpoint failed: <address> (604)

Cause: System failure or target condition.

Action: Run performance verification (Terminal Interface pv command), and check target system.

Disable breakpoint failed: <address> (666)

Cause: System failure or target condition.

Action: Check memory mapping and configuration questions.

Display register failed: <register> (634)

Cause: The emulator was unable to display the register you requested.

Action: To resolve this, you must look at the other status messages displayed. It's likely that emulator was unable to break to the monitor to perform the register display. See message 608.

Don't care number unexpected

Cause: While defining an expression in your command, you included a don't care number (a binary, octal, decimal, or hexadecimal number containing "x"), which was not expected. Don't care numbers are not valid for all commands. See the EXPR command syntax for more information about expressions.

Emul700dmn continuation failed

Cause: Communication between the emulator and the host system to continue the emulation session failed.

Action: Check the data communication switch settings on the rear panel of the HP 64700 series emulator. If necessary, refer to the *HP 64700 Installation/Service Guide.*

Emul700dmn executable not found

Cause: The emulation session could not begin because the host system could not locate the HP 64700 emulator daemon process executable.

Action: Make sure that software installation is correct. Then try starting the emulator again.

Emul700dmn failed to start

Cause: The emulation session could not begin because the host system could not start the HP 64700 emulator daemon process.

Action: Make sure there is sufficient disk space under /usr/hp64000. Make sure the host system is operating properly, that all Softkey Interface software has been loaded correctly, and the data communication switch settings on the emulator rear panel match the settings in the /usr/hp64000/etc/64700tab.net (or 64700tab) file.

Emul700dmn message too large
Emul700dmn message too small
Emul700dmn queue and/or semaphores missing
Emul700dmn queue failure
Emul700dmn error in file operation
Emul700dmn queue full

Cause: The HP 64700 emulator daemon process command was too large for the host system to process.

Action: You must press end_release_system to exit this emulation session completely; then start a new session. Make sure the host system is operating properly, that all Softkey Interface software has been loaded correctly, and the data communication switch settings on the emulator rear panel match the settings in the /usr/hp64000/etc/64700tab.net (or 64700tab) file. You may have to cycle power and use emul700 -u ,logical name> to unlock the system.

Emul700dmn sem op failed, perhaps kernel limits too low

Cause: The host system could not start the emulation session; there may be too many processes running on the host system.

Action: Make sure the host system is operating properly, and is not overloaded with currently executing processes. Stop or remove some processes on the system. Also, verify that the semaphore capabilities have been installed in the UNIX kernel. Then try starting the emulation session again.

Emul700dmn version incompatible with this product

Cause: The emulation session could not begin because the version of the HP 64700 emulator daemon executable on host system is not compatible with the version of the Softkey Interface you are using.

Action: Make sure the software has been properly installed. Then try starting the emulator again.

Emulation analyzer defaulted to delete label

Cause: Analyzer trace labels were changed or modified while labels were in use in the trace specification.

Action: Enter the previous trace specification and try again.

Emulation memory access failed (702)

Cause: System failure.

Action: Run performance verification (Terminal Interface pv command).

Emulator locked by another user (10326)

Cause: This message occurs when you try to start an emulation interface, but your attempt failed because the emulator is being used by someone else.

Action: The current user must release the emulator.

Emulator locked by another user interface (10330)

Cause: You tried to start an emulator interface, but your attempt failed because the emulator is already in use by someone else.

Action: Current user must release the emulator.

Emulator locked by user: <USER NAME> (10329)

Cause: You tried to start an emulator interface, but your attempt failed because the emulator is already in use by someone else.

Action: Current user must release the emulator.

Emulator terminated hung bus cycle: <cycle> at <address>(164)

Cause: A hung bus cycle occurred during a memory access operation. This message indicates that the emulator detected the hung bus cycle and terminated it.

Action: Retry the command that caused the hung bus cycle. You may need to determine the source of termination (such as the processor, emulation memory, or target memory) and make required corrections.

Enable breakpoint failed: <breakpoint> (665)

Cause: System failure or target condition.

Action: Check memory mapping and configuration questions.

Ending released

Cause: This is a status message. The emulation session is being exited with the end release_system. The emulator will be released for others to access and use it.

Error: display size is <LINES> lines by <COLUMNS> columns. It must be at least 24 by 80.

Cause: You tried to specify an incorrect window size.

Action: Set the window size accordingly, then start the emulation session. The size of the window must be a minimum of 24 lines (rows) by 80 columns to operate an emulation session.

Error in configuration process
Error starting configuration process

Cause: Unexpected configuration error.

Action: Verify proper software installation and call your HP 64000 representative.

Exceeded maximum 64700 command line length (10351)

Cause: Your command is longer than 240 characters.

Action: Shorten the command.

Exceeded number of emulation memory terms available (142)

Cause: Too many emulation memory map terms have been used.

Action: reduc size or number of memory mapped terms.

External label in use: <label> (1301)

Cause: This error occurs when you attempt to select the external analyzer's independent state mode while an external trace label is currently used as a qualifier in the emulation analyzer trace specification.

Action: Remove any external trace label qualifiers from emulation trace specifications before selecting the external analyzer's independent state mode.

Failed to disable step mode (684)

Cause: System failure.

Action: Run performance verification (Terminal Interface pv command).

Fatal error from function <ADDRESS OF FUNCTION>

Cause: This is an unexpected fatal system error.

Action: Cycle power on the emulator and start again. If this is a persistent problem, call your HP 64000 representative.

FATAL SYSTEM SOFTWARE ERROR (204, 205, 208)

Cause: The system has encountered an error from which it cannot recover.

Action: Write down the sequence of commands which caused the error. Cycle power on the emulator and re-enter the commands. If the error repeats, call your local HP Sales and Service office for assistance.

File could not be opened

Cause: You tried to store or load trace data to a file with incorrect permission. Or the analyzer could not find the file you specified, or else there were already too many files open when you entered your command.

Action: Check the directory and file for correct read and write permission. Specify a file that is accessible to the analyzer. Close the other files that are presently open.

File perf.out does not exist

Cause: You tried to execute the "restore" command to continue a previous software performance measurement, and the SPMT software found that no "performance_measurement_end" command was previously executed to create a file from which "restore" could be performed.

Action: Execute a new SPMT measurement.

File perf.out not generated by measurement software

Cause: The file named perf.out exists in the current directory, but it was not created by the "performance_measurement_end" command.

Action: Rename the old "perf.out" file, or move it to another directory.

File transfer aborted (410)

Cause: A transfer operation was aborted due to a break received, most likely a <CTRL>c from the keyboard.

Action: If you typed <CTRL>c, you probably did so because you thought the transfer was about to fail. Retry the transfer, making sure to use the correct command options. If you are unsuccessful, make sure that the data communications parameters are set correctly on the host and on the HP 64700, then retry the operation.

Fuse F1 blown on HP64748C ABG Control Board (191)

Cause: Hardware fault.

Guarded memory access break (614)

Cause: This message is displayed if the emulation processor attempts to read or write memory mapped as guarded.

Action: Troubleshoot your program; or, you may have mapped memory incorrectly.

Guarded memory break: <address> (628)

Cause: A memory access to a location mapped as guarded memory has occurred during execution of the user program.

Action: Investigate the cause of the guarded memory access by the user program.

HP 64700 I/O error; communications timeout

Cause: This is a communication failure.

Action: Check power to the emulator and check that all cables are connected properly. If you are using LAN and heavy LAN traffic is present, try setting the environment variable to HP64700TIMEOUT="30" (or larger if needed). The value is the number of seconds before timeout occurs. Then try running again.

HP 64700 I/O error; power down detected

Cause: The emulator power was cycled.

Action: Do not do this during a user interface session; this may force the user interface to end immediately.

HP64700 I/O channel busy; communications timed out

Cause: The communications channel is in use for an unusually long period of time by another command.

Action: try again later.

HP64700 I/O channel in use by emulator: <LOGICAL NAME> (10331)

Cause: You tried to start an emulator interface, but your attempt failed because the emulator is already in use by someone else.

Action: Current user must release the emulator.

HP64700 I/O channel semaphore failure: <string>

Cause: Semaphore (ipc) facility not installed.

Action: Reconfigure the kernel to add ipc facility.

HP64700 I/O error; connection timed out

Cause: A user abort occurred while attempting to connect via LAN.

Action: Possibly connecting to an emulator many miles away, be patient.

Illegal status combination

Cause: You tried to specify combinations of status qualifiers in expressions incorrectly when entering commands.

Action: Refer to the "Emulator/Analyzer Interface Commands" chapter for information about syntax of commands.

Illegal symbol name

Cause: You tried to specify incorrect symbol names when entering commands.

Action: Specify correct symbol names. To see global symbol names, use the display global_symbols command. To see local symbol names, use the display local_symbols_in <SYMB> command.

Incompatible compatibility table entry (206)

Cause: The emulation firmware (ROM) is not compatible with the analysis or system firmware in your HP 64700 system.

Action: The ROMs in your emulator must be compatible with each other for your emulation system to work correctly. Contact your Hewlett-Packard Representative.

Incompatible with 64700 firmware version (10352)

Cause: The installed interface firmware combination is incorrect or incompatible.

Action: Upgrade the interface software of product firmware.

Initialization failed

Cause: The emulator could not be initialized.

Action: Make sure your data communication switch settings are correct, and that all Softkey Interface software has been loaded properly. Cycle power on the emulator, then try starting up the emulation session again.

Initialization load failed

Cause: The emulator could not be initialized.

Action: Make sure your data communication switch settings are correct, and that all Softkey Interface software has been loaded properly. Cycle power on the emulator, then try starting up the emulation session again.

Initializing emulator with default configuration

Cause: This is a status message. The host system started the emulation session and initialized the emulator using the default configuration. The emulator is probably operating correctly.

Initializing user interface with default config file

Cause: This is a status message. The host system started the emulation session and Softkey Interface using the default configuration file. The emulator is probably operating correctly.

Insufficient emulation memory, memory map may be incomplete

Cause: You can map only the amount of emulation memory available in your emulator. Trying to map additional unavailable memory may cause information to be missing from your memory map.

Action: Modify your configuration and update the memory map to correctly reflect the amount of emulation memory available.

Insufficient emulation memory (21)

Cause: You have attempted to map more emulation memory than is available, or you have attempted to include tag memory with a target system memory range when there is not enough emulation memory available.

Action: Reduce the amount of emulation memory or tag memory that you are trying to map.

Internal memory must not be mapped to 0 or 1000H

Cause: Register EMSIM EBAR set the internal memory to 0H or 1000H.

Action: Set the internal memory to another number.

Target may be asserting INTR 7 (156)

Cause: Target may be asserting interrupt 7 signals while the emulator is trying to break to the background monitor.

Action: Have the target negate interrupt level 7 or configure the emulator to disable target interrupts.

Invalid analysis subsystem; product address: <address> (902)

Cause: This error occurs when the HP 64700 system controller determines that the analysis firmware (ROM) is invalid.

Action: This message is not likely to occur unless you have upgraded the ROMs in your emulator. Be sure that the correct ROMs are installed in the analyzer board.

Invalid answer in <CONFIGURATION FILENAME> ignored

Cause: You must provide acceptable responses to questions in the configuration file (file.EA). The emulator ignored the incorrect response. Incorrect responses may appear in configuration files when you have saved the configuration to a file, edited it later, and tried reloading it into the emulator. This may also occur if you have loaded a configuration file that you created while using another emulator, and the response differs from the response required for this emulator.

Action: Examine your configuration file to check for inappropriate responses to configuration file questions.

Invalid attribute for memory type: <attribute> (140)

Cause: You tried to specify a memory attribute for target memory. Attributes are valid for emulation memory only, not for target or guarded memory.

Action: Re-enter your specification and use attributes only when specifying emulation memory space.

Invalid auxiliary subsystem; product address: <address> (904)

Cause: For future products.

Invalid ET subsystem; product address: <address> (903)

Cause: Detects an invalid ET. Used only internally.

Invalid firmware for emulation subsystem (901)

Cause: This error occurs when the HP 64700 system controller determines that the emulation firmware (ROM) is invalid.

Action: This message is not likely to occur unless you have upgraded the ROMs in your emulator. Be sure that the correct ROM is installed in the emulation controller.

Invalid trigger duration: <duration> (2031)

Cause: This error occurs when you attempt to specify an external timing trigger duration which is in the valid range but is not a multiple of 10 ns.

Action: Re-enter the command with the trigger duration as a multiple of 10 ns.

Invalid word access for odd address (2)

Cause: When the access mode is "word", you have attempted to modify a target system memory location at an odd address.

Action: Either change to an even address or modify the access mode to bytes if possible.

Invalid word access for odd number of bytes (3)

Cause: When the access mode is "word" and the display mode is "byte", you have attempted to modify a range of target system memory (perhaps, as small as a two byte range) with an odd number of byte values.

Action: Either specify an even number of bytes or modify the access mode to bytes if possible.

Inverse assembly file <INVERSE ASSEMBLER FILENAME> could not be loaded
Inverse assembly file <INVERSE ASSEMBLER FILENAME> not found, <filename>
Inverse assembly not available

Cause: The file does not exist.

Action: Reload your interface and/or real-time operating system software.

Inverse assembly not available

Cause: The inverse assembler for your emulator is missing.

Action: Verify proper software installation.

Joining session already in progress, continue file loaded

Cause: This is a status message. When operating the emulator in multiple windows, a new emulation session is "joined" to a current session. In this case, the new session was able to continue because the continue file loaded properly.

Joining session already in progress, user interface defaulted

Cause: When operating the emulator in multiple windows, a new emulation session is "joined" to a current session. In this case, the new session used the user interface default selections.

Lab firmware analysis subsystem; product address: <address> (912)

Cause: This message should never occur. It shows that you have an unreleased version of analysis firmware.

Lab firmware auxiliary subsystem; product address: <address> (914)

Cause: This message should never occur. It shows that you have an unreleased firmware version of the auxiliary subsystem.

Lab firmware for emulation subsystem (911)

Cause: This message should never occur. It shows that you have an unreleased version of emulation firmware.

Lab firmware subsystem; product address: <address> (913)

Cause: This message should never occur. It shows that you have an unreleased version of system controller firmware.

Load aborted

Cause: While loading a file into the emulator, an event occurred that caused the host system to stop the load process.

Action: Use the display error_log command to view any errors. If the problem persists, make sure the host system and emulator are operating properly, and that you are trying to load an acceptable file. See the "Emulator/Analyzer Interface Commands" chapter for information about the load command.

Load completed with errors

Cause: While loading a file into the emulator, one or more events occurred that caused errors during the load process.

Action: Use the display error_log command to view any errors. You may need to modify the configuration and map memory before you load the file again. If the problem persists, make sure the host system and emulator are operating properly, and that you are trying to load an acceptable file.

Logical emulator name unknown; not found in 64700tab file (10315)

Cause: This message may occur while trying to start up the emulator. It indicates that the emulator name specified could not be found in the 64700tab.net or /etc/hosts files.

Action: Specify the name in one of these files.

Map term n limited to 4 chip selects (165)

Cause: This status message occurs if more than 4 chip selects are assigned to a mapped term (as defined by the EMSIM registers: EMCSOR[0-10], EMCSBR[0-10], EMCSORBT, or EMCSBRBT). Only the first 4 chip selects will be allowed.

Measurement system not found

Cause: You tried to end the current emulation session and select another measurement system module which could not be located by the host system.

Action: Either try the end select measurement_system command again or end and release the emulation session.

Memory allocation failed, ending released

Cause: This is a fatal system error because the emulation session was unable to allocate memory.

Action: You may need to reconfigure your UNIX kernel to increase the per process maximum memory limit and available swap space. Reboot your UNIX system and try starting a new session again.

Memory modify aborted; next address: <address> (754)

Cause: This message is displayed if a break occurs during processing of a modify memory command. The break could result from any of the break conditions (except a software breakpoint) or could have resulted from a <CTRL>c break.

Action: Retry the operation. If breaks are occurring continuously, you may wish to disable some of the break conditions.

Memory range overflow

Cause: A modify memory command is attempted that would cross physical 0.

Action: Limit the modify memory command to not overflow physical 0 or break the command into two separate modify commands.

Memory range overflow (710)

Cause: Accessing a word or short word, for example display memory 0ffffffff blocked word will cause a rounding error that overflows physical memory.

Action: Reduce memory display request.

Message overflow (141)

Cause: The *display configuration_info diagnostics* command may emit more messages than HP 64700 will allow. This occurs when more than 16 messages have been emitted.

M6830x probe not connected or configured incorrectly (160)

Cause: The emulator probe hardware is not that for the M6830x emulator.

Action: make sure the three cables from the emulator control board to the emulator are properly connected. Make sure you are using the proper hardware.

Negated patterns not allowed in timing (2030)

Cause: This error occurs when you attempt to specify a "not equals" expression when defining the external timing trigger. You can only specify labels which equal patterns (of 1's, 0's, or X's).

Action: Do not attempt to specify negated timing patterns.

No address label defined

Cause: The address trace label was somehow removed in the terminal interface using the tlb command.

Action: End session and start again.

No more processes may be attached to this session

Cause: You can operate an emulator in four windows. Each time you start the emulator in another window, a new process is attached to the current session.

Action: Do not try to use more than four windows. Once you have started the emulator in four windows, you have reached the maximum number of processes allowed for that emulator.

No symbols loaded

Cause: You tried to step through lines in the source file before symbols are loaded.

Action: Load symbols and try again, or use step with the "source" option (i.e. step assembly language program).

No valid trace data

Cause: You tried to store trace data before a trace was completed.

Action: Wait until valid trace data is available before attempting to store a trace.

Not a valid trace file - load aborted

Cause: You tried to load a file.TR that was not created by the emulation session.

Action: Only load trace data files that were created by the emulator.

Not an absolute file
No absolute file: <file>
No absolute file, No database: <file>

Cause: You tried to load a file into the emulator that is not an executable or absolute file, so the host system stopped the load process.

Action: Try your command again, and make sure you specify a valid absolute file name to be loaded.

Not compatible trace file - load aborted

Cause: You tried to load a file.TR that was created by another type of emulator.

Action: Only load trace data files that were created by the same type of emulator.

Number of lines not in range: 1 <= valid lines <= 50

Cause: You tried to enter a number of lines that was outside the range from 1 to 50.

Action: Try entering the command again using a valid number of lines.

Number of spaces not in range: 2 <= valid spaces <= 15

Cause: You tried to enter a number of spaces outside the range from 2 to 15.

Action: Try entering the command again using a valid number of spaces.

opcode extends beyond specified address range

Cause: Memory disassembly is attempted on an address range that is too small.

Action: Display memory mnemonic using a large address range, or no address range at all.

perf.out file could not be opened - created

Cause: The performance analyzer failed to open or create a file named "perf.out" in response to your "performance_measurement_end" command.

Action: Free up some file space or correct the write permissions in your current working directory.

Perfinit - Absolute file (database) must be loaded line <LINE NUMBER>

Cause: No symbolic data base has been opened (or exists) for the target file when you executed the "performance_measurement_initialize" command.

Action: Make sure a data base has been loaded for the target file.

Perfinit - error in input file line <LINE NUMBER> invalid symbol

You included a "label" file name with your "performance_measurement_initialize" command, and that file contains an invalid symbol.

Action: Edit the file and correct the invalid symbol.

Perfinit - error in input file line <NUMBER>

Cause: You included an input file name with your "performance_measurement_initialize" command, and that file contains a syntax error.

Action: Edit the file and correct the syntax error.

Perfinit - File could not be opened

Cause: You specified a file as an option to "performance_measurement_initialize", and the file you specified could not be found or opened by SPMT software.

Action: Make sure you entered the correct file name.

Perfinit - No events in file

Cause: You specified a file along with your "performance_measurement"initialize" command that contained no events. Any measurement displayed from this file will have NULL results.

Action: Either edit the file to add events, or use the default setup to start a new measurement.

Perfinit <——EXPR—- ERROR> line <LINE NUMBER>

Performance tool must be initialized

Cause: You tried to make a performance measurement when the Software Performance Measurement Tool (SPMT) was not initialized.

Action: The Software Performance Measurement Tool (SPMT) must be initialized before making performance measurements on your software. Use the performance_measurement_initialize command to initialize the SPMT.

Performance tool not initialized

Cause: The Software Performance Measurement Tool (SPMT) has not been initialized.

Action: To make accurate activity or duration measurements on current data, use the performance_measurement_initialize command to initialize the SPMT before running a performance measurement.

Period not in 1/2/5 sequence: <period> (2021)

Cause: This error message occurs when the external timing sample period is not in a 1/2/5 sequence; for example, 10ns, 20ns, 50ns, 100ns, 200ns, 500ns, 1us, 2us, 5us, etc. Some examples of invalid sample period specifications are: 12ns, 18ns, 25ns, 60ns, 80ns, etc.

Action: Use a number in the 1/2/5 sequence when specifying the external timing sample period.

Program counter is odd (84)

Cause: You attempted to modify the program counter to an odd value using the modify registers command on a processor which expects even alignment of opcodes.

Action: Modify the program counter only to even numbered values.

Question file missing or invalid

Cause: Some of the Softkey User Interface files are missing or are corrupted.

Action: Reinstall the host software and try starting the emulation session again.

Range crosses segment boundary

Cause: On a segment offset processor, an address range is specified that would cross different segments.

Action: Break the memory command into multiple commands so that the address ranges start and end in the same segment.

Read memory failed at <PHYSICAL ADDRESS> - store aborted

Cause: While storing memory from the emulator to a file, a read memory error occurred.

Action: Use the display error_log command to view any errors. You may need to modify the configuration and map memory before storing the file again.

Read PC failed during break (603)

Cause: System failure or target condition.

Action: Try again.

Record checksum failure (400)

Cause: During a transfer operation, the checksum specified in a file did not agree with that calculated by the HP 64700.

Action: Retry the transfer operation. If the failure is repeated, make sure that both your host and the HP 64700 data communications parameters are configured correctly.

Records expected: <number>; records received: <number> (401)

Cause: The HP 64700 received a different number of records than it expected to receive during a transfer operation.

Action: Retry the transfer. If the failure is repeated, make sure that the data communications parameters are set correctly on the host and on the HP 64700.

Register access aborted (630)

Cause: Occurs when a <CTRL>c is entered during register display.

Register class cannot be modified: <register_class> (637)

Cause: You tried to modify a register class instead of an individual register.

Action: You can only modify individual registers. Refer to the display registers command description for a list of register names.

Register not writable: <register> (636)

Cause: This error occurs when you attempt to modify a read only register.

Action: If this error occurs, you cannot modify the contents of the register with the modify register command.

Request access to guarded memory: <address> (707)

Cause: The address or address range specified in the command included addresses within a range mapped as guarded memory. When the emulator attempts to access these during command processing, the above message is printed, along with the specific address or addresses accessed.

Action: Re-enter the command and specify only addresses or address ranges within emulation or target RAM or ROM. Or, you can remap memory so that the desired addresses are no longer mapped as guarded.

Restricted to real time runs (40)

Cause: While the emulator is restricted to real-time execution, you have attempted to use a command that requires a temporary break in execution to the monitor. The emulator does not permit the command and issues this error message.

Action: You must break the emulator's execution into the monitor before you can enter the command.

Retry limit exceeded, transfer failed (412)

Cause: The limit for repeated attempts to send a record during a transfer operation was exceeded, therefore the transfer was aborted.

Action: Retry the transfer. Make sure you are using the correct command options for both the host and the HP 64700. The data communications parameters need to be set correctly for both devices. Also, if you are in a remote location from the host, it is possible that line noise may cause the failure.

Run failed during CMB execute (694)

Cause: System failure or target condition.

Action: Run performance verification (Terminal Interface pv command), and check target system.

Sample period out of bounds: <bounds> (2022)

Cause: The external timing sample period must be between 10 ns and 50 ms (in a 1/2/5 sequence).

Action: Re-enter the command with the sample period between the bounds shown.

Session aborted

Cause: This will only happen when running multiple emulation windows and a fatal system error occurs.

Action: Find the window that caused the error and see the error message that it displayed. All the additional windows will simply state "session aborted". Cycle power on the emulator and enter emul700 -u <logical name> to make sure the emulator is unlocked.

Session cannot be continued, ending released

Cause: The emulation session is ending automatically because it could not be continued from the previous session. When the session has ended the emulator will be released, meaning that others can access and use it.

Action: When you restart the emulation session later, the new session will use all default settings.

Severe error detected, file transfer failed (411)

Cause: An unrecoverable error occurred during a transfer operation.

Action: Retry the transfer. If it fails again, make sure that the data communications parameters are set correctly on the host and on the HP 64700. Also make sure that you are using the correct command options, both on the HP 64700 and on the host.

Slave clock requires at least one edge

Cause: The analyzer has an invalid clock specification.

Action: Modify your configuration and try your command again.

Software breakpoint: <address> (615)

Cause: This status message will be displayed if a software breakpoint is encountered during a program run. The emulator is broken to the monitor. The string <address> indicates the address where the breakpoint was encountered.

Software breakpoint break condition is disabled (661)

Cause: You have attempted to set or clear a software breakpoint when software breakpoints are disabled.

Action: You must enable software breakpoints before you can set them.

Specified breakpoint not in list: <address> (663)

Cause: You tried to clear a software breakpoint that was not previously set. The string <address> prints the address of the breakpoint you attempted to clear.

Action: You must first set a software breakpoint before it can be cleared.

Starting address greater than ending address

Cause: You specified a starting address that is greater than the ending address.

Action: Specify a starting address that is less than or equal to the ending address.

Starting new session, continue file loaded

Cause: This is a status message. The emulator was started using a new emulation session, and the continue file loaded properly.

Starting new session, user interface defaulted

Cause: The emulator was started using a new emulation session, and the user interface was set to default selections.

Action: Call your HP Service Representative.

Status unknown, run "emul700 -l <LOGICAL NAME>"

Cause: The host system cannot determine the status of the emulator.

Action: To verify communication between the emulator and the host system, and display the emulator status, enter the emul700 -l <logical name> command. The emulator logical name is located in the /usr/hp64000/etc/64700tab.net (or 64700tab) file.

Step count must be 1 through 999

Cause: You tried to use a step count greater than 999.

Action: Use a step count less than 1000.

Step display failed (688)

Cause: System failure or target condition.

Action: Check memory mapping and configuration questions.

Stepping aborted; number steps completed: <number> (686)

Cause: This message is displayed if a break was received during a step command with a step count greater than zero. The break could have been due to any of the break conditions or a <CTRL>c break. The number of steps completed is displayed.

Stepping aborted; number steps completed: <STEPS TAKEN>

Cause: Stepping aborted because <CTRL>c or software breakpoint was hit, guarded memory was accessed, or some other kind of error occurred.

Action: See the error log display for any abnormal errors. Correct those errors and then step again.

Stepping complete

Cause: Stepping was completed successfully.

Stepping failed (680)

Cause: Stepping has failed for some reason.

Action: Usually, this error message will occur with other error messages. Refer to the descriptions of the accompanying error messages to find out more about why stepping failed.

Supervisor stack is set for ROM or guarded memory

Cause: The supervisor stack is set for inappropriate access to memory.

Action: Reset the stack.

Symbols not accessible, symbol database not loaded

Cause: You specified a trace list with values expressed using symbols defined in the source code modules, such as source on, and the database file has not been loaded into emulation. Example: display trace symbols on.

Target memory access failed (700)

Cause: This message is displayed if the emulator was unable to perform the requested operation on memory mapped to the target system.

Action: In most cases, the problem results from the emulator's inability to break to the monitor to perform the operation. See message 608.

Target memory access; bus error (155)

Cause: Target memory received a bus error when attempting to access the target memory.

Action: Change the processor chip select configuration, the emulation DTACK configuration, or target system to supply DTACK.

Timeout, receiver failed to respond (415)

Cause: Communication link or transfer protocol incorrect.

Action: Check link and transfer options.

Timeout in emul700dmn communication

Cause: The host system could not start the emulation session because the HP 64700 emulator process ran out of time before the emulator could start.

Action: You must press end_release_system to exit this emulation session completely; then start a new session. Make sure the host system is operating properly, that all Softkey Interface software has been loaded correctly, and the data communication switch settings on the emulator rear panel match the settings in the /usr/hp64000/etc/64700tab.net (or 64700tab) file.

Trace error during CMB execute (692)

Cause: System failure.

Action: Run performance verification (Terminal Interface pv command).

Trace file not found

Cause: You tried to load trace data file that does not exist.

Action: Find the correct name and path of the trace data file and try again.

Transfer failed to start (413)

Cause: Communication link or transfer protocol incorrect.

Action: Check link and transfer options.

trig1 break (618)

Cause: This status message will be displayed if you use the break_on_trigger syntax of the trace command and the analyzer has found the trigger condition while tracing a program run. The emulator is broken to the monitor.

trig2 break (619)

Cause: This status message will be displayed if you have used the internal trig2 line to connect the analyzer or external analyzer trigger output to the emulator break input and the analyzer has found the trigger condition. The emulator is broken to the monitor.

Trigger delay out of bounds: <bounds> (2042)

Cause: This error occurs when you attempt to specify an external timing trigger delay outside the valid range. The external timing trigger delay must be between 0 and 10 ms (in 10 ns increments).

Action: Re-enter the command with the trigger delay within the bounds shown.

Trigger duration out of bounds: <bounds> (2032)

Cause: This error occurs when you attempt to specify an external timing trigger duration outside the valid range. A "greater than" duration must fall within the range of 30 ns to 10 ms (and must be a multiple of 10 ns). A "less than" duration must fall within the range 40 ns to 10ms (and must be a multiple of 10 ns).

Action: Re-enter the command with the trigger duration within the bounds shown.

Unable to break (608)

Cause: This message is displayed if the emulator is unable to break to the monitor because the emulation processor is reset, halted, or is otherwise disabled.

Action: First, look at the emulation prompt and other status messages displayed to determine why the processor is stopped. If reset by the emulation controller, use the break command to break to the monitor. If reset by the emulation system, release that reset. If halted, try reset and break to get to the monitor. If there is a bus grant, wait for the requesting device to release the bus before retrying the command. If there is no clock input, perhaps your target system is faulty. It’s also possible that you have configured the emulator to restrict to real time runs, which will prohibit temporary breaks to the monitor.

Unable to configure break on software breakpoints (651)

Cause: The emulator controller cannot enable breakpoints, possibly because the emulator is in an unknown state or because of a hardware failure.

Action: Initialize the emulator or cycle power, then re-enter the command. If the same failure occurs, call your HP sales and service office.

Unable to configure break on write to ROM (650)

Cause: The emulator controller is unable to configure for breaks on writes to ROM, possibly because the emulator was left in an unknown state or because of a hardware failure.

Action: Initialize the emulator or cycle power. Then re-enter the command. If the same failure occurs, call your HP sales and service office.

Unable to delete label; used by emulation analyzer: <label> (1105)

Cause: This error occurs when you attempt to delete an emulation trace label which is currently being used as a qualifier in the emulation trace specification or is currently specified in the emulation trace format.

Action: You stop the trace or must change the trace command before you can delete the label.

Unable to delete label; used by external state analyzer: <label> (1106)

Cause: This error occurs when you attempt to delete an external trace label which is currently being used as a qualifier in the external state trace specification or is currently specified in the external trace format.

Action: You stop the trace or must change the trace command before you can delete the label.

Unable to delete label; used by external timing analyzer: <label> (1107)

Cause: This error occurs when you attempt to delete an external trace label which is currently being used as a qualifier in the external timing trace specification.

Action: Remove the label from the external timing analyzer specifications, and then delete the label.

Unable to load new memory map; old map reloaded (725)

Cause: There is not enough emulation memory left for this request.

Action: Reduce the amount of emulation memory requested.

Unable to modify register: <register>=<value> (632)

Cause: The emulator was unable to modify the register you requested.

Action: To resolve this, you must look at the other status messages displayed. It's likely that emulator was unable to break to the monitor to perform the register modification. See message 608.

Unable to read registers in class: <class> (631)

Cause: The emulator was unable to read the registers you requested.

Action: To resolve this, you must look at the other status messages displayed. Most likely, the emulator was unable to break to the monitor to perform the register read. See message 608.

Unable to redefine label; used by emulation analyzer: <label> (1108)

Cause: This error occurs when you attempt to redefine an emulation trace label which is currently used as a qualifier in the emulation trace specification.

Action: You stop the trace or must change the trace command before you can redefine the label.

Unable to redefine label; used by external state analyzer: <label> (1109)

Cause: This error occurs when you attempt to redefine an external trace label which is currently used as a qualifier in the external state trace specification.

Action: You stop the trace or must change the trace command before you can redefine the label.

Unable to redefine label; used by external timing analyzer: <label> (1110)

Cause: This error occurs when you attempt to redefine an emulation or external trace label which is currently being used as a qualifier in the external timing trace specification.

Action: Remove the label from the external timing analyzer specifications, and then redefine the label.

Unable to reload old memory map; hardware state unknown (726)

Cause: System failure.

Action: Run performance verification (Terminal Interface pv command).

Unable to reset (640)

Cause: Target condition or system failure.

Action: Check target system, and run performance verification (Terminal Interface pv command).

Unable to run (610)

Cause: System failure or target condition.

Action: Run performance verification (Terminal Interface pv command), and check target system.

Unable to run after CMB break (606)

Cause: System failure or target condition.

Action: Run performance verification (Terminal Interface pv command), and check target system.

Unable to run performance verification tests (191)

Cause: You entered the pv command but the emulator was unable to start performance verification because the firmware did not identifythe probe as being the MC6830x.

Action: Make sure the correct emulator probe is connected and that all cables are secured. make sure the demo board is connected to the emulator probe, and that the power cable is connected between the card cage and the demo board.

Undefined software breakpoint: <address> (605)

Cause: The emulator has encountered a software breakpoint in your program that was not inserted with the modify software_breakpoints set command.

Action: Remove the breakpoint instructions in your code before assembly and link.

Unexpected message from emul700dmn

Cause: The host system could not start the emulation session because of an unexpected message from the HP 64700 emulator process command.

Action: You must press end_release_system to exit this emulation session completely; then start a new session. Make sure the host system is operating properly, that all Softkey Interface software has been loaded correctly, and the data communication switch settings on the emulator rear panel match the settings in the /usr/hp64000/etc/64700tab.net (or 64700tab) file.

Unexpected software breakpoint (620)

Cause: If you have enabled software breakpoints, this message is displayed if a software breakpoint instruction is encountered in your program that was not inserted by a modify software_breakpoints set command and is therefore not in the breakpoint table.

Action: Remove the breakpoint instructions in your code before assembly and link, and use the modify software_breakpoints set command to reinsert them after the program is loaded into memory.

Unexpected step break (621)

Cause: System failure.

Action: Run performance verification (Terminal Interface pv command).

Unknown expression type

Cause: While entering your command, you included an unknown expression type.

Action: See the EXPR command syntax for more information about expressions. Then try entering your command again with a known expression type.

Unload trace data failed

Cause: Unexpected error occurred while waiting for a trace to be completed.

Action: End and release the session, and then try again.

Update HP 64740 firmware to version A.02.02 or newer (181)

Cause: This error occurred when you attempted to disassemble a trace and the analyzer firmware was found to be out of date.

Action: Refer to Chapter 16 "Installing/ Updating Emulator Firmware". You must update the firmware either to the version specified in the message or a newer firmware version.

Wait time failure, could not determine system time

Cause: The system call failed.

Action: Verify that 'date' executes correctly from the UNIX prompt.

Warning: at least one integer truncated to 32 bits
Warning: at least one integer truncated to 16 bits
Warning: at least one integer truncated to 8 bits

Cause: The number entered was too large for the currently specified display or access size.

Action: Try entering the command again using the correct size of number.

Width not in range: 1 <= valid width <= 80

Cause: You tried to specify the width of the field outside the range from 1 to 80.

Action: Try entering the command again using a valid number for the width.

Write to ROM break: <address> (628)

Cause: When the emulator is configured to break on writes to ROM, a memory write access to a location mapped as ROM has occurred during execution of the user program.

Action: Investigate the cause of the write to ROM by the user program. You can configure the emulator so that it does not break on writes to ROM.

Write to ROM break (612)

Cause: This status message will be printed if you have enabled breaks on writes to ROM and the emulation processor attempted a write to a memory location mapped as ROM.

Action: None (except troubleshooting your program).

Part 4

Concept Guide

Part 4

Topics that explain concepts and apply them to advanced tasks.

14

Concepts

Concepts

This chapter provides conceptual information on the following topics:

- X resources and the Graphical User Interface.
- Concepts of the EMSIM

X Resources and the Graphical User Interface

This section contains more detailed information about X resources and scheme files that control the appearance and operation of the Graphical User Interface. This section:

- Describes the X Window concepts surrounding resource specification.
- Describes the Graphical User Interface's implementation of scheme files.

X Resource Specifications

An X resource specification is a resource name and a value. The resource name identifies the element whose appearance or behavior is to be defined, and the value specifies how the element should look or behave. For example, consider the following resource specification:

```
Application.form.row.done.background: red
```

The resource name is "Application.form.row.done.background:" and the value is "red".

Resource Names Follow Widget Hierarchy

A *widget* is an OSF/Motif graphic device from which X applications are built. For example, pushbuttons and menu bars are Motif widgets. Applications are built using a hierarchy of widgets, and the application's X resource names follow this hierarchy. For example:

```
Application.form.row.done.background: red
```

In the resource name above, the top-level widget is named after the application. One of the top-level widget's children is a form widget, one of the form widget's children is a row-column manager widget, and one of the row-column manager widget's children is a pushbutton widget. Resource names show a path in the widget hierarchy.

Each widget in the hierarchy is a member of a widget class, and the particular instance of the widget is named by the application programmer.

Class Names or Instance Names Can Be Used

When specifying resource names, you can use either instance names or class names. For example, a "Done" pushbutton may have an instance name of "done" and a class name of "XmPushButton". To set the background color for a hypothetical "Done" pushbutton, you can use:

```
Application.form.row.done.background: red
```

Or, you can use:

```
Application.form.row.XmPushButton.background: red
```

Applications also have class and instance names. For example, an application may have an instance name of "applic1" and a class name of "Application". To set the background color for a hypothetical "Done" pushbutton only in the "applic1" application, you can use:

```
applic1.form.row.done.background: red
```

Note that instance names are more specific than class names. That is, class names may apply to many instances of the widget.

The class and instance names for the widgets in the Graphical User Interface can be displayed by choosing **Help→X Resource Names** and clicking on the "All names" button.

Wildcards Can Be Used

A wildcard may be used to match a resource specification to many different widgets at once. For example, to set the background color of all pushbuttons, you can use:

```
Application*XmPushButton.background: red
```

Note that resource names with wildcards are more general than those without wildcards.

Specific Names Override General Names

A more specific resource specification will override a more general one when both apply to a particular widget or application.

The names for the application and the main window widget in HP64_Softkey applications have been chosen so that you may specify custom resource values that apply in particular situations:

Apply to ALL HP64_Softkey applications:

HP64_Softkey*<resource>: <value>

Apply to specific types of HP64_Softkey applications:

emul*<resource>: <value> (for the emulator)
perf*<resource>: <value> (for the performance analyzer)

Apply to all HP64_Softkey applications, but only when they are connected to a particular type of microprocessor:

m6830x<resource>: <value> (for the 6830x)
m68020<resource>: <value> (for the 68020)

Apply to a specific HP64_Softkey application connected to a specific processor:

perf.m6830x*<resource>: <value> (for the 6830x perf. analyzer)
emul.m68020*<resource>: <value> (for the 68020 emulator)

If all four examples above are used for a particular resource, #3 will override #2 for all applications connected to a 6830x emulator, and #4 will override #2, but only for the specifically mentioned type of microprocessor.

When modifying resources, your resource paths must either match, or be more specific than, those found in the application defaults file.

How X Resource Specifications are Loaded

When the Graphical User Interface starts up, it loads resource specifications from a set of configuration files located in system directories as well as user-specific locations.

Application Default Resource Specifications

Default resource specifications for an application are placed in a system directory:

HP-UX	/usr/lib/X11/app-defaults
SunOS	/usr/openwin/lib/X11/app-defaults

The name of the Graphical User Interface application defaults file is HP64_Softkey (same as the application class name). This file is well-commented and contains information about each of the X resources you can modify. You can easily view this file by choosing **Help→Topic** and selecting the "X Resources: App Default File" topic. Do not modify the application defaults file; any changes to this file will affect the appearance and behavior of the application for all users.

User-Defined Resource Specifications

User-defined resources (for any X application) are located in the X server's RESOURCE_MANAGER property or in the user's $HOME/.Xdefaults file.

Load Order

Resource specifications are loaded from the following places in the following order:

The application defaults file. For example, /usr/lib/X11/app-defaults/HP64_Softkey when the operating system is HP-UX or /usr/openwin/lib/X11/app-defaults/HP64_Softkey when the operating system is SunOS.

The $XAPPLRESDIR/HP64_Softkey file. (The XAPPLRESDIR environment variable defines a directory containing system-wide custom application defaults.)

The server's RESOURCE_MANAGER property. (The xrdb command loads user-defined resource specifications into the RESOURCE_MANAGER property.)

If no RESOURCE_MANAGER property exists, user defined resource settings are read from the $HOME/.Xdefaults file.

The file named by the XENVIRONMENT environment variable.

If the XENVIRONMENT variable is not set, the $HOME/.Xdefaults-*host* file is read (typically contains resource specifications for a specific remote host).

Resource specifications included in the command line with the -xrm option.

When specifications with identical resource names appear in different places, the latter specification overrides the former.

Scheme Files

Several of the Graphical User Interface's X resources identify *scheme files* that contain additional X resource specifications. Scheme files group resource specifications for different displays, computing environments, and languages.

Resources for Graphical User Interface Schemes

There are five X resources that identify scheme files:

HP64_Softkey.labelScheme:

Names the scheme file to use for labels and button text. Values can be: Label, $LANG, or a custom scheme file name. The default uses the $LANG environment variable if it is set and if a scheme file named Softkey.$LANG exists in one of the directories searched for scheme files; otherwise, the default is Label.

HP64_Softkey.platformScheme:

Names the subdirectory for the platform specific color, size, and input scheme files. This resource should be set to the platform on which the X server is running (and displaying the Graphical User Interface) if it is different than the platform where the application is running. Values can be: HP-UX, SunOS, pc-xview, or a custom platform scheme directory name.

HP64_Softkey.colorScheme:

Names the color scheme file. Values can be: Color, BW, or a custom scheme file name.

HP64_Softkey.sizeScheme:

Names the size scheme file which defines the fonts and the spacing used. Values can be: Large, Small, or a custom scheme file name.

HP64_Softkey.inputScheme:

Names the input scheme file which specifies mouse and keyboard operation. Values can be: Input, or a custom scheme file name.

The actual scheme file names take the form: "Softkey.<value>".

Scheme File Names

There are six scheme files provided with the Graphical User Interface. Their names and brief descriptions of the resources they contain follow.

Softkey.Label	Defines the labels for the fixed text in the interface. Such things as menu item labels and similar text are in this file. If the $LANG environment variable is set, the scheme file "Softkey.$LANG" is loaded if it exists; otherwise, the file "Softkey.Label" is loaded.
Softkey.BW	Defines the *color scheme* for black and white displays. This file is chosen if the display cannot produce at least 16 colors.
Softkey.Color	Defines the *color scheme* for color displays. This file is chosen if the display can produce 16 or more colors.
Softkey.Large	Defines the *size scheme* (that is, the window dimensions and fonts) for high resolution displays (1000 pixels or more vertically).
Softkey.Small	Defines the *size scheme* (that is, the window dimensions and fonts) for low resolution displays (less than 1000 pixels vertically).
Softkey.Input	Defines the *input scheme* (that is, the button and key bindings for the mouse and keyboard).

Load Order for Scheme Files

Scheme files are searched for in the following directories and in the following order:

System scheme files in directory /usr/hp64000/lib/X11/HP64_schemes.

System-wide custom scheme files located in directory $XAPPLRESDIR/HP64_schemes.

User-defined scheme files located in directory $HOME/.HP64_schemes (note the dot in the directory name).

Custom Scheme Files

You can modify scheme files by copying them to the directory for user-defined schemes and changing the resource specifications in the file. For example, if you wish to modify the color scheme, and your platform is HP-UX, you can copy the /usr/hp64000/lib/X11/HP64_schemes/HP-UX/Softkey.Color file to $HOME/.HP64_schemes/HP-UX/Softkey.Color and modify its resource specifications.

You can create custom scheme files by modifying the X resource for the particular scheme and by placing the custom scheme file in the directory for user-defined schemes. For example, if the following resource specifications are made:

```
HP64_Softkey.platformScheme:  HP-UX
HP64_Softkey.colorScheme:     MyColor
```

The custom scheme file would be:

```
$HOME/.HP64_schemes/HP-UX/Softkey.MyColor
```

Concepts of the EMSIM

The 6830x rocessors contain a System Integration Block (SIB) which integrates various peripherals with the M68000 core. How certain parts of the SIB are programmed affect operation of the emulator. These emulator-sensitive parts of the SIB are referred to as the processor's System Integration Module (SIM).

The SIM contains the BAR and SCR registers along with the chip-select, port control, and interrupt control registers, plus any clock control registers, if present. For the 68302 emulator (64798C) the SIM registers are: BAR, SCR, BR0, OR0, BR1, OR1, BR2, OR2, BR3, OR3, PACNT, PADDR, PBCNT, PBDDR, and GIMR.

The programming of the SIM registers by the user affects how the emulator must be configured to operate properly. For example, the GIMR determines how interrupt level 7 is detected by the processor. The emulator uses interrupt level 7 to break to the monitor and so must be configured to what the processor expects. The chip select registers determine the DTACK source for a bus cycle within the range of a chip select. This information is needed for the emulator to properly complete bus cycles.

The EMSIM registers, which are an emulator version of the SIM registers, are used to configure the emulator hardware. The EMSIM registers are usually set to the "after initialization code" values desired for the SIM registers. By default the EMSIM registers contain the "processor reset" SIM values (refer to the appropriate *Motorola MC6830x User's Manual* for specific values). Therefore, the default programming of the emulator hardware matches the SIM reset values. For the 68302 emulator (64798C) the EMSIM registersare: EMBAR, EMSCR, EMBR0, EMOR0, EMBR1, EMOR1, EMBR2, EMOR2, EMBR3, EMOR3, EMPACNT, EMPADDR, EMPBCNT, EMPBDDR, and EMGIMR.

Some processors, such as the MC68LC302 do not connect certain upper address signals to the outside of the chip. For the MC68LC302 the upper four address bits (A20 through A23) which are sent to the emulation memory mapping hardware and the bus analyzer hardware must be re-created from the chip select signals. This "address regeneration" hardware in the emulator is programmed based on the EMSIM register set. The emulator needs access to the full 24-bit address bus which is maintained internally in the processor. If the "address regeneration" hardware is not properly programmed, that is, if the EMSIM register set is not kept correct, the emulator may show analyzer trace information incorrectly, and the emulator may not map emulation memory correctly.

EMSIM Utility Command

Modify→SIM Registers

This capability lets the user compare and transfer register values between the SIM and EMSIM register sets. Note even though the word "sim" is used in the command, all operations also include the RAM and EMRAM register sets.

Modify→SIM Registers→Copy Emulator SIM to Processor SIM

This transfers the current values of the EMSIM registers into the SIM registers. This happens automatically each time a break to the monitor from emulation reset occurs. This ensures that the processor is prepared to properly access memory when a program is downloaded to the emulator.

Modify→SIM Registers→Copy Processor SIM to Emulator SIM

This transfers the current values of the SIM registers into the EMSIM registers. This is useful if initialization code that configures the processor SIM exists, but you don't know its values. In this case, you can use the default configuration, run from reset to execute the initialization code, and then configure the emulator to match the processor SIM.

Display→SIM Register Differences

This shows current differences between the SIM registers and the EMSIM registers. This presents a list of all registers whose values are different between the SIM and the EMSIM. Use this to compare the programming between the SIM and EMSIM.

Display→Configuration Info

This displays information about the emulator configuration and processor SIM programming

Display→Configuration Info→Diagnostics

This checks the emulator configuration. Any inconsistencies and potential problems found during the check are listed. Resolve any items in the list to ensure correct operation of the emulator.

Display→Configuration Info→Chip Selects (SIM)

This displays chip selects in the SIM (processor) register set in a table. Use this to see how the SIM registers have configured the chip select pins of the processor.

Display→Configuration Info→Chip Selects (Emulator SIM)

This displays chip selects in the EMSIM (emulator) register set in a table. Use this to see how the EMSIM registers have configured the chip select pins of the emulation copy.

Display→Configuration Info→Bus Interface Ports (SIM)

This displays bus interface ports in the SIM (processor) register set in a table. Use this to see how the SIM registers have configured the external bus interface pins of the available ports.

Display→Configuration Info→Bus Interface Ports (Emulator SIM)

This displays bus interface ports in the EMSIM (emulator) register set in a table. Use this to see the SIM register values that will be loaded into the processor SIM when the monitor is entered from emulation reset.

Display→Configuration Info→Memory Map

This displays detailed information about the memory map in a table. Use this to check the way the memory map has been configured.

Display→Configuration Info→Reset Mode Value

This displays the reset mode configuration value and operation in a table. This is the value that will be driven onto the data bus to configure the processor when it comes out of reset. The meaning of each data bit in the value is shown.

Display→Configuration Info→Initialization Source Code

This displays the assembly language program to initialize the processor SIM and RAM based on the current contents of the EMSIM and EMRAM register sets.

Part 5

Installation Guide

Instructions for installing and configuring the product.

15

Installation

Installation

This chapter shows you how to install interface software. It also shows you how to verify installation by starting the emulator analyzer interface for the first time. These installation tasks are described in the following sections:

- Where to find information on connecting the HP 64700 to a computer or LAN.
- Installing HP 9000 software.
- Installing Sun SPARCsystem software.
- Verifying the installation.

For information about installing hardware, refer to the *MC6830x Emulator/Analyzer Installation/Service/Terminal Interface User's Guide*.

Minimum HP 9000 Hardware and System Requirements

The following is a set of minimum hardware and system recommendations for operation of the Graphical User Interface on HP 9000 Series 300/400 and Series 700 workstations.

HP-UX

For Series 9000/300 and Series 9000/400 workstations, the minimum supported version of the operating system is 7.03 or later. For Series 9000/700 workstations, the minimum supported version of the operating system is version 8.01.

Motif/OSF

For Series 9000/700 workstations, you must also have the Motif 1.1 dynamic link libraries installed. They are installed by default, so you do not have to install them specifically for this product, but you should consult your HP-UX documentation for confirmation and more information.

Hardware and Memory

Any workstation used with the Graphical User Interface should have a minimum of 16 megabytes of memory. Series 300 workstations should have a minimum performance equivalent to that of a HP 9000/350. A color display is also highly recommended.

From here, you should proceed to the section titled "Installation for HP 9000 Hosted Systems" for instructions on how to install, verify, and start the Graphical User Interface on HP 9000 systems.

Minimum Sun SPARCsystem Hardware and System Requirements

The following is a set of minimum hardware and system recommendations for operation of the Graphical User Interface on Sun SPARCsystem (SunOS or Solaris) workstations.

SunOS

The Graphical User Interface software is designed to run on a Sun SPARCsystem with SunOS version 4.1 or 4.1.1 or greater, and Solaris version 2.3. Each tape uses the QIC-24 data format.

64700 Operating Environment

The Graphical User Interface requires version A.04.10 or greater of the 64700 Operating Environment. (The Graphical User Interface version is A.04.00.)

Hardware and Memory

Any workstation used with the Graphical User Interface should have a minimum of 16 megabytes of memory. A color display is also highly recommended.

From here, you should proceed to the section titled "Installation for Sun SPARCsystems" for instructions on how to install, verify, and start the Graphical User Interface on SPARCsystem workstations.

Connecting the HP 64700 to a Computer or LAN

Refer to the *HP 64700 Series Installation/Service Guide* for instructions on connecting the HP 64700 to a host computer (via RS-422 or RS-232) or LAN and setting the HP 64700's configuration switches. (RS-422 and RS-232 are only supported on HP 9000 Series 300/400 machines.)

Installing HP 9000 Software

This section shows you how to install the Graphical User Interface on HP 9000 workstations. These instruction also tell you how not to install the Graphical User Interface if you want to use just the conventional Softkey Interface.

This section shows you how to:

- Install the software from the media.
- Verify the software installation.
- Start the X server and the Motif Window Manager (mwm), or start HP VUE.
- Set the necessary environment variables.

Step 1. Install the software from the media

The tape that contains the Graphical User Interface software may contain several products. Usually, you will want to install all of the products on the tape. However, to save disk space, or for other reasons, you can choose to install selected filesets.

If you plan on using the Softkey Interface instead of the Graphical User Interface, you can save about 3.5 megabytes of disk space by not installing the XUI suffixed filesets in the "64700 Operating Environment" and "<processor-type> Emulation Tools" partitions. (Also, if you choose not to install the Graphical User Interface, you will not have to use a special command line option to start the Softkey Interface.)

Refer to the information on updating HP-UX in your HP-UX documentation for instructions on viewing partitions and filesets and marking filesets that should not be loaded.

The following sub-steps assume that you want to install all products on the tape.

1 Become the root user on the system you want to update.

2 Make sure the tape's write-protect screw points to SAFE.

3 Put the product media into the tape drive that will be the *source device* for the update process.

4 Confirm that the tape drive BUSY and PROTECT lights are on. If the PROTECT light is not on, remove the tape and confirm the position of the write-protect screw. If the BUSY light is not on, check that the tape is installed correctly in the drive and that the drive is operating correctly.

5 When the BUSY light goes off and stays off, start the update program by entering

   /etc/update

   at the HP-UX prompt.

6 When the HP-UX update utility main screen appears, confirm that the source and destination devices are correct for your system. Refer to the information on updating HP-UX in your HP-UX documentation if you need to modify these values.

7 Select "Load Everything from Source Media" when your source and destination directories are correct.

8 To begin the update, press the softkey <Select Item>. At the next menu, press the softkey <Select Item> again. Answer the last prompt with

   y

   It takes about 20 minutes to read the tape.

9 When the installation is complete, read /tmp/update.log to see the results of the update.

Step 2. Set the necessary environment variables

The DISPLAY environment variable must be set before the Graphical User Interface will start. Also, you should modify the PATH environment variable to include the "/usr/hp64000/bin" directory, and, if you have installed software in a directory other than "/", you need to set the HP64000 environment variable.

The following instructions show you how to set these variables at the UNIX prompt. Modify your ".profile" or ".login" file if you wish these environment variables to be set when you log in. The following instructions also assume that you're using "sh" or "ksh"; if you're using "csh", environment variables are set using the "setenv <VARIABLE> <value>" command.

1 Set the DISPLAY environment variable by entering

DISPLAY=<hostname>:<server_number>.<screen_number>
export DISPLAY

For example:

DISPLAY=myhost:0.0; export DISPLAY

Consult the X Window documentation supplied with the UNIX system documentation for an explanation of the DISPLAY environment variable.

2 Set the HP64000 environment variable.

If you installed the software relative to a directory other than the root directory, it is strongly recommended that you use a symbolic link to make the software appear to be under /usr/hp64000. For example, if you installed the software relative to directory /users/team, you would enter

ln -s /users/team/usr/hp64000 /usr/hp64000

If you installed the HP 64000 software relative to the root directory, "/", or established a symbolic link to /usr/hp64000, then you would enter

HP64000=/usr/hp64000; export HP64000

If you did not install relative to the root directory, or do not wish to establish a symbolic link, you can set the HP64000 variable to the full path that contains the HP 64000 software. Again, if you installed relative to /users/team, you would enter

HP64000=/users/team/usr/hp64000; export HP64000

3 Set the PATH environment variable to include the **usr/hp64000/bin** directory by entering

PATH=$PATH:$HP64000/bin; export PATH

Including usr/hp64000/bin in your PATH relieves you from prefixing HP 64700 executables with the directory path.

4 Set the MANPATH environment variable to include the **$HP64000/man** and **$HP64000/contrib/man** directories by entering

MANPATH=$MANPATH:$HP64000/man:$HP64000/contrib/man
export MANPATH

Including these directories in your MANPATH variable lets you access the online "man" page information included with the software.

Step 3. Verify the software installation

A number of new filesets were installed on your system during the software installation process. This and following steps assume that you chose to load the Graphical User Interface filesets.

You can use this step to further verify that the filesets necessary to successfully start the Graphical User Interface have been loaded and that customize scripts have run correctly. Of course, the update process gives you mechanisms for verifying installation, but these checks can help to double-check the install process.

1 Verify the existence of the **HP64_Softkey** file in the **/$hp64000/X11/app-defaults** subdirectory by entering **ls /$hp64000/X11/app-defaults/HP64_Softkey** at the HP-UX prompt.

Finding this file verifies that you loaded the correct fileset and also verifies that the customize scripts executed because this file is created from other files during the customize process.

2 Examine **/$hp64000/X11/app-defaults/HP64_Softkey** near the end of the file to confirm that there are resources specific to your emulator.

Near the end of the file, there will be resource strings that contain references to specific emulators. For example, if you installed the Graphical User Interface for the 6830x emulator, resource name strings will have m6830x embedded in them.

After you have verified the software installation, you must start the X server and an X window manager (if you are not currently running an X server). If you plan to run the Motif Window Manager (mwm), or similar window manager, continue with Step 4a of these instructions. If you plan to run HP VUE, skip to Step 4b of these instructions.

Step 4. Start the X server and the Motif Window Manager (mwm)

If you are not already running the X server and a window manager, do so now. The X server is required to use the Graphical User Interface because it is an X Windows application. A window manager is not required to execute the interface, but, as a practical matter, you must use some sort of window manager with the X server.

- Start the X server by entering **x11start** at the HP-UX prompt.

Consult the X Window documentation supplied with the HP-UX operating system documentation if you do not know about using X Windows and the X server.

After starting the X server and Motif Window Manager, continue with step 2 of these instructions.

Step 5. Start HP VUE

If you are running the X server under HP VUE and have not started HP VUE, do so now.

HP VUE is a window manager for the X Window system. The X server is executing underneath HP VUE. Unlike the Motif Window Manager, HP VUE provides a login shell and is your default interface to the HP 9000 workstation.

Installing Sun SPARCsystem Software

This section shows you how to install the Graphical User Interface on Sun SPARCsystem workstations. These instructions also tell you how not to install the Graphical User Interface if you want to use just the conventional Softkey Interface.

This section shows you how to:

- Install the software from the media.
- Start the X server and OpenWindows.
- Set the necessary environment variables.
- Verify the software installation.
- Map your function keys.

Step 1. Install the software from the media

The tape that contains the Graphical User Interface software may contain several products. Usually, you will want to install all of the products on the tape. However, to save disk space, or for other reasons, you can choose to install selected filesets.

If you plan on using the conventional Softkey Interface instead of the Graphical User Interface, you can save about 3.5 megabytes of disk space by not installing the XUI suffixed filesets. (Also, if you choose not to install the Graphical User Interface, you will not have to use a special command line option to start the Softkey Interface.)

Refer to the *Software Installation Notice* for software installation instructions. After you are done installing the software, return here.

Step 2. Start the X server and OpenWindows

If you are not already running the X server, do so now. The X server is required to run the Graphical User Interface because it is an X application.

- Start the X server by entering **/usr/openwin/bin/openwin** at the UNIX prompt.

  Consult the OpenWindows documentation if you do not know about using OpenWindows and the X server.

Step 3. Set the necessary environment variables

The DISPLAY environment variable must be set before the Graphical User Interface will start. Also, you should modify the PATH environment variable to include the "usr/hp64000/bin" directory, and, if you have installed software in a directory other than "/", you need to set the HP64000 environment variable.

The following instructions show you how to set these variables at the UNIX prompt. Modify your ".profile" or ".login" file if you wish these environment variables to be set when you log in. The following instructions also assume that you're using "csh"; if you're using "sh", environment variables are set in the "<VARIABLE>=<value>; export <VARIABLE>" form.

1 The DISPLAY environment variable is usually set by the **openwin** startup script. Check to see that DISPLAY is set by entering

  echo $DISPLAY

  If DISPLAY is not set, you can set it by entering

  setenv DISPLAY=<hostname>:<server_number>.<screen_number>

  For example:

  setenv DISPLAY=myhost:0.0

  Consult the OpenWindows documentation for an explanation of the DISPLAY environment variable.

2 Set the HP64000 environment variable.

For example, if you installed the HP 64000 software relative to the root directory, "/", you would enter

```
setenv HP64000 /usr/hp64000
```

If you installed the software relative to a directory other than the root directory, it is strongly recommended that you use a symbolic link to make the software appear to be under /usr/hp64000. For example, if you installed the software relative to directory /users/team, you would enter

```
ln -s /users/team/usr/hp64000 /usr/hp64000
```

If you do not wish to establish a symbolic link, you can set the HP64000 variable to the full path that contains the HP 64000 software; also set the LD_LIBRARY_PATH variable to the directory containing run-time libraries used by the HP 64000 products. Again, if you installed relative to /users/team, you would enter

```
setenv HP64000 /users/team/usr/hp64000
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${HP64000}/lib
```

3 Set the PATH environment variable to include the **usr/hp64000/bin** directory by entering

```
setenv PATH ${PATH}:${HP64000}/bin
```

Including usr/hp64000/bin in your PATH relieves you from prefixing HP 64700 executables with the directory path.

4 Set the MANPATH environment variable to include the **usr/hp64000/man** and **usr/hp64000/contrib/man** directories by entering

```
setenv MANPATH ${MANPATH}:${HP64000}/man
setenv MANPATH ${MANPATH}:${HP64000}/contrib/man
```

Including these directories in your MANPATH variable lets you access the online "man" page information included with the software.

5 If the Graphical User Interface is to run on a SPARCsystem computer that is not running OpenWindows, include the /usr/openwin/lib directory in LD_LIBRARY_PATH.

```
setenv LD_LIBRARY_PATH
${LD_LIBRARY_PATH}:/usr/openwin/lib
```

Step 4. Verify the software installation

A number of product filesets were installed on your system during the software installation process. Due to the complexity of installing on NFS mounted file systems, a script that verifies and customizes these products was also installed. This stand alone script may be run at any time to verify that all files required by the products are in place in the file system. If required files are not found, this script will attempt to symbolically link them from the $HP64000 install directory to their proper locations.

- Run the script **$HP64000/bin/envinstall**.

Step 5. Map your function keys

If you are using the conventional Softkey Interface, map your function keys by following the steps below.

1 Copy the function key definitions by typing:

cp $HP64000/etc/ttyswrc ~/.ttyswrc

This creates key mappings in the .ttyswrc file in your $HOME directory.

2 Remove or comment out the following line from your .xinitrc file:

```
xmodmap -e 'keysym F1 = Help'
```

If any of the other keys F1-F8 are remapped using xmodmap, comment out those lines also.

3 Add the following to your .profile or .login file:

```
stty erase ^H
setenv KEYMAP sun
```

The erase character needs to be set to backspace so that the Delete key can be used for "delete character."

If you want to continue using the F1 key for HELP, you can use use F2-F9 for the Softkey Interface. All you have to do is set the KEYMAP variable. If you use OpenWindows, type:

setenv KEYMAP sun.2-9

If you use xterm windows (the xterm window program is located in the directory /usr/openwin/demo), type:

setenv KEYMAP xterm.2-9

Reminder: If you are using OpenWindows, add /usr/openwin/bin to the end of the $PATH definition, and add the following line to your .profile:

```
setenv OPENWINHOME /usr/openwin
```

After you have mapped your function keys, you must start the X server and an X window manager (if you are not currently running an X server).

Verifying the Installation

This section shows you how to:

- Determine the logical name of your emulator.
- Start the emulator/analyzer interface for the first time.
- Exit the emulator/analyzer interface.

Step 1. Determine the logical name of your emulator

The *logical name* of an emulator is a label associated with a set of communications parameters in the $HP64000/etc/64700tab.net file. The 64700tab.net file is placed in the directory as part of the installation process.

1 **Display the 64700tab.net file by entering** **more /usr/hp64700/etc/64700tab.net** **at the HP-UX prompt.**

2 **Page through the file until you find the emulator you are going to use.**

This step will require some matching of information to an emulator, but it should not be difficult to determine which emulator you want to address.

Examples

A typical entry for a 6830x emulator connected to the LAN would appear as follows:

```
#---------------------------------------------------------------------------
# Channel| Logical   | Processor | Remainder of Information for the Channel
#  Type  |  Name     |   Type    | (IP address for LAN connections)
#---------------------------------------------------------------------------
   lan:     em6830x     m6830x      21.17.9.143
```

A typical entry for a 6830x emulator connected to an RS-422 port would appear as follows:

```
#-------------------------------------------------------------------------------
#       |           |           |      |            |Xpar|Parity|Flow|Stop|Char
# Channel| Logical   | Processor | Host |  Physical  |Mode|      |    |Bits|Size
#  Type  |  Name     |   Type    | Name |   Device   |    |      |XON |    |
#       |           |           |      |            |OFF | NONE |RTS | 2  | 8
#-------------------------------------------------------------------------------
  serial:   em6830x     m6830x    myhost /dev/emcom23 OFF   NONE  RTS   2    8
```

Step 2. Start the interface with the **emul700** command

1 **Apply power to the emulator you wish to access after making sure the emulator is connected to the LAN or to your host system.**

On the HP 64700 Series Emulator, the power switch is located on the front panel near the bottom edge. Push the switch in to turn power on to the emulator.

2 **Wait a few seconds to allow the emulator to complete its startup initialization.**

3 **Choose a terminal window from which to start the Graphical User Interface.**

4 **Start the Graphical User Interface by entering emul700 command and giving the logical name of the emulator as an argument to the command, as in**

$HP64000/bin/emul700 <logical_name> &

or
emul700 <logical name> &

if **$HP64000/bin** is in your path.

If you are running the X server, if the Graphical User Interface is installed, and if your DISPLAY environment variable is set, the emul700 command will start the Graphical User Interface. Otherwise, emul700 starts the conventional Softkey Interface.

You should include an ampersand ("&") with the command to start the Graphical User Interface as a background process. Doing so frees the terminal window where you started the interface so that the window may still be used.

5 **Optionally start additional Graphical User Interface windows into the same emulation session by repeating the previous step.**

You can also choose to use the conventional Softkey Interface under X Windows, but you must include a command line argument to emul700 to override the default Graphical User Interface. Start the conventional interface by entering

```
emul700 -u skemul <logical name>
```

Example

Suppose you have discovered that the logical name for a 6830x emulator connected to the LAN is "em6830x". To start the Graphical User Interface and begin communicating with that emulator, enter (assuming your $PATH includes $HP64000/bin)

```
emul700 em6830x
```

After a few seconds, the Graphical User Interface Emulator/Analyzer window should appear on your screen. The window will be similar to the following:

Step 3. Exit the Graphical User Interface

1 **Position the mouse pointer over the pull-down menu named "File" on the menu bar at the top of the interface screen.**

2 **Press and hold the command select mouse button until the File menu appears.**

3 **While continuing to hold the mouse button down, move the mouse pointer down the menu to the "Exit" menu item.**

4 **Display the Exit cascade menu by moving the mouse pointer to the right edge of the Exit menu choice. There is an arrow on the right edge of the menu item.**

5 **Choose "Released" from the cascade menu.**

The interface will terminate and release the emulator for use by others.

16

Installing/Updating Emulator Firmware

Installing/Updating Emulator Firmware

The 6830x emulator firmware is included with the emulator/analyzer interface software, and the program that downloads emulator firmware is included with the HP B1471 64700 Operating Environment product.

(The firmware, and the program that downloads it into the control card, are also included with the 6830x emulator probe on an MS-DOS format floppies. The floppies are for users that do not have hosted interface software.)

Before you can update emulator firmware, you must have already installed the emulator into the HP 64700, connected the HP 64700 to a host computer or LAN, and installed the emulator/analyzer interface and HP B1471 software as described in the "Installation" chapter.

This chapter describes how to:

- Update firmware with the "progflash" command.
- Display current firmware version information.

To update emulator firmware with "progflash"

- Enter the **progflash -v <emul_name> <products ...>** command.

The progflash command downloads code from files on the host computer into Flash EPROM memory in the HP 64700.

The -v option means "verbose". It causes progress status messages to be displayed during operation.

The <emul_name> option is the logical emulator name as specified in the /usr/hp64000/etc/64700tab.net file.

The <products> option names the products whose firmware is to be updated.

If you enter the progflash command without options, it becomes interactive. If you don't include the <emul_name> option, it displays the logical names in the /usr/hp64000/etc/64700tab.net file and asks you to choose one. If you don't include the <products> option, it displays the products which have firmware update files on the system and asks you to choose one. (In the interactive mode, only one product at a time can be updated.) You can abort the interactive progflash command by pressing <CTRL>c.

Progflash will print "Flash programming SUCCEEDED" and return 0 if it is successful; otherwise, it will print "Flash programming FAILED" and return a nonzero (error).

You can verify the update by displaying the firmware version information.

Examples

To install or update the HP 64798 emulator firmware:

```
$ progflash <RETURN>


          HPB1471-19309 A.05.00 03Jan94
        64700 SERIES EMULATION COMMON FILES

         A Hewlett-Packard Software Product
         Copyright Hewlett-Packard Co. 1988


  All Rights Reserved. Reproduction, adaptation, or translation without prior
  written  permission  is prohibited, except as allowed under copyright laws.

                              RESTRICTED RIGHTS LEGEND

     Use , duplication , or disclosure  by the  Government is  subject to
     restrictions as set forth in subparagraph (c) (1) (II) of the Rights
     in Technical Data and Computer Software clause at  DFARS 52.227-7013.
     HEWLETT-PACKARD Company , 3000 Hanover St. , Palo Alto, CA 94304-1181

            Logical Name        Processor
          1 em68k               m68000
          2 em80960             i80960
          3 em6830x             m6830x

Number of Emulator to Update? (intr (usually cntl C or DEL) to abort)
```

To update firmware in the HP 64700 that contains the 6830x emulator, enter "3".

```
            Product
          1 64700
          2 64703/64704/64706/64740
          3 64744
          4 64798
          5 64760

Number of Product to Update? (intr (usually cntl C or DEL) to abort)
```

To update the HP 64798 6830x emulator firmware, enter "4".

```
Enable progress messages? [y/n] (y)
```

To enable status messages, enter "y".

```
Checking System firmware revision...
Mainframe is a 64700B

Reading configuration from '/usr/hp64000/inst/update/64798.cfg'
ROM identifier address = 2FFFF0H
Required hardware identifier = 1FFFH, 12FFH, 1201H, 1202H, 1203H, 1204H, 1205H,
1206H
Control ROM start address = 280000H
Control ROM size = 40000H
Control ROM width = 16
Programming voltage control address = 2FFFFEH
Programming voltage control value = FFFFH
Programming voltage control mask = 0H

Rebooting HP64700...
Checking Hardware id code...
Erasing Flash ROM
Downloading ROM code: /usr/hp64000/inst/update/64798.X
    Code start 280000H (should equal control ROM start)
    Code size 2348CH (must be less than control ROM size)
Finishing up...

Rebooting HP64700...
Flash programming SUCCEEDED
$
```

You could perform the same update as in the previous example with the following command:

$ **progflash -v** em6830x 64798 <RETURN>

To display current firmware version information

- **Use the Terminal Interface ver command to view the version information for firmware currently in the HP 64700.**

When using the Graphical User Interface or Softkey Interface, you can enter Terminal Interface commands with the pod_command command. For example:

```
display pod_command <RETURN>
pod_command "ver" <RETURN>
```

Examples The Terminal Interface ver command displays information similar to:

```
                  Copyright (c) Hewlett-Packard Co. 1987
All Rights Reserved.  Reproduction, adaptation, or translation without prior
written permission is prohibited, except as allowed under copyright laws.

  HP64700B Series Emulation System
    Version:   B.01.00 20Dec93
    Location:  Flash
    System RAM:1 Mbyte

  HP64798C Motorola 68302 Emulator
    Version:   A.03.00
    Control:   HP 64748C Emulation Control Board
    Speed:     20.1 MHz
    Memory:    1024 Kbytes
    BANK 0:    HP 64171B (35 ns) or HP 64172B (20 ns) 1 MByte Memory Module

  HP64740 Emulation Analyzer
    Version:   A.02.02 13Mar91
```

If there is a power failure during a firmware update

If there is a power glitch during a firmware update, some bits may be lost during the download process, possibly resulting in an HP 64700 that will not boot up.

☐ Repeat the firmware update process.

☐ If the HP 64700 is connected to the LAN in this situation and you are unable to connect to the HP 64700 after the power glitch, try repeating the firmware update with the HP 64700 connected to an RS-232 or RS-422 interface.

Glossary

absolute file	This file contains machine-readable instructions and/or data. the instructions and/or data are stored at absolute addresses. Absolute files are generated by the compiler/assembler/linker. These files are loaded into memory for execution by the target processor.
access mode	Specifies the types of cycles used to access target system memory locations. For example a "byte" access mode tells the monitor program to use load/store byte instructions to access target memory.
	arm condition
	A condition that reflects the state of a signal external to the analyzer. The arm condition can be used in branch or storage qualifiers. External signals can be from another analyzer or an instrument connected to the CMB or BNC.
analyzer	An instrument that captures data on signals of interest at discreet periods.
background	The emulator mode in which foreground operation is suspended so the emulation processor can be used for communication with the emulation controller. The background monitor does not occupy any processor address space.
background emulation monitor	An emulation monitor program that does not execute as part of the user program, and therefore, operates in the emulator's background mode.
background memory	Memory space reserved for the emulation processor when it is operating in the background mode. Background memory does not take up any of the microprocessor's address space.
BNC connector	A connector that provides a means for the emulator to drive/receive a trigger/signal to/from an external device (such as a logic analyzer, oscilloscope, or HP 64000-UX system).
breakpoint	A point at which emulator execution breaks from the target program and begins executing in the monitor.
compiler	A program that translates high-level language source code into object code, or produces an assembly language program with subsequent translation into object code by an assembler. Compilers typically generate a program listing which may list errors displayed during the translation process.

data segment	A segment that contains data (other than immediate data) for an executable segment. A data segment is identified by a specific type code in the descriptor of the segment.
display mode	When displaying memory, this mode tells the emulator the size of the memory locations to display. When modifying memory, the display mode tells the emulator the size of the values to be written to memory.
embedded microprocessor system	The microprocessor system which the emulator plugs into.
emulation/analysis system	A set of hardware and software capable of performing emulation functions on a target system that uses a particular microprocessor.
emulation bus	The emulation bus contains all of the signals on the pins of the emulation microprocessor.
emulation bus analyzer	The internal analyzer that captures emulator bus cycle information synchronously with the processor's clock signal.
emulation memory	This is memory space that resides in your emulator hardware.
emulation memory map	The emulation memory map defines the addresses supported by memory hardware during emulation. You set up this map during the emulation configuration process. In it, you assign hardware memory in your emulator, and/or in your target system, to support ranges of addresses. In this map, you also define the behavior of the memory so that it will act as RAM hardware or ROM hardware to emulate the type of memory you intend to install in your target system when its design is complete.
emulation monitor program	A program that is executed by the emulation processor which allows the emulation controller to access target system resources. For example, when you display target system memory locations, the monitor program executes microprocessor instructions that read the target memory locations and send their contents to the emulation controller.
emulation probe	The cable that connects the emulator to the target system microprocessor socket.
emulation processor	The emulation processor is the processor that replaces the target system processor during an emulation session. The emulation processor is pat of the emulation probe.

`emulator`	An instrument that performs just like the microprocessor it replaces, but at the same time, it gives you information about the operation of the processor. An emulator gives you control over target system execution and allows you to view or modify the contents of processor registers, target system memory, and I/O resources.
`entry point`	An executable segment offset that identifies the starting point for execution, as when the segment is invoked via a gate.
`external analyzer`	An analyzer that captures activity on signal nodes external to the emulation processor bus.
`external analyzer probe`	A set of signal lines that connect the external analyzer to target system signals.
`external clock`	Any clock other than the clock source of the emulator. Typically, the clock of the target system is used as an external clock for emulation tests and measurements.
`foreground`	The mode in which the emulator is executing the user program. In other words, the mode in which the emulator operates as the target microprocessor would.
`glitch`	This is the name assigned to the detection of at least one transition in both directions between any two sampling clock pulses during a timing measurement.
`global restart`	When the same secondary branch condition is used for all terms in the analyzer's sequencer, and secondary branches are always back to the first term.
`host computer`	A computer to which an HP emulator can be connected. A host computer may run interface programs that control the emulator. Host computers may also be used to develop programs to be downloaded into the emulator.
`instrumentation card cage`	The hardware frame and power supply built to accept installation of emulation and analysis board assemblies, and to provide interconnections for boards and interconnections for development stations that control the development hardware.
`monitor`	The monitor is a collection of routines that perform many of the functions needed in an emulator, such as displaying the content of registers or loading code into memory so that the code can be executed during a test.
`pod commands`	Another name for Terminal Interface commands. (The Terminal Interface is the low-level interface that resides in the firmware of the emulator.) Pod commands bypass the graphical interface and go directly to the emulator.

`prestore`	The analyzer feature that allows up to two states to be stored before normally stored states. This feature is useful when you want to find the cause of a particular state. For example, if a variable is accessed from many different places in the program, you can qualify the trace so that only accesses of that variable are stored and turn on prestore to find out where accesses of that variable originate from.
`primary sequencer branch`	Occurs when the analyzer finds the primary branch state specified at a certain level and begins searching for the states specified at the primary branch's destination level.
`real time`	Refers to continuous execution of the user 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 or I/O.)
`secondary sequencer branch`	Occurs when the analyzer finds the secondary branch state specified at a certain level before it found the primary branch state and begins searching for the states specified at the secondary branch's destination level.
`sequence terms`	Individual levels of the sequencer. The HP 64705A analyzer provides eight sequence terms.
`sequencer`	The part of the analyzer that allows it to search for a certain sequence of states before triggering.
`sequencer branch`	Occurs when the analyzer finds the primary or secondary branch state specified at a certain level and begins searching for the states specified at another level.
`target system`	The microprocessor system which the emulator plugs into.
`target system memory`	This is memory space that resides within your target system hardware.
`trace`	A collection of states captured on the emulation bus (in terms of the emulation bus analyzer) or on the analyzer trace signals (in terms of the external analyzer) and stored in trace memory.
`trigger`	The captured analyzer state about which other captured states are stored. The trigger state specifies when the trace measurement is taken.

Index

Safety

Summary of Safe Procedures

The following general safety precautions must be observed during all phases of operation, service, and repair of this instrument. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the instrument. Hewlett-Packard Company assumes no liability for the customer's failure to comply with these requirements.

Ground The Instrument

To minimize shock hazard, the instrument chassis and cabinet must be connected to an electrical ground. The instrument is equipped with a three-conductor ac power cable. The power cable must either be plugged into an approved three-contact electrical outlet or used with a three-contact to two-contact adapter with the grounding wire (green) firmly connected to an electrical ground (safety ground) at the power outlet. The power jack and mating plug of the power cable meet International Electrotechnical Commission (IEC) safety standards.

Do Not Operate In An Explosive Atmosphere

Do not operate the instrument in the presence of flammable gases or fumes. Operation of any electrical instrument in such an environment constitutes a definite safety hazard.

Keep Away From Live Circuits

Operating personnel must not remove instrument covers. Component replacement and internal adjustments must be made by qualified maintenance personnel. Do not replace components with the power cable connected. Under certain conditions, dangerous voltages may exist even with the power cable removed. To avoid injuries, always disconnect power and discharge circuits before touching them.

Do Not Service Or Adjust Alone

Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

Do Not Substitute Parts Or Modify Instrument

Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification of the instrument. Return the instrument to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.

Dangerous Procedure Warnings

Warnings, such as the example below, precede potentially dangerous procedures throughout this manual. Instructions contained in the warnings must be followed.

WARNING Dangerous voltages, capable of causing death, are present in this instrument. Use extreme caution when handling, testing, and adjusting.

Safety Symbols Used In Manuals

The following is a list of general definitions of safety symbols used on equipment or in manuals:

⚠	Instruction manual symbol: the product is marked with this symbol when it is necessary for the user to refer to the instruction manual in order to protect against damage to the instrument.
ϟ	Indicates dangerous voltage (terminals fed from the interior by voltage exceeding 1000 volts must be marked with this symbol).
⏚ OR ⏚	Protective conductor terminal. For protection against electrical shock in case of a fault. Used with field wiring terminals to indicate the terminal which must be connected to ground before operating the equipment.
⏚	Low-noise or noiseless, clean ground (earth) terminal. Used for a signal common, as well as providing protection against electrical shock in case of a fault. A terminal marked with this symbol must be connected to ground in the manner described in the installation (operating) manual before operating the equipment.
⏚ OR ⊥	Frame or chassis terminal. A connection to the frame (chassis) of the equipment which normally includes all exposed metal structures.
∿	Alternating current (power line).
⎓	Direct current (power line).
≂	Alternating or direct current (power line).

Caution	The Caution sign denotes a hazard. It calls your attention to an operating procedure, practice, condition, or similar situation, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product.
Warning	The Warning sign denotes a hazard. It calls your attention to a procedure, practice, condition or the like, which, if not correctly performed, could result in injury or death to personnel.

Certification and Warranty

Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

Warranty

This Hewlett-Packard system product is warranted against defects in materials and workmanship for a period of 90 days from date of installation. During the warranty period, HP will, at its option, either repair or replace products which prove to be defective.

Warranty service of this product will be performed at Buyer's facility at no charge within HP service travel areas. Outside HP service travel areas, warranty service will be performed at Buyer's facility only upon HP's prior agreement and Buyer shall pay HP's round trip travel expenses. In all other cases, products must be returned to a service facility designated by HP.

For products returned to HP for warranty service, Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country. HP warrants that its software and firmware designated by HP for use with an instrument will execute its programming instructions when properly installed on that instrument. HP does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environment specifications for the product, or improper site preparation or maintenance.

No other warranty is expressed or implied. HP specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.

Exclusive Remedies

The remedies provided herein are buyer's sole and exclusive remedies. HP shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.