HEWLETT
PACKARD

Graphical User Interface User's Guide

MC68360/68EN360 Emulator/Analyzer (HP 64780A)

Notice

Printing History

New editions are complete revisions of the manual. 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   B3091-97000, December 1993

Edition 2   B3091-97001, April 1996

Safety and Certification and Warranty

Safety information, and certification and warranty information can be found at the end of this manual on the pages before the back cover.

The HP 64780A Emulator

Description

The HP 64780A emulator supports the Motorola 68360 and 68EN360 microprocessor operating at clock speeds up to 25 MHz.

The emulator supports both 5V and 3.3V operation. The emulator plugs directly into a PGA socket, and it can be plugged into a PQFP target system using optional accessories.

The emulator plugs into the modular HP 64700 instrumentation card cage and offers 80 channels of processor bus analysis with the HP 64794A or HP 64704A emulation-bus analyzer. Up to eight megabytes of emulation memory may be installed on the probe. High performance download is achieved through the use of an optional LAN or RS-422 interface. A pair of RS-232 ports and a firmware-resident interface allow debugging of a target system at remote locations.

For software development, the HP AxCASE environment is available on SUN SPARCsystems and on HP workstations. This environment includes an ANSI standard C compiler, assembler/linker, a debugger, the HP Software Performance Analyzer that allows you to optimize your product software, and the HP Branch Validator for test suite verification. The C compiler, assembler/linker, and debugger are also available for MS-DOS systems.

Language support is also available from several third-party vendors. This capability is provided through the HP 64700's ability to consume several industry standard output file formats.

Ada language support is provided on HP 9000 workstations by third-party vendors such as Alsys and Verdix. An Ada application developer can use the HP emulator and any compiler that generates HP/MRI IEEE-695 to do exhaustive, real-time debugging in-circuit or out-of-circuit.

Features

HP 64780A Emulator

- 25 MHz active probe emulator
- 5V and 3.3V operation
- No wait states to target memory up to 25 MHz
- Fast termination cycles to target memory up to 25 MHz
- Unlimited software breakpoints
- Symbolic support
- 36 inch cable and 219 mm (8.8") x 102 mm (4") probe, terminating in PGA package
- Background and foreground monitors
- Simulated I/O with workstation interfaces
- Consumes IEEE-695, HP-OMF, Motorola S-Records, and Extended Tek Hex File formats directly. (Symbols are available with IEEE-695 and HP-OMF formats.)
- Multiprocessor emulation
  - synchronous start of 32 emulation sessions
  - cross triggerable from another emulator, logic analyzer, or oscilloscope
- Demo board and self test module included

Emulation-bus analyzer

- 80-channel emulation-bus analyzer
- Post-processed dequeued trace with symbols
- Eight events, each consisting of address, status, and data comparators
- Events may be sequenced eight levels deep and can be used for complex trigger qualification and selective store

Emulation memory

- Up to 8 Mbytes of emulation memory
- All emulation memory is dual-ported
- Mapping resolution is 256 bytes
- No wait states required for emulation memory for processor speeds up to 25 MHz
- Fast termination cycles to emulation memory supported up to 10 MHz

In This Book

This manual shows you how to use the HP 64780A emulator through its Graphical User Interface for the MC68360 microprocessor. It is divided into the following parts:

Part 1 contains the Quick Start Guide. It shows you how to quickly become productive with the emulation system.

Part 2 explains how to accomplish common tasks, often requiring use of several emulator/analyzer commands together. This part assumes you know how to use the commands to control the emulator. Instructions are given to help you connect the emulation probe into a target system, get the desired interface on screen, and use the emulator/analyzer commands to control the emulation processor while making emulation measurements. This part also shows you how to use the emulation-bus analyzer for debugging, use the Software Performance Measurement Tool supplied with the emulator, and couple two or more emulators to coordinate measurements involving more than one processor.

Part 3 shows you how to change the appearance or behavior of the Graphical User Interface, and describes in detail each of the commands available in the emulator/analyzer, and lists each of the messages you may see while using the MC68360 emulator/analyzer, along with suggested corrective actions.

Part 4 of this book shows you how to install the Graphical User Interface and Softkey Interface software, and how to update your emulator/analyzer firmware with the progflash command, and display current firmware version information.

The Hewlett-Packard *M68360 Emulator/Analyzer Installation/Service/Terminal Interface User's Guide* shows you how to install the emulator hardware into the card cage, install SRAM modules and covers on the emulation probe, and how to connect the probe to the demo board and verify performance of the emulation hardware. It also provides a thorough analysis of possible problems and their solutions. It also lists the complete specifications and characteristics of the M68360 emulator. It shows you how to connect the emulator into an MC68360 target system and overcome differences between the specifications and characteristics of the target microprocessor and those of the emulator.

Contents

Contents

Contents

12 Emulator Error Messages

Part 1

Quick Start Guide

Quick Start Guide

In This Part

This part describes how to quickly become productive with the emulation system.

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, on-line help, customizable action keys and pop-up recall buffers, etc.

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.

The Softkey Interface

Display area.

```
                                        cmdscript
 Memory   :mnemonic :file = main(module)."main.c":
   address    data
  00000FC2  4E560000    LINK.W    A6,#$0000
  00000FC6  2F0B        MOVE.L    A3,-(A7)
  00000FC8  2F0A        MOVE.L    A2,-(A7)
  00000FCA  247C000071  MOVEA.L   #$00007156,A2
  00000FD0  267C000071  MOVEA.L   #$0000717E,A3
  00000FD6  14BC0001    MOVE.B    #$01,(A2)
  00000FDA  4EB9000016  JSR       $00001650
  00000FE0  4EB900001E  JSR       $00001E24
  00000FE6  4213        CLR.B     (A3)
  00000FE8  4E71        NOP
  00000FEA  157C000100  MOVE.B    #$01,($0001,A2)
  00000FF0  16BC0001    MOVE.B    #$01,(A3)
  00000FF4  4EB9000017  JSR       $0000171A
  00000FFA  52B9000077  ADDQ.L    #1,$0000771C
  00001000  4879000077  PEA       $0000771C
  00001006  4EB9000010  JSR       $00001056

STATUS:   M68360--Emulation reset_____...R....


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

Status line.

Command line.

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 Graphical User Interface

Menu bar

Action keys

Entry buffer

Entry buffer recall
button.

Display area.

Scroll bar.

Status line.

Command line.

Command line entry
area.

Softkey pushbuttons

```
Hewlett Packard Emulator/Analyzer: hplsds2 (m68360)

File Display Modify Execution Breakpoints Trace Settings                          Help

Action keys:    < Demo >    Disp Src ( )    Trace ( )       Run       Step Source
 <Your Key>      Make      Disp Src Prev Run Xfer to ( )    Break       Step Asm
 REG KEYS:    Sync $MBAR    Set $MBAR     Boot Code    Help Reg ( )   ** Help 360 **
  Pick Util    Pick Reg 360 Pick Chip 360  Reg 360 All   Mod 360 ( )    Reg 360 ( )
  Run Util ( )   ^STEP #1^    Set Chip ( )  PRBD 360 All Mod Memory    PRBD 360 ( )

( ):  main                                                                   Recall

Memory  :mnemonic :file = main(module)."main.c":
   address  label         data
       91        extern void update_system();       /* update system variables */
       92        extern void interrupt_sim();        /* simulate an interrupt */
       93        extern void do_sort();              /* sets up ascii array and calls
       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:    M68360--Emulation reset                                        ◄ ►

_display memory main mnemonic



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

Command: Return Recall        Cursor: Backup Forward Clear to end Clear      Help
```

Command buttons. Includes command   Cursor buttons for command line area
recall button.   control.

Menu Bar. Provides pulldown 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 pulldown 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 pulldown 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** pulldown 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 pulldown 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 Description

Generic Button Name	Bindings: HP 9000	Sun SPARCsystem	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 pulldown 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 tables show the default keyboard bindings.

Keyboard Key Bindings

Generic Key Name	HP 9000	Sun SPARCsystem
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 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 64780 emulator and 80-channel 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 Chapter 14, "Installation."

- The emulator is operating out-of-circuit (that is, plugged into the demo board, not your target system).

- You have selected the appropriate clock module and installed it in the emulator probe according to the instructions in the MC68360 Emulator/Analyzer (HP 64780A) Installation/Service/Terminal Interface manual that was supplied with your emulator hardware.

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.

Depending of the version of the demo program you are using and of your compiler, line numbers and memory locations may not match those shown in this manual.

Step 1. Start the demo

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

1 Change to the demo directory.

```
$ cd /usr/hp64000/demo/debug_env/hp64780 <RETURN>
```

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 <RETURN>
```

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 <logical_emul_name> <RETURN>
```

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).
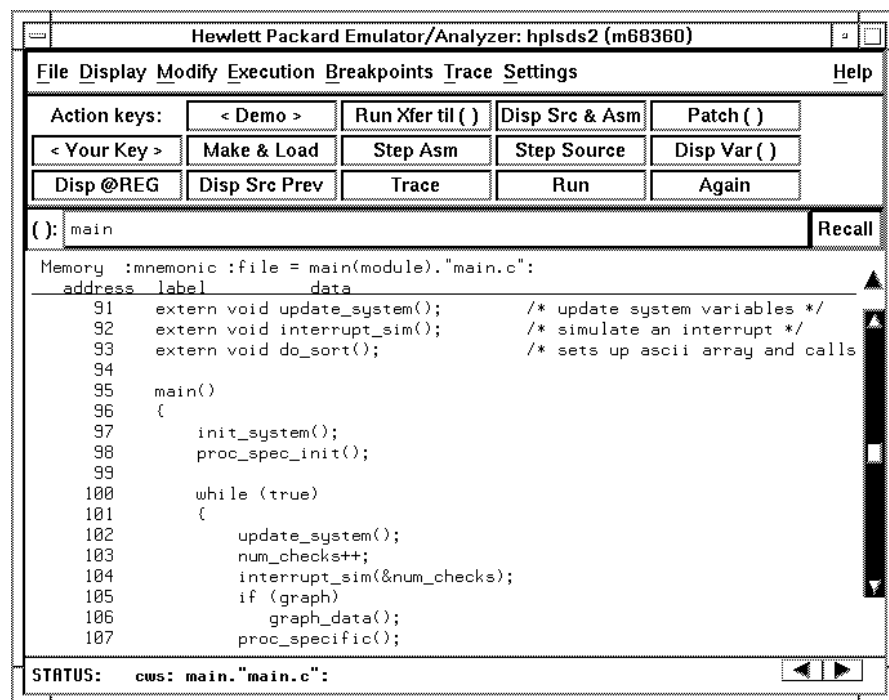
Step 2: Display the program in memory

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_** <RETURN>

```
┌──────────────────────────────────────────────────────────────────────────┐
│ ▬     Hewlett Packard Emulator/Analyzer: hplsds2 (m68360)         ▫ □    │
├──────────────────────────────────────────────────────────────────────────┤
│ 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    │
├──────────────────────────────────────────────────────────────────────────┤
│ ( ): main                                                        Recall   │
├──────────────────────────────────────────────────────────────────────────┤
│ Memory  :mnemonic :file = main(module)."main.c":                     ▲   │
│   address  label        data                                             │
│      91     extern void update_system();      /* update system variables */│
│      92     extern void interrupt_sim();      /* simulate an interrupt */ │
│      93     extern void do_sort();            /* sets up ascii array and calls│
│      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."main.c":                                    ◄ │ ►  │
└──────────────────────────────────────────────────────────────────────────┘
```

The default display mode settings cause source lines and symbols to appear in displays where appropriate. Notice 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 <RETURN>

```
Memory  :@sp :mnemonic :file = main(module)."main.c":
   address  label          data
      91      extern void update_system();      /* update system variables */
      92      extern void interrupt_sim();      /* simulate an interrupt */
      93      extern void do_sort();            /* sets up ascii array and calls
      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:   M68360--Running in monitor      Software break: 000000fc2@sp    ◄ ►
```

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 <RETURN>

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.

```
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;
```

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 <RETURN>

  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.

```
┌──────────────────────────────────────────────────────────────────────┐
│═    Hewlett Packard Emulator/Analyzer: hplsds2 (m68360)      ▫ □      │
├──────────────────────────────────────────────────────────────────────┤
│ 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    │ │
│ └──────────────────────────────────────────────────────────────────┘ │
│ ( ): main                                                   Recall    │
│ Memory  :@sp :mnemonic :file = main(module)."main.c":                │
│   address  label        data                                          │
│       91    extern void update_system();     /* update system variables */ │
│       92    extern void interrupt_sim();     /* simulate an interrupt */ │
│       93    extern void do_sort();           /* sets up ascii array and calls │
│       94                                                               │
│       95    main()                                                    │
│       96    {                                                         │
│       97        init_system();                                        │
│       98        proc_spec_init();  ┌─────────────────────────────┐    │
│       99                           │ Choose Action for Highlighted Line│ │
│      100        while (true)       ├─────────────────────────────┤    │
│      101        {                  │ Set/Clear Software Breakpoint │    │
│      102            update_system( │ Edit Source                 │    │
│      103            num_checks++;  ├─────────────────────────────┤    │
│      104            interrupt_sim( │ Run Until                   │    │
│      105            if (graph)     ├─────────────────────────────┤    │
│      106                graph_data( │ Trace After                │    │
│      107            proc_specific( │ Trace Before               │    │
│ STATUS:   cws: main."main.c":      │ Trace About                │◄ ►│ │
│                                    │ Trace Until                │    │
└──────────────────────────────────────────────────────────────────────┘
```

Or, using the command line, enter:

run until main."main.c": line 98 <RETURN>

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 Click on the **Disp Var ()** action key.

Or, using the command line, enter:

display data , num_checks ***int32*** <RETURN>

```
Data  :update
  address  label         type         data
  0000771C  |_num_checks  int32                 0
```

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 <RETURN>

```
Registers

Next PC 00000FE0@sp
 PC    00000FE0   STATUS 2704 < s  z  >         USP 56B54C44      SSP 00012F8C
 D0-D7 00000020 00000020 000080A8 0000071E 1B0076FE AA304483 1B2397C8 FA023074
 A0-A7 00007180 0000774C 00007156 0000717E 00007868 0000F156 00012F94 00012F8C
 VBR   00000000 SFC    0 DFC    0
```

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 <RETURN>

```
Registers

Next PC 00000FE0@sp
 PC    00000FE0   STATUS 2704 < s  z  >         USP 56B54C44     SSP 00012F8C
 D0-D7 00000020 00000020 000080A8 0000071E 1B0076FE AA304483 1B2397C8 FA023074
 A0-A7 00007180 0000774C 00007156 0000717E 00007868 0000F156 00012F94 00012F8C
 VBR   00000000 SFC    0 DFC    0

Step_PC 00000FE0@sp  JSR       p.proc_spec_init
Next PC 00001E24@sp
 PC    00001E24   STATUS 2704 < s  z  >         USP 56B54C44     SSP 00012F88
 D0-D7 00000020 00000020 000080A8 0000071E 1B0076FE AA304483 1B2397C8 FA023074
 A0-A7 00007180 0000774C 00007156 0000717E 00007868 0000F156 00012F94 00012F88
 VBR   00000000 SFC    0 DFC    0
```

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 <RETURN>

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 <RETURN>

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 <RETURN>

```
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
        ##########main.c - line       1 thru    96 ################################
        extern void interrupt_sim();          /* simulate an interrupt */
        extern void do_sort();                /* sets up ascii array and calls combs

        main()
        {
after     prog|main.main    LINK.W    A6,#$0000                   -----------
+001    =pr|main+00000004  MOVE.L    A3,-(A7)                           120      nS
+002     sysstac+00007F94   $00012FF0     supr data long wr (ds32)      240      nS
+003     pr|main+00000006  MOVE.L    A2,-(A7)                           160      nS
        =pr|main+00000008  MOVEA.L   #$00007156,A2
+004     sysstac+00007F90   $000080A0     supr data long wr (ds32)      200      nS
+005     pr|main+0000000A   $00007156     supr prgm long rd (ds32)      160      nS
+006     sysstac+00007F8C   $000077A8     supr data long wr (ds32)      160      nS
+007     pr|main+0000000E  MOVEA.L   #$0000717E,A3                      160      nS
        ##########main.c - line      97 ################################
```

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: Display memory at an address in a register

1 Click on the **Disp @REG** action key.

Or, using the command line, enter the name of the command file:

```
mematreg <RETURN>
```

A command file dialog box appears (or a prompt appears in the command line).

2 Move the mouse pointer to the dialog box text entry area, type "A7", and click on the "OK" button.

Or, if the prompt is in the command line:

```
A7  <RETURN>
```

```
Memory  :@sp :bytes :blocked :update
  address      data     :hex                                    :ascii
  00012E88-8F    00  00  11  1C  00  00  77  2C    . . . . .  . . w ,
  00012E90-97    00  00  71  56  00  01  2F  2C    . . q V  . . / ,
  00012E98-9F    00  00  15  74  00  00  73  33    . . . t  . . s 3
  00012EA0-A7    00  01  2E  CF  00  00  00  80    . . . . .  . . .
  00012EA8-AF    00  00  08  CA  00  00  08  CA    . . . . .  . . .
  00012EB0-B7    AB  30  44  83  3B  23  97  C8    . 0 D .  ; # . .
  00012EB8-BF    FA  00  30  64  00  00  71  56    . . 0 d  . . q V
  00012EC0-C7    00  00  73  04  00  00  78  68    . . s .  . . x h
  00012EC8-CF    4C  65  6E  20  00  00  00  00    L e n    . . . .
  00012ED0-D7    00  01  2E  F0  00  00  00  00    . . . . .  . . .
  00012ED8-DF    00  00  71  80  00  00  00  00    . . q .  . . . .
  00012EE0-E7    00  00  00  00  00  00  00  3C    . . . . .  . . <
  00012EE8-EF    00  00  00  11  00  00  1E  A2    . . . . .  . . .
  00012EF0-F7    00  00  00  02  00  00  00  00    . . . . .  . . .
  00012EF8-FF    00  00  08  CA  00  00  00  04    . . . . .  . . .
  00012F00-07    3B  23  97  C8  FA  00  30  64    ; # . .  . . 0 d
  00012F08-0F    00  00  00  01  00  01  2F  5C    . . . . .  . . / \
```

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 <RETURN>

Solving Problems

If you encounter problems when using the emulator/analyzer, refer to the chapter
titled "Solving Problems" in the MC68360 Emulator/Analyzer
Installation/Service/Terminal Interface Manual.

Part 2

Using The Emulator

Making Measurements

When you've become familiar with the basic emulation process, you'll want to make specific measurements to analyze your software and target system. The emulator has many features that allow you to control program execution, view processor resources, and program activity.

In This Part 2

Chapter 2, "Plugging into a Target System," tells you how to correctly connect the emulation probe into a target system.

Chapter 3, "Starting and exiting HP 64700 Interfaces," tells you how to get the desired interface on screen.

Chapter 4, "Entering Commands," tells you how to use the commands and features of the Graphical User Interface and Softkey Interface.

Chapter 5, "Configuring the Emulator," explains how to use the emulator/analyzer commands to allocate emulation resources such as memory and how to enable and disable certain emulator features.

Chapter 6, "Using the Emulator," shows you how to use the emulator/analyzer commands to control the emulation processor and make simple emulation measurements.

Chapter 7, "Using the Emulation-Bus Analyzer," explains how to use the emulation-bus analyzer to record program execution for debugging.

Chapter 8, "Making Software Performance Measurements," shows you how to use the Software Performance Measurement Tool supplied with the emulator.

Chapter 9, "Making Coordinated Measurements," tells how to couple two or more emulators to coordinate measurements involving more than one processor.

This part of the manual explains how to accomplish various common tasks, often requiring use of several emulator/analyzer commands together. It assumes you know how to use the commands to control the emulator. If you need a general introduction to using the emulator, refer to Part 1.

2

Plugging into a Target System

Plugging the Emulator into a Target System

This chapter describes the tasks you must perform when plugging the emulator into a target system. These tasks are grouped into the following sections:

- Connecting the emulator to the target system.

- Plugging into the Motorola QUADS target system.

Before attempting to run the emulator, ensure that you have selected the proper clock module and installed it in the emulator probe. The details of clock module selection are discussed in the MC68360 Emulator/Analyzer (HP 64789A) Installation/Service/Terminal Interface manual.

Connecting the Emulator to the Target System

The 68360 emulator probe plugs into a PGA through-hole socket that is soldered into the target system. There are three ways of connecting the 68360 emulator probe to a target system:

- Plug it into the PGA socket directly.

- Plug it into the PGA socket via flexible cable.

- Plug in a PQFP adapter socket into the PGA socket in place of the emulator, and then connect the emulator probe to the PQFP adapter.
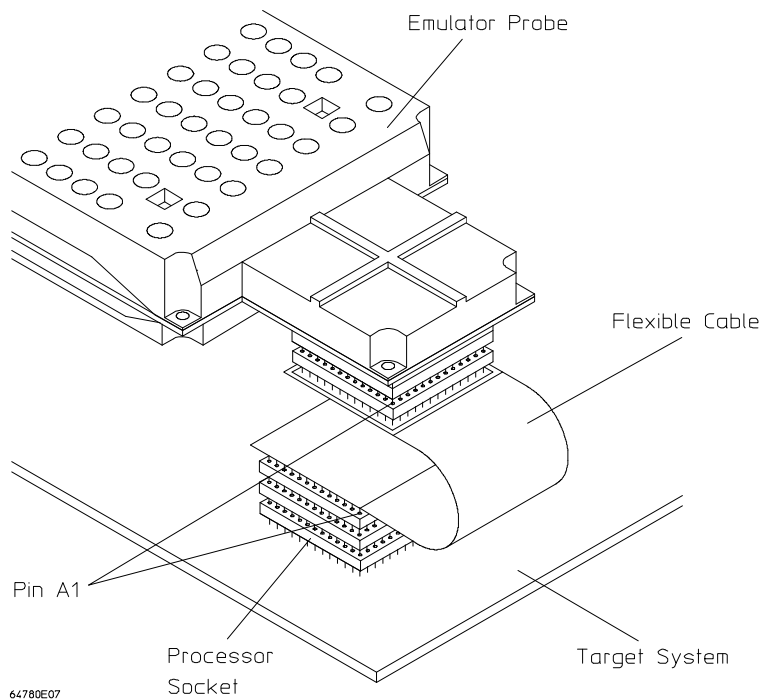
Emulator Probe

Flexible Cable

Pin A1

Processor
Socket

Target System

64780E07

If using a PQFP adapter, the emulator will tristate your target 68360 and use the emulator's 68360 to run your target system.

This section describes the steps you must perform when connecting the emulator to a target system:

1 Turn OFF power.

2 Plug the emulator probe into the target system.

3 Turn ON power.

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.

Step 1. Turn OFF power

CAUTION   **Possible Damage to the Emulator.** Make sure target system power is OFF and make sure HP 64700 power is OFF before removing or installing the emulator probe into the target system.

Do not turn HP 64700 power OFF while the emulator is plugged into a target system whose power is ON.

1 If the emulator is currently plugged into a different target system, turn that target system's power OFF.

2 Turn emulator power OFF.

Step 2. Connect the probe to the target system

CAUTION
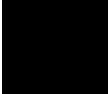
Possible Damage to the Emulator Probe. A pin extender is included with the emulator probe. **Do not use the probe without a pin extender installed**. Replacing a broken pin extender is much less expensive than replacing other pieces.

The use of more than one pin extender is discouraged, unless it is necessary for mechanical clearance reasons, because pin extenders cause signal quality degradation.

1 Install the emulator probe into the target system socket. Make sure that pin 1 of the connector aligns with pin 1 of the socket. **Damage to the emulator will result if the probe adapter is incorrectly installed.**

EMULATOR PROBE

MALE-TO-MALE HEADER

PGA SOCKET (PIN PROTECTOR)

PIN A1

DEMO BOARD

64780E19
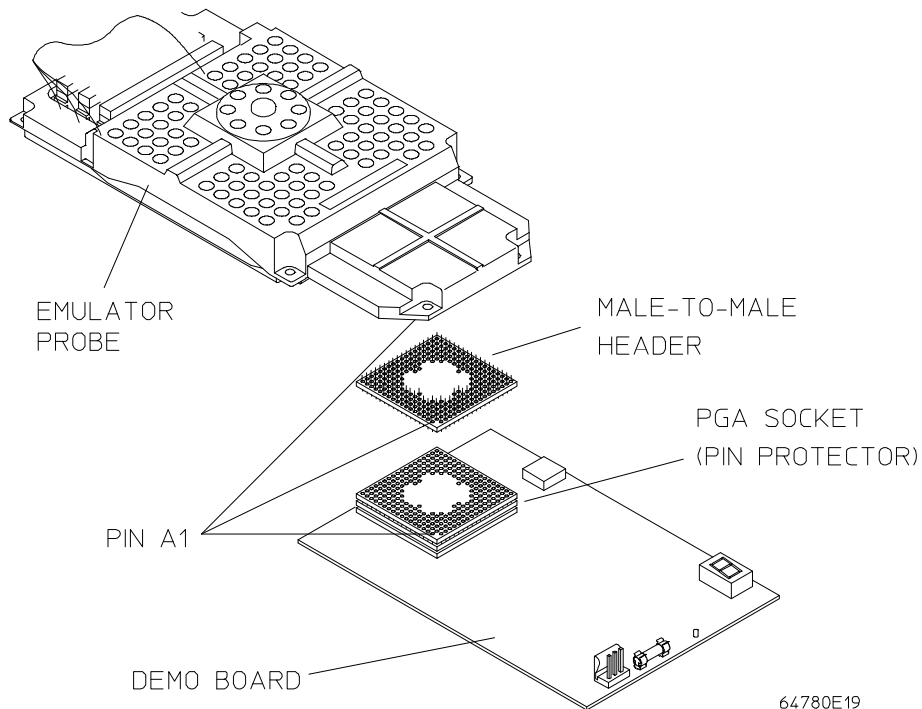
Step 3. Turn ON power

1 Turn emulator power ON.

2 Turn target system power ON.

Plugging into the Motorola QUADS Target System

This section shows you how to:

- Connect the emulator to the Motorola QUADS board.

The Motorola QUADS board gives you an opportunity to plug the emulator into a target system that contains one 68360 master chip, and one 68360 slave chip.

To connect the emulator to the Motorola QUADS

1 Turn OFF power.

If the emulator is currently plugged into a different target system, turn that target system's power OFF. Then, turn emulator power OFF.

2 Plug the emulator probe into the Motorola Quads Board.

The emulator contains devices that are susceptible to damage by static discharge. Take precautionary measures before handling the emulator probe to avoid damage.

The Motorola Quads board contains two 68360 processors: one running in Master mode, and the other running in Slave mode. The emulator must be plugged into the master processor.

Make sure pin 1 of the Quads board microprocessor socket and pin 1 of the emulator probe are properly aligned before inserting the probe into the socket. Otherwise, you may damage the emulator circuitry. Three or four pin protectors will be required to lift the probe above other hardware on the Quads board.

Ensure that the default clock module is plugged in. The default clock module must be installed in the emulator probe in order for the 68360 emulator to work with the Quads board. Refer to the Hewlett-Packard *MC68360 Emulator/Analyzer Installation/Service/Terminal Interface User's Guide* for details.

MALE-TO-MALE
HEADER

PGA PIN
PROTECTOR
(5181-0206)

MOTOROLA
QUADS BOARD

PIN A1

MASTER PROCESSOR

SLAVE PROCESSOR

64780E18

PGA PIN
PROTECTORS
(5181-0206)

MASTER PROCESSOR

64780E15

SLAVE PROCESSOR
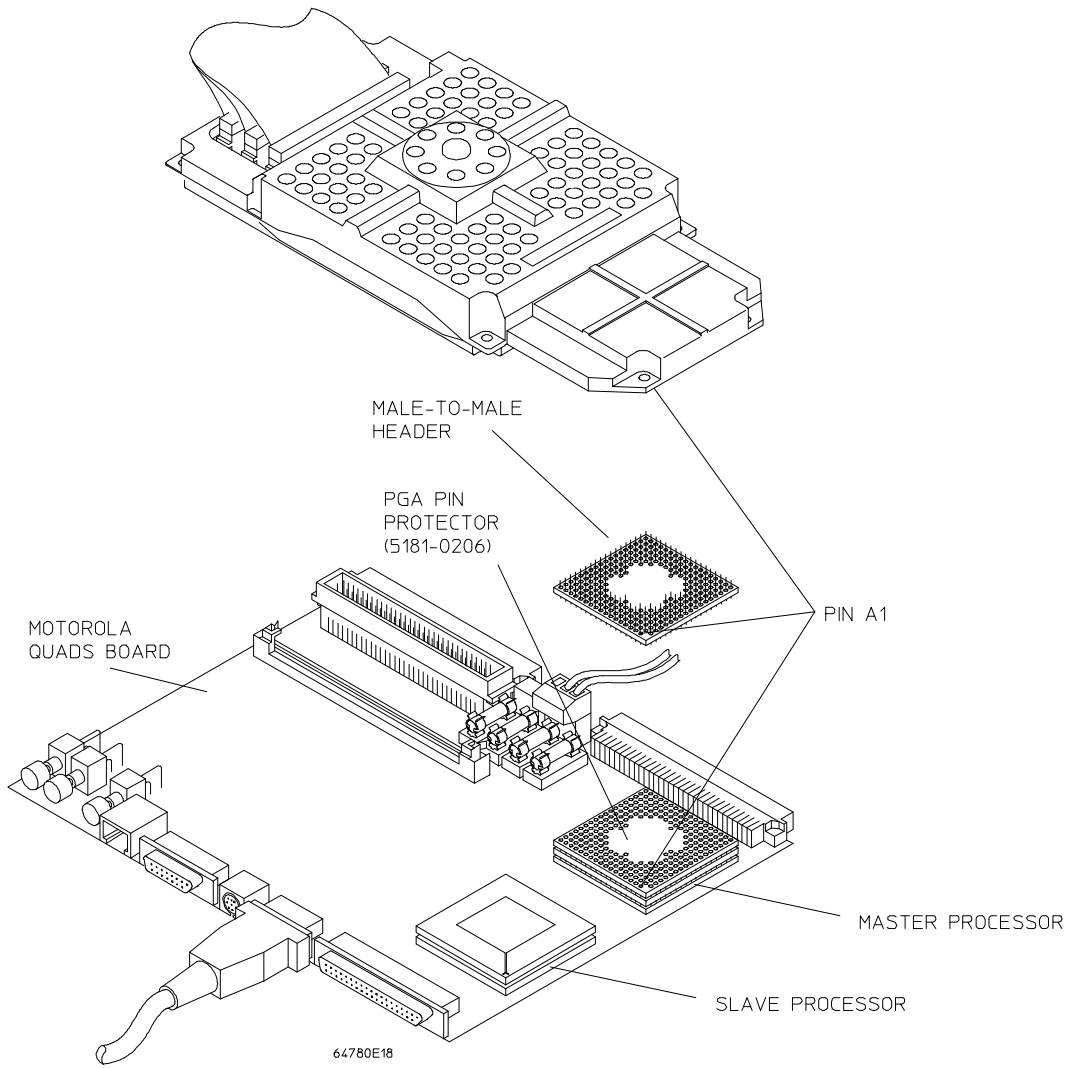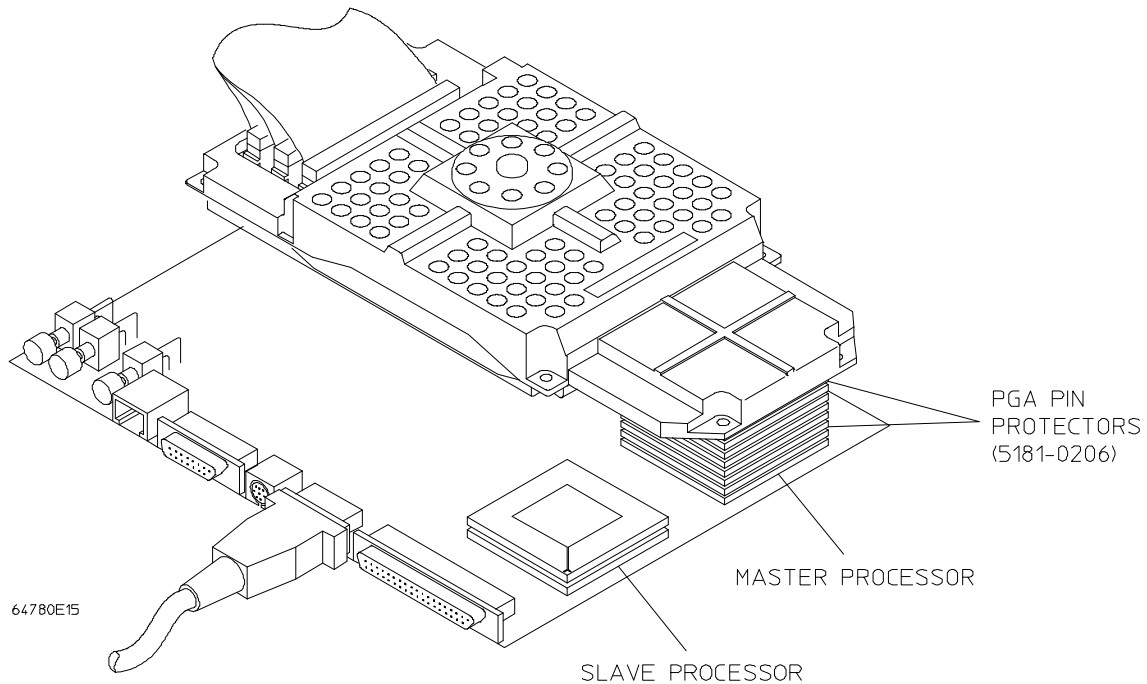
3 Turn ON power. First turn on the emulator power. Then turn target system power
ON.

4 You will need to select three configuration items for the emulator. With the
Graphical User Interface on screen, choose **Modify→Emulator Config...** In the
Emulator Configuration dialog box, select Emulator Pod Settings. In the Pod
Settings dialog box, select the following:

- "Yes" for Buffer AS, DS and R/W.

- "Off" for Clock O1 Drive to Target.

- "8 bits" for Memory Size for Chip Select.

5 Choose **Execution→Run→from Reset**.   Verify that the Quads board boots up
and runs normally.

6 Verify that the master registers display normally:

   1   Press the "Sync $MBAR" Action Key. The Browser Window should show that HP64MBAR360 is synchronized to 20000H.

   2   Press the "Pick Reg 360" Action Key.

   3   Select pepar from the Register List in the Browser Window and click Done.

   4   Press the "Reg 360 ()" Action Key.

       A Browser Window opens. It shows the contents of the master processor's pepar registers. Press Done when finished reviewing the register contents.

       To view another register in the master register set, simply repeat steps 3 and 4 above.

7 Verify that the slave registers display normally:

   1   Press the "Pick Util" Action Key.

   2   Select "assign68360chip" from the Utilities Selection Browser Window and then press Done.

   3   Press the "Run Util ()" Action Key to run the selected utility.

   4   In the Define command file parameter dialog box, enter 1 in the entry field and click OK. This identifies the slave chip as chip 1.

   5   In the next Define command file parameter dialog box, enter 22000H and click OK. This identifies the address of the slave processor.

   6   Click Done in the New Slave Addresses Browser Window when you have seen the content that identifies the addresses of the master and slave processors.

       Note that steps 1 through 6 of this procedure only have to be done to set up the emulator for slave registers displays. Once these steps have been done, you can view any of the registers in the slave chip. To switch between displaying of master and slave registers, you can simply start with the next step (Step 7) to see the slave registers or press "Sync $MBAR" (in the previous procedure) to pick the master registers.

   7   Press the "Pick Chip 360" Action Key.

   8   From the Available MC68360 Slaves Browser Window, select HP64MBAR360_1 and click Done.

   9   Press the "Set Chip ()" Action Key. This identifies the chip that will be controlled and viewed in the interface.

  10   The Current M68360 Browser Window should show that HP64MBAR360 is set to 22000H.

  11   Press the "Pick Reg 360" Action Key.

12 Select pepar from the Register List in the Browser Window and click Done.

13 Press the "Reg 360 ()" Action Key.

A Browser Window opens. It shows the contents of the slave processor's pepar registers. Press Done when finished reviewing the register contents.

To view another register in the slave register set, simply repeat steps 12 and 13 above.

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 C debugger interface (which is a separate product) lets you view the stack backtrace and high-level data structures, and it lets you use C language expressions and macros. These features are most useful when debugging 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 C debugger interface window and one SPA window are 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 Chapter 14, "Installation".

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 Chapter 14, "Installation"), 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).

  If you are running a window system on your host computer (for example, the X Window System), you can run the interface in up to 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 68360 emulator:

$ **emul700** em68360 <RETURN>

The "em68360" 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:      em68360    m68360      21.17.9.143
serial:      em68360    m68360      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 Chapter 12, "Emulator Error
Messages".

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 says to use 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 to the **emul700** command.

Refer to the "Using Command Files" section in Chapter 4, "Entering Commands" for information on creating command files.

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

```
$ emul700 -c startup em68360 <RETURN>
```

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 "em68360":

```
$ emul700 -lv em68360 <RETURN>
```

The information may be similar to:

```
em68360 - m68360 running; user = guest
    description:      M68360 emulation, 512K bytes emul mem
    user interfaces:  xdebug, xemul, xperf, skemul, sktiming
    device channel:   /dev/emcom23
```

Or, the information may be similar to:

```
em68360 - m68360 running; user = guest@myhost
    description:     M68360 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 to 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 "em68360":

```
$ emul700 -U em68360 <RETURN>
```

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 and the high-level debugger interface).

This section shows you how to:

- Open additional emulator/analyzer interface windows.

- Open the high-level debugger interface window.

- Open the software performance analyzer (SPA) interface window.

To open additional emulator/analyzer windows

- To open additional Graphical User Interface windows, choose **File→Emul700→Emulator/Analyzer** *under Graphic Windows*, or enter the **emul700 <emul_name>** command in another terminal emulation window.

- To open additional Softkey Interface windows, choose **File→Emul700→Emulator/Analyzer** *under Terminal Windows*, or enter the **emul700 -u skemul <emul_name>** command in another terminal emulation window.

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 in which the interface is running. When you enter commands in different windows, the command entered in the first

45

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 high-level debugger interface window

- Choose **File→Emul700→High-Level Debugger ...** under "Graphic Windows", or
  enter the **emul700 -u xdebug <emul_name>** command in another terminal
  emulation window.

For information on how to use the high-level debugger interface, refer to the
debugger/emulator *User's Guide*.

To open the software performance analyzer (SPA) interface window

- Choose **File→Emul700→Performance Analyzer ...** under "Graphic Windows",
  or enter the **emul700 -u xperf <emul_name>** command in another terminal
  emulation window.

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, in the emulator/analyzer interface command line, enter the **end** command with no options.

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, in the emulator/analyzer interface command line, enter the **end locked** command.

- To exit the interface and release the emulator for access by other users, choose **File→Exit→Released**. Or, in the emulator/analyzer interface command line, enter the **end release_system** command.

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 save the current configuration to a configuration file (if the configuration has changed), or the changes will be lost.

4
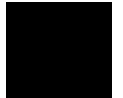
Entering Commands

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, on-line 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 pulldown menu item using the mouse.

- Choose a pulldown 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.
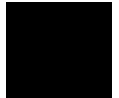
To choose a pulldown menu item using the mouse (method 1)

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 pulldown 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 pulldown menu item using the keyboard

- To initially display a pulldown menu, press and hold the **menu select** key (for example, the "Extend char" key on a 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 lower case only.)

- To move right to another pulldown menu after having initially displayed a menu, press the **right-arrow** key.

- To move left to another pulldown 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 Chapter 10, "Setting X Resources", 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.

The following pop-up menus are available in the Graphical User Interface:

- Mnemonic Memory Display.

- Breakpoints Display.

- Global Symbols Display.

- Local Symbols Display.

- Status Line.

- Command Line.
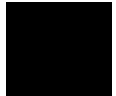
To place values into the entry buffer using the keyboard

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.

Note

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.
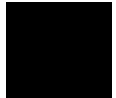
Example   To paste the symbol "num_checks" into the entry buffer from the interface display area, position the mouse pointer over the symbol and then click the paste mouse button.
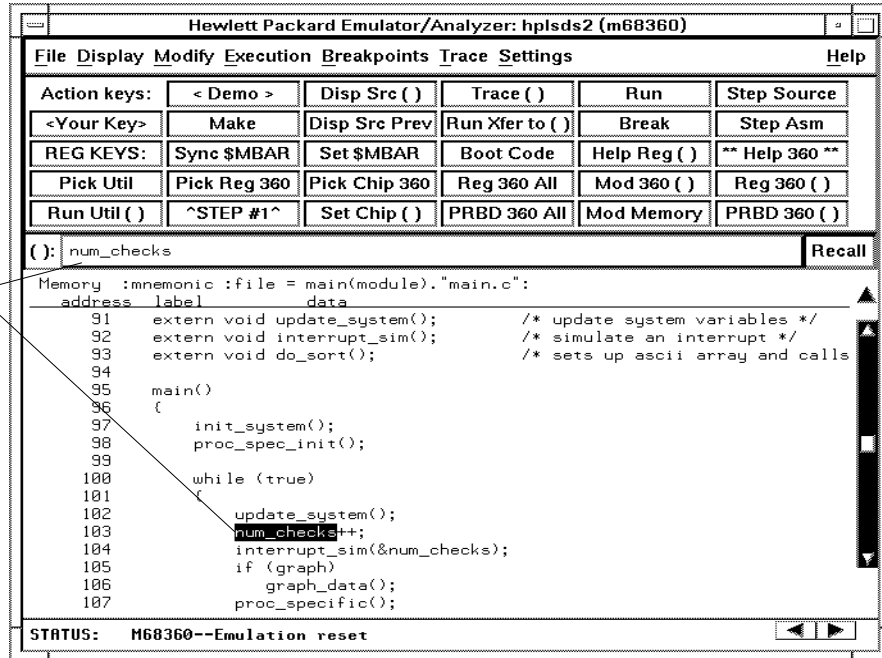
A mouse click causes the interface to expand the highlight to include the symbol "num_checks" and paste the symbol into the entry buffer.

```
┌────────────────────────────────────────────────────────────────────────────┐
│  ───         Hewlett Packard Emulator/Analyzer: hplsds2 (m68360)    □  □│   │
│ ┌──────────────────────────────────────────────────────────────────────────┐│
│ │ File  Display  Modify  Execution  Breakpoints  Trace  Settings       Help ││
│ │ ┌───────────┬───────────┬───────────┬──────────────┬──────────┬──────────┐│
│ │ │Action keys:│ < Demo >  │ Disp Src ( )│  Trace ( )  │   Run    │Step Source││
│ │ ├───────────┼───────────┼───────────┼──────────────┼──────────┼──────────┤│
│ │ │ <Your Key> │   Make    │Disp Src Prev│Run Xfer to ( )│  Break   │ Step Asm ││
│ │ ├───────────┼───────────┼───────────┼──────────────┼──────────┼──────────┤│
│ │ │ REG KEYS:  │Sync $MBAR │ Set $MBAR │  Boot Code   │Help Reg ( )│** Help 360 **││
│ │ ├───────────┼───────────┼───────────┼──────────────┼──────────┼──────────┤│
│ │ │ Pick Util  │Pick Reg 360│Pick Chip 360│ Reg 360 All │Mod 360 ( )│Reg 360 ( )││
│ │ ├───────────┼───────────┼───────────┼──────────────┼──────────┼──────────┤│
│ │ │Run Util ( )│ ^STEP #1^ │ Set Chip ( )│PRBD 360 All│Mod Memory│PRBD 360 ( )││
│ │ └───────────┴───────────┴───────────┴──────────────┴──────────┴──────────┘│
│ │ ┌─────┐                                                        ┌────────┐ ││
│ │ │( ): │ num_checks                                             │ Recall │ ││
│ │ └─────┘                                                        └────────┘ ││
│ │ Memory  :mnemonic :file = main(module)."main.c":                    ▲     ││
│ │   address  label         data                                             ││
│ │       91     extern void update_system();      /* update system variables */│
│ │       92     extern void interrupt_sim();      /* simulate an interrupt */ ││
│ │       93     extern void do_sort();            /* sets up ascii array and calls│
│ │       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:    M68360--Emulation reset                         │  ◄  │  ►    ││
│ │ └──────────────────────────────────────────────────────────┴─────────────┘│
│ └──────────────────────────────────────────────────────────────────────────┘│
└────────────────────────────────────────────────────────────────────────────┘
```

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 Chapter 10, "Setting X Resources").

  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 that by:

- Copying the text from the display area using the copy-and-paste feature.

- Enter the text directly by typing it into the entry buffer text area.

- Choose 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 Chapter 10, "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:

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, you can select an existing file name or specify a new file name.

Entry Buffer Recall   You can 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   You can 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 Chapter 10, "Setting X Resources").

Examples   To use the File Selection dialog box:
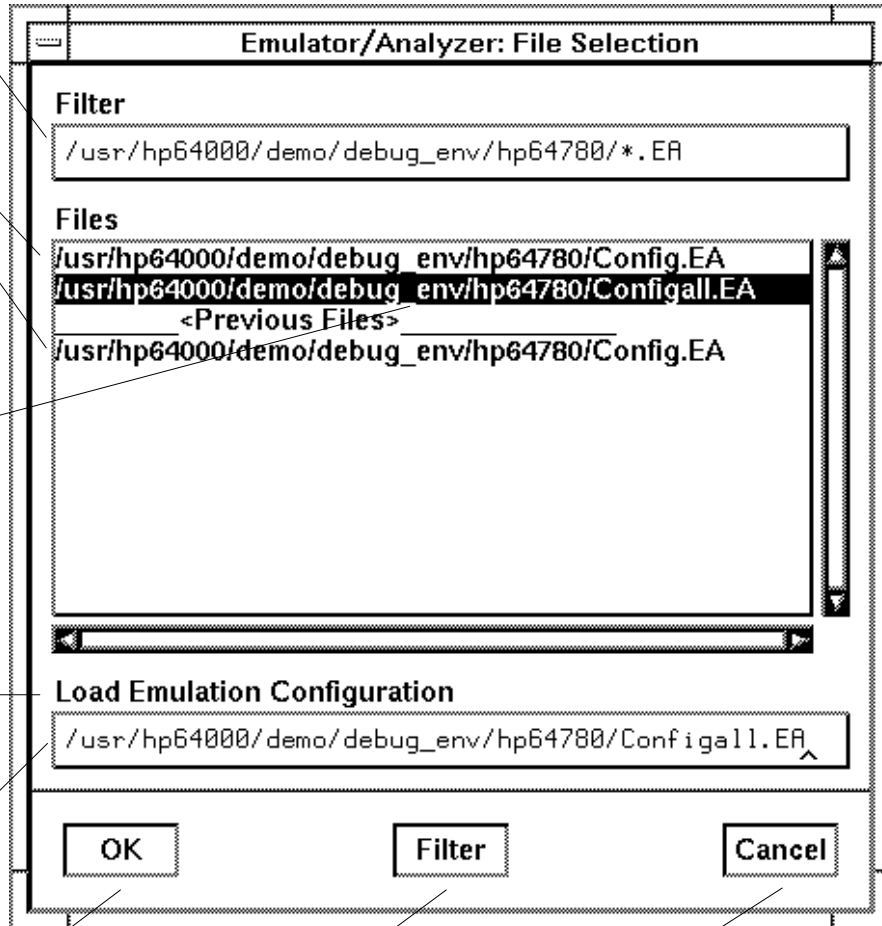
The file filter selects
specific files.

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.

Emulator/Analyzer: File Selection

Filter

/usr/hp64000/demo/debug_env/hp64780/*.EA

Files

/usr/hp64000/demo/debug_env/hp64780/Config.EA
/usr/hp64000/demo/debug_env/hp64780/Configall.EA
                    <Previous Files>
/usr/hp64000/demo/debug_env/hp64780/Config.EA

Load Emulation Configuration

/usr/hp64000/demo/debug_env/hp64780/Configall.EA

| OK | Filter | Cancel |

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
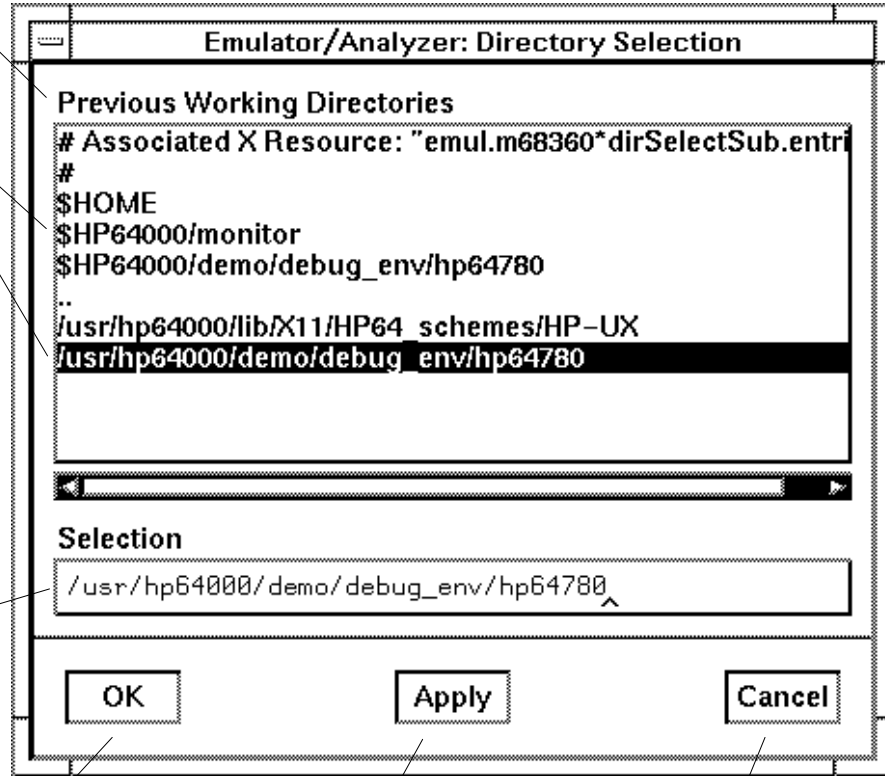operation and closes the
dialog box.

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

```
# Associated X Resource: "emul.m68360*dirSelectSub.entri
#
$HOME
$HP64000/monitor
$HP64000/demo/debug_env/hp64780
..
/usr/hp64000/lib/X11/HP64_schemes/HP-UX
/usr/hp64000/demo/debug_env/hp64780
```

Selection

/usr/hp64000/demo/debug_env/hp64780

| OK | Apply | Cancel |

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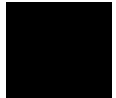
Clicking this button cancels the directory selection operation and closes the dialog box.

To access help information

- Help available in the help index:

    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.

- Help available for use of the Action Keys:

    - General information about using Action Keys in the 68360 emulator is
      available by pressing the "Help 360" Action Key.

    - Detailed information for configuring a particular SIM60 or CPM register can
      be obtained by placing the name of the register in the entry field and pressing
      the "Help Reg ()" Action Key.

    - Help for understanding how action keys work in the Graphical User Interface
      is available in Chapter 10, "Setting X Resources."

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.

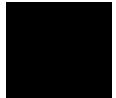- Display the help window.

To turn the command line on or off

- To turn the command line on or off using the pulldown 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 pulldown 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 pulldown 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 pulldown 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 halfbright and unresponsive when the cursor reaches the end of the command string. Similarly, the **Backup** pushbutton becomes halfbright 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 — pulldown 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 Chapter 10, "Setting X Resources").

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 on-line 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 <RETURN>

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 <RETURN>
```

To edit commands

- Use the <Left arrow>, <Right arrow>, <Tab>, <Shift><Tab>, <Insert char>, <Back space>, <Delete char>, <Clear line>, and <CTRL>u keys.

The <Left arrow> and <Right arrow> keys move the cursor single spaces to the left or right.

The <Tab> and <Shift><Tab> keys move the cursor to the next or previous word on the command line.

The <Insert char> key enters the insert editing mode and allows characters or command options to be inserted at the cursor location.

The <Back space> key deletes the character to the left of the cursor.

The <Delete char> key deletes the character to the right of the cursor.

The <Clear line> key deletes the characters from the cursor to the end of the line.

The <CTRL>u key erases the command line.

To access on-line help information

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

To access the command line's on-line help information, type either **help** or **?** on the command line. You will notice a new set of softkeys. By pressing one of these softkeys and <RETURN>, you can display information on that topic.

Examples   To display information on the system commands:

help ***system_commands*** <RETURN>

Or:

? ***system_commands*** <RETURN>

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 (**&**),
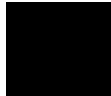which are the variables of the command file.

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
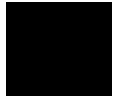
To start logging commands to a command file

- 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**.

- Using the command line, enter the **log_commands off** command.

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.

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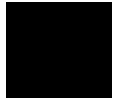
Issuing certain communications-related commands can prevent the interface from communicating with the emulator and cause abnormal termination of the interface.

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 high-level debugger and 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>" <RETURN>

This section shows you how to:

• Forward commands to the high-level debugger.

• Forward commands to the software performance analyzer.

To forward commands to the high-level debugger

• Enter the **forward debug "<command string>"** command using the command line.

Examples   To send the "Program Run" command to the debugger:

forward debug "Program Run" <RETURN>

Or, since only the capitalized key is required:

forward debug "P R" <RETURN>

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" <RETURN>

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.

- Verifying the emulator configuration.

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

The interactive measurement configuration options are further described in Chapter 9, "Making Coordinated Measurements".

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.

- Store configuration changes to a file.

- Change the configuration directory context.

- Display the configuration context.

- Access help topics.

- Access help for a configuration item in a dialog box.

- Exit the configuration interface.

- Load an existing configuration file.
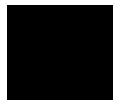
To start the configuration interface

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

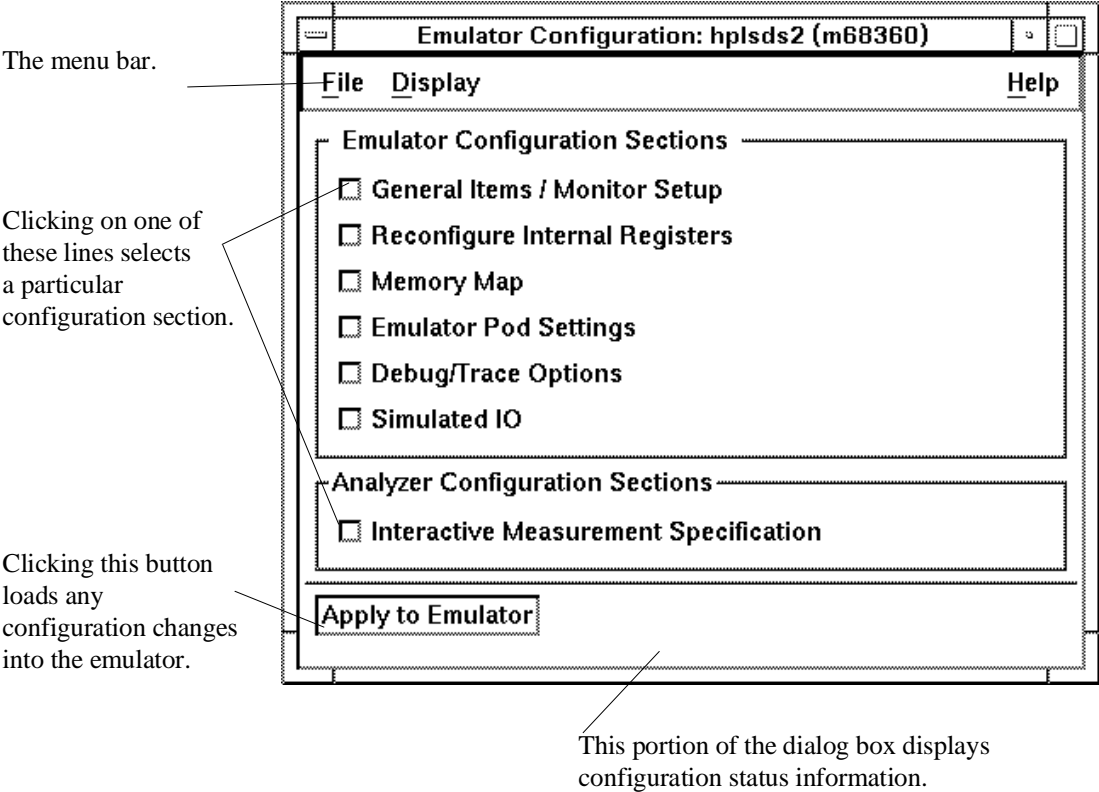- Using the command line, enter the **modify configuration** command.

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

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

  If you are 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 68360 emulator configuration interface top-level dialog box is shown below.

The menu bar.

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

Clicking this button
loads any
configuration changes
into the emulator.

Emulator Configuration: hplsds2 (m68360)

File   Display   Help

Emulator Configuration Sections

☐ General Items / Monitor Setup

☐ Reconfigure Internal Registers

☐ Memory Map

☐ Emulator Pod Settings

☐ Debug/Trace Options

☐ Simulated IO

Analyzer Configuration Sections

☐ Interactive Measurement Specification

Apply to 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.

If you are 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 "Emulator Pod Settings" section, answer "yes" to the "Modify emulator pod configuration?" question.

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

Most configuration sections provide dialog boxes similar to the following.

The dialog for this section has been opened

Applies configuration changes to the emulator

Configuration options in this section

Closes the dialog box

Cancels all changes since last "OK", "Apply to Emulator", or store to file.

Presents emulator configuration help topic browser.

Emulator Configuration: hplsds2 (m68360)

File Display   Help

Emulator Configuration Sections

☐ General Items / Monitor Setup

☐ Reconfigure Internal Registers

☐ Memory Map

☐ Emulator Pod Settings

▣ Debug/Trace Options

☐ Simulated IO

Analyzer Configuration Sections

☐ Interactive Measurement Specification

Apply to Emulator

Emulator Configuration: Debug Options

Debug/Trace Options

Break on Write to ROM   ◆ Yes ◇ No

Break on Memory Violation ◆ Yes ◇ No

Trace Mode (Type of Cycles) Foreground ▭

OK   Cancel   Help

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).

To apply configuration changes to the emulator

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

This 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 to a file, 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 pulldown 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.
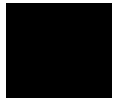
To store configuration changes to a file

- Choose **File→Store...** from the pulldown menu in the top-level configuration interface window, and use the file selection dialog box to name the configuration file.

- If you are 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 User 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 Chapter 4, "Entering Commands".
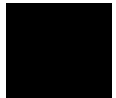
To change the configuration directory context

• Choose **File→Directory...** from the pulldown 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 Chapter 4, "Entering Commands".

To display the configuration context

• Choose **Display→Context...** from the pulldown 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.
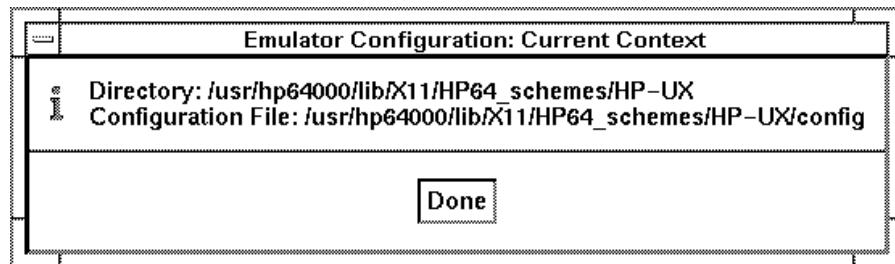
To access help topics

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

To access help for a configuration item in a dialog box

- Place the mouse pointer on the line of interest and press the f1 key.

- Choose **Help→On Item...** from the pulldown menu in the top-level configuration interface window. The mouse pointer changes from an arrow to a question mark. Place the question mark over a selection button or in the entry field on the line of interest, and click any mouse button.

  The configuration interface provides individual help for each item in the top level dialog box and throughout the configuration section dialog boxes.

To exit the configuration interface

- Choose **File→Exit...** from the pulldown 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 pulldown menu, and use the file selection dialog box to specify the configuration file to be loaded.

- Using the command line, enter the **load configuration <FILE>** command.

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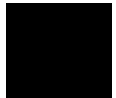
Verifying the Emulator Configuration

The 68360 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 or the emulator/analyzer interface pulldown menu.

- Using the emulator/analyzer interface command line, enter the **display configuration_info sim_chip_selects** or **display configuration_info emsim_chip_selects** command.

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.

Other information from the option register.

To display information about bus interface ports

• Choose **Display→Configuration Info→Bus Interface Ports (SIM)** or
  Display→Configuration Info→Bus Interface Ports (Emulator SIM) from either
  the configuration interface or the emulator/analyzer interface pulldown menu.

• Using the emulator/analyzer interface command line, enter the **display
  configuration_info bus_interface_ports** or **display configuration_info
  embus_interface_ports** command.

  These commands let you display information about bus interface Port E pin
  assignments from the sim (processor) register set or the emsim (emulator) register
  set.

  The resulting display shows the pin assignments for the port.

To display information about the memory map

• Choose **Display→Configuration Info→Memory Map** from either the
  configuration interface or the emulator/analyzer interface pulldown menu.

• Using the emulator/analyzer interface command line, enter the **display
  configuration_info memory_map** command.

  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, current programming of the chip selects in the EMSIM and EMRAM
  register sets, and processor resources.

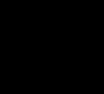## To display information about the reset mode configuration

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

- Using the emulator/analyzer interface command line, enter the **display configuration_info reset_mode** command.

  The display will show data bus size and global chip select memory access size.

To review the upper address mode of the present configuration

- Choose **Display**→**Configuration Info**→**Upper Address Mode** from either the configuration interface or the emulator/analyzer interface pulldown menu.

- Using the emulator/analyzer interface command line, enter the **display configuration_info upper_address** command.

  This selection will show whether the upper address bits are used as A31-A28 or WE3-WE0.

To display information about the present clock input mode

- Choose **Display**→**Configuration Info**→**Clock Input Mode** from either the configuration interface or the emulator/analyzer interface pulldown menu.

- Using the emulator/analyzer interface command line, enter the **display configuration_info clock_mode** command.

  The clock mode depends on the clock module installed in the emulation probe. Refer to the Hewlett-Packard *MC68360 Emulator/Analyzer Installation/Service/Terminal Interface User's Guide* for details.

To display assembly language instructions for setting up the SIM

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

- Using the emulator/analyzer interface command line, enter the **display configuration_info init_source_code** command.

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

To check for configuration inconsistencies

- Choose **Display**→**Configuration Info**→**Diagnostics** from either the configuration interface or the emulator/analyzer interface pulldown menu.

- Using the emulator/analyzer interface command line, enter the **display configuration_info diagnostics** command.

   This command:

   - Checks for inconsistencies between the emulator 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.

To verify the emulator configuration

1 Choose **Display→Configuration Info→Memory Map** from either the configuration interface or the emulator/analyzer interface pulldown menu to display information about the memory map and its correlation with chip selects, internal module register block, and RAM.

This shows more detailed information about the memory map for the 68360. If foreground monitor is selected, the map will include the map term for the foreground monitor.

2 Choose **Display→Configuration Info→Chip Selects (Emulator SIM)** from either the configuration interface or the emulator/analyzer interface pulldown menu to display information about chip selects.

A table appears. It shows the current values in the emsim (Emulator Copy) register set.

6
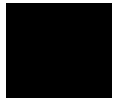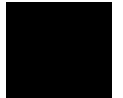
Using the Emulator

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 absolute files.

- Using symbols.

- Executing user programs (starting, stopping, stepping, and resetting the emulator).

- Using software breakpoints.

- Displaying and modifying registers.

- Displaying and modifying memory.

- Changing the interface settings.

- Using system commands.

If you encounter problems when using the emulator/analyzer, refer to the chapter titled "Solving Problems" in the MC68360 emulator/Analyzer Installation/Service/Terminal Interface Manual. Ensure that you are using the appropriate clock module for the system you are probing. Details of clock module selection are also discussed in the MC68360 emulator/Analyzer Installation/Service/Terminal Interface Manual.

Using the EMSIM Registers

The 68360 processor contains a System Integration Module (SIM) which has the external bus interface, eight chip selects, and other circuitry to reduce external logic in a typical microprocessor system. The SIM can be programmed or configured in a variety of ways to suit the need of various systems.

The HP 64780 emulator contains circuitry that accommodates the flexibility of the SIM and maintains consistent emulation features.

The 68360 SIM is configured through the registers in the SIM register class; these registers control how the 68360 uses external signal lines to access memory.

The emulator is configured through the registers in the EMSIM register class. This programming controls how the emulator interprets the signals from the 68360 when accessing emulation memory and passing information to the analysis trace.

Normally, the SIM and EMSIM registers should be programmed with the same values so they will be working together.

One of the primary functions is to provide A31-A28 to the memory mapper and analyzer so they will have the complete 32-bit address bus. This is easy if Port E of the 68360 is programmed as address lines; however, if these lines are programmed as write enables, the corresponding address lines are not available external to the 68360. The chip selects, however, have access to the full 32-bit address inside the 68360. You can therefore locate memory using a chip select at an address that is not possible to decode externally. The emulator can use information in the programming of the chip selects to re-create the upper address lines. This provides a correct address in the analysis trace so that symbolic debugging is possible. Unfortunately, these addresses cannot be recreated in time for memory mapping so the mapper is limited to 28 bits of addressing when the upper address lines are not available.

Normally, the emulator is programmed through the EMSIM registers to match the programming of the 68360 SIM as it will exist after all of the boot-up configuration is complete. This can be done before the boot-up code is run. In fact, the programming of the EMSIM registers is part of the configuration and will be loaded along with the memory map and other configuration items when a configuration file is loaded.

The default programming of the EMSIM register set matches the reset values of the 68360 SIM (refer to the Motorola *MC68360 User's Manual* for specific values).

There are three places where 68360 SIM registers are kept: the target 68360, the emulator 68360, and the emulator configuration file. At any given time during a run of program, the content of the SIM and EMSIM registers will likely be different, and they may both be different from the content of the configuration file. Differences between the SIM and EMSIM registers can affect the accuracy of emulator behavior and displays. The following commands can be used to minimize these differences.

If desired, the programming of the EMSIM register set can be transferred into the 68360 SIM with the **sync_sim_registers to_68360_from_config** command. This happens automatically each time a break to the monitor from emulation reset occurs. This ensures that the 68360 is prepared to properly access memory when a program is downloaded to the emulator.

Alternatively, the emulator's EMSIM register set can be programmed from the 68360 SIM with the **sync_sim_registers from_68360_to_config** command. This is useful if initialization code that configures the 68360 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 use the **sync_sim_registers from_68360_to_config** command to configure the emulator to match the 68360 SIM.

At any time, you can verify if the SIM and EMSIM register sets are programmed the same with the **sync_sim_registers difference** command. Any differences between the two register sets will be listed.

If desired, you can reset the SIM and EMSIM register sets to their default (power up) values with the **sync_sim_registers default_config** command.

Note that that the emulator is configured 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 by tracking instructions that will program the 68360 SIM.

This section shows you how to:

• View the SIM register differences.

• Synchronize to the 68360 SIM registers.

• Synchronize to the EMSIM registers.

To view the SIM register differences

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

- Using the emulator/analyzer interface command line, enter the **sync_sim_registers difference** command.

  A list will appear. It will show only registers having different values between the SIM and EMSIM.

To synchronize to the 68360 SIM registers

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

- Using the emulator/analyzer interface command line, enter the **sync_sim_registers from_68360_to_config** command.

  The contents of the 68360 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 interface pulldown menu.

- Using the emulator/analyzer interface command line, enter the **sync_sim_registers to_68360_from_config** command.

  The contents of the emulation copy of the SIM registers are copied to the 68360 SIM registers.

To restore default values in the EMSIM registers

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

- Using the emulator/analyzer interface command line, enter the **sync_sim_registers default_config** command.

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

To assign an MBAR value for the M68360 register set
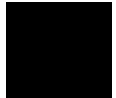
- Press the "Sync $MBAR" Action Key.

  The emulator will read the value of the MBAR register in the target 68360 processor and record the value for use in defining the 68360 register set in the emulator. A browser window will appear. It will show you the value of the target MBAR register that was recorded by this process.

- Press the "Set $MBAR" Action Key.

  A Define command file parameter dialog box will open. Enter any desired value for the emulator's MBAR register in the dialog box. Then click OK.

  The value entered should be the address portion of the MBAR register.

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.

- Using the command line, enter the **load <absolute_file>** command.

  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, enter the following command:

load ecs.x <RETURN>

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

load user_mem absfile <RETURN>

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

load emul_mem absfile <RETURN>

To load absolute files without symbols

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

- Using the command line, enter the **load <absolute_file> nosymbols** command.

To store memory contents into absolute files

- Using the command line, enter the **store memory** command.

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 <RETURN>

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 symbols that are local to a program module 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 Chapter 10, "Setting X Resources").

This section describes how to:

- Load symbols.

- Display global symbols.

- Display 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.

- Using the command line, enter the **load symbols <absolute_file>** command.

Unless you use the **nosymbols** option when loading absolute files, symbols are loaded automatically. However, if you did use the **nosymbols** option when loading the absolute file, you can load the symbols without loading the absolute file again.

This option is particularly useful for loading symbols for files located in target ROM so that you can use symbols with that code.

Examples   To load symbols from the demo program:

load symbols ecs.x <RETURN>

To display global symbols

- Choose **Display**→**Global Symbols**.

- Using the command line, enter the **display global_symbols** command.

  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 <RETURN>

```
Global symbols in ecs.x
Procedure symbols
Procedure name _____ Address range __ Segment _____ Offset
__fflush                         004414 - 0044AD  libc                   0000
_bufsync                         002CE4 - 002D11  libc                   0000
_dbl_to_str                      002EF6 - 00339B  libc                   01E4
_doprnt                          003648 - 004379  libc                   0036
_exec_funcs                      002692 - 0026B1  libc                   002C
_findbuf                         0044AE - 004541  libc                   0000
_startup                         000604 - 000723  env                    0000
_swrite                          004724 - 004759  libc                   0000
_wrtchk                          00475A - 0047F1  libc                   0000
_xflsbuf                         0047F2 - 004883  libc                   0000
atexit                           002666 - 002691  libc                   0000
calloc                           002BCE - 002BFF  libc                   0412
clear_screen                     000A06 - 000A3B  env                    01CA
close                            0008D6 - 00090B  env                    009A
combsort                         00124C - 001443  prog                   028A
do_sort                          00144A - 0014E9  prog                   0488
```

113

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** ().

- Using the command line, enter the **display local_symbols_in <module>** command.

  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:

```
 ─                    Hewlett Packard Emulator/Analyzer: hplsds2 (m68360)        □ ▢

 File Display Modify Execution Breakpoints Trace Settings                        Help

  Action keys:   │ < Demo >    │ Disp Src ( )  │   Trace ( )   │     Run     │ Step Source
  <Your Key>     │   Make      │ Disp Src Prev │ Run Xfer to ( )│    Break    │  Step Asm
  REG KEYS:      │ Sync $MBAR  │  Set $MBAR    │  Boot Code    │ Help Reg ( ) │ ** Help 360 **
  Pick Util      │ Pick Reg 360│ Pick Chip 360 │  Reg 360 All  │ Mod 360 ( )  │ Reg 360 ( )
  Run Util ( )   │ ^STEP #1^   │  Set Chip ( ) │ PRBD 360 All  │ Mod Memory   │ PRBD 360 ( )

 ( ): main                                                                      Recall

 Global symbols in ecs.x
 Procedure symbols
 Procedure name _____ Address range __ Segment _____ Offset
 strncmp                          0000310A - 0000313F libc                 0000
 unlink                           00000B5A - 00000B9F env                  0324
 update_system                    00001210 - 00001257 ▓▓▓g                 0006
 wait_for_io                   ┌─── Global Symbols Display ───┐            0000
 write                         │                               │          0154
 write_hdwr                    │ Display Local Symbols         │g          03E0
                               │ Display Parent Symbols        │
 Static symbols                │ Cut Full Symbol Name          │ment _____ Offset
 Symbol name _____│ Edit File Defining Symbol     │ck          0000
 ?A5                           └───────────────────────────────┘            0000
 JSR_ENTRY                        ▓▓▓▓▓▓▓▓▓▓▓▓ ▓▓▓                           0000
 L_1_IO_check_loop                00000840              env                 0000
 L_2_IO_exit_loop                 0000084C              env                 0000
 MONITOR_MESSAGE                  000081F4 - 000081F7 mondata               0000
 TopOfHeap                        00016FFE                                  0000
 TopOfStack                       00013000              heap                0000

 STATUS:   M68360--Running user program      Emulation trace complete
```

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

Using the command line:

To display local symbols in a module:

display local_symbols_in update_sys <RETURN>

```
Symbols in update_sys(module)
Procedure symbols
Procedure name _____ Address range __ Segment _____ Offset
get_targets                       00165C - 0016E5   prog                   00C8
graph_data                        001986 - 001A3F   prog                   03F2
read_conditions                   0016EC - 00177B   prog                   0158
save_points                       00189A - 00197F   prog                   0306
set_outputs                       001782 - 001813   prog                   01EE
update_system                     00159A - 001655   prog                   0006
write_hdwr                        00181A - 001893   prog                   0286

Filename symbols
Filename _____
update_sys.c
```

To display local symbols in a procedure:

display local_symbols_in update_sys.save_points <RETURN>

```
Symbols in update_sys(module).save_points(procedure)
Procedure special symbols
Procedure special name _____ Address range __ Segment _____ Offset
ENTRY                             00189A            prog                   0306
EXIT                             00197E            prog                   03EA
TEXTRANGE                        00189A - 00197F   prog                   0306
```

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

display local_symbols_in update_sys."update_sys.c":
<RETURN>

```
Symbols in update_sys(module)."update_sys.c":
Source reference symbols
Line range _____ Address range __ Segment _____ Offset
#1-#47                          0015D8 - 0015F7  prog                  0006
#48-#53                         0015F8 - 00160B  prog                  0026
#54-#56                         00160C - 001617  prog                  003A
#57-#59                         001618 - 001637  prog                  0046
#60-#60                         001638 - 00163D  prog                  0066
#61-#61                         00163E - 00164D  prog                  006C
#62-#63                         00164E - 001653  prog                  007C
#64-#64                         001654 - 001661  prog                  0082
#65-#68                         001662 - 001673  prog                  0090
#69-#72                         001674 - 001687  prog                  00A2
#73-#75                         001688 - 00168D  prog                  00B6
#76-#77                         00168E           prog                  00BC
#78-#94                         00169E - 0016B9  prog                  00CC
#95-#95                         0016BA - 0016D1  prog                  00E8
#96-#99                         0016D2 - 0016D7  prog                  0100
#100-#100                       0016D8 - 0016D9  prog                  0106
```

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.

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 pulldown 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.

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 symbol context.

To display the current directory and symbol context

- Choose **Display→Context**.

- Using the command line, enter the **pwd** and **pws** commands.

  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.

```
┌──────────────────────────────────────────────────┐
│ ⊟    Emulator/Analyzer: Current Context           │
├──────────────────────────────────────────────────┤
│ ⓘ  Directory: /usr/hp64000/demo/debug_env/hp64780 │
│    Symbol File: /usr/hp64000/demo/debug_env/hp64780/ecs.x │
│    Symbol Scope: update_sys                       │
├──────────────────────────────────────────────────┤
│                 ┌──────┐                          │
│                 │ Done │                          │
│                 └──────┘                          │
└──────────────────────────────────────────────────┘
```
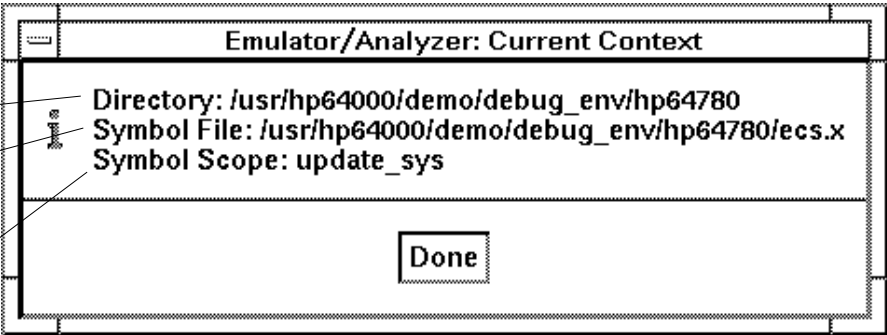
To change the directory context

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

- Using the command line, enter the **cd <directory>** command.

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 Chapter 10, "Setting X Resources").

To change the current working symbol context

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

- Using the command line, enter the **cws <symbol_context>** command. (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 Chapter 10, "Setting X Resources").

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.

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**.

- Using the command line, enter the **run** command.

  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 ()**.

- Using the command line, enter the **run from <address>** command.

Examples

To run from address 920H:

run from 920h <RETURN>

To run programs from the transfer address

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

- Using the command line, enter the **run from transfer_address** command.

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**.

- Using the command line, enter the **run from reset** command.

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 hard reset, and then allowed to run.

To run programs from soft reset

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

- Using the command line, enter the **run from soft_reset** command.

The run from soft reset command pulses the $\overline{\text{RESETS}}$ line to the processor to force a soft reset. In this mode, execution begins from the reset vector without changing any of the internal register values of the 68360.

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 ()**.

- Using the command line, enter the **run until <address>** command.

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 address main <RETURN>

125

To stop (break from) user program execution

- Choose **Execution→Break**.

- Using the command line, enter the **break** command.

  This command generates a break to the background 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 <RETURN>

To step high-level source lines

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

- Using the command line, enter the **step source** command.

  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 <RETURN>

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

step source from main <RETURN>

To step assembly-level instructions

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

- Using the command line, enter the **step** command.

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 <RETURN>

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

step 8 <RETURN>

To step a number of instructions from a specified address:

step 16 *from* 920h <RETURN>

To reset the emulation processor

- Choose **Execution**→**Reset**.

- Using the command line, enter the **reset** command.

  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.
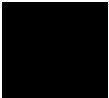
Using Software Breakpoints

Software breakpoints provide a way to accurately stop the execution of your program at selected locations.

Note

Version A.04.00 or greater of the HP 64700 system firmware provides support for permanent as well as temporary breakpoints. If your version of HP 64700 system firmware is less than A.04.00, only temporary breakpoints are supported.

When you set a software breakpoint at an address, the instruction at that address is replaced with a BGND instruction. When the BGND instruction is executed, the emulator enters its monitor state, and the original instruction is restored in the user program.

If the BGND instruction was not inserted as the result of a **modify software_breakpoints set** command, the "Undefined software breakpoint" message is displayed on the status line.

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 emulation RAM, and loading the absolute file.)

Another way to break user program execution at a certain point is to break on the analyzer trigger.

This section shows you how to:

- Display the breakpoints list.

- Enable/disable breakpoints.

- Set a permanent breakpoint.

- Set a temporary breakpoint.

- Set all breakpoints.

- Deactivate a breakpoint.

- Re-activate a breakpoint.

- Clear a breakpoint.

- Clear all breakpoints.

To display the breakpoints list

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

- Using the command line, enter the **display software_breakpoints** command.

The breakpoints display shows the address and status of each breakpoint currently defined. If symbolic addresses are turned on (when setting the display modes), the symbolic label associated with a breakpoint is also displayed. Also, the breakpoints display shows whether the breakpoint feature is enabled or disabled.

```
Software breakpoints  :enabled
   address         label                                        status
   000FC8          main(module)."main.c":              line   96  pending
   000FCC @sp      main(module)."main.c":              line   97  permanent
   000FD2 @sp      main(module)."main.c":              line   98  inactivated
   00159A          update_sys(module)."update_sys.c":  line   47  temporary
```

The status of a breakpoint can be:

temporary   Which means the temporary breakpoint has been set but not encountered during program execution. These breakpoints are removed when the breakpoint is encountered.

permanent   Which means the permanent breakpoint is active. Permanent breakpoints remain active after they are encountered during execution.

inactivated   Which 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   Which 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.

To enable/disable breakpoints

- Choose the **Breakpoints→Enable** toggle.

- When displaying the breakpoint list, press and hold the *select* mouse button and then choose **Enable/Disable Software Breakpoints** from the pop-up menu.

- Using the command line, enter the **modify software_breakpoints enable** or **modify software_breakpoints disable** command.

The breakpoints feature must be enabled before you can set, inactivate, or clear breakpoints.

If breakpoints were set when the feature was disabled, they are "inactivated" when the feature is re-enabled, and you must set them again.

The emulator/analyzer interface will enable software breakpoints whenever the **XEnv_68k_except** symbol is present in the symbol data base.

131

The run-time library provided with the 68360 C Cross Compiler uses software breakpoints to interrupt program execution when exceptions (for example, divide by zero) are encountered. If software breakpoints are disabled, exception processing may result in "access to guarded memory" errors and/or other unpredictable behavior. To prevent this, a special global symbol, **XEnv_68k_except**, is included in the library.

When the **XEnv_68k_except** symbol is present, the 68360 emulator writes a value to this location. The value tells the run-time library to use the BGND instruction to perform a software break.

Examples   To enable software breakpoints using the breakpoints display pop-up menu:

Bring up menu and choose this item to change states.

```
┌──────────────────────────────────────────────────────────────────────┐
│ ▭            Hewlett Packard Emulator/Analyzer: hplsds2 (m68360)   ▫ ▯ │
├──────────────────────────────────────────────────────────────────────┤
│ File Display Modify Execution Breakpoints Trace Settings        Help   │
├──────────────────────────────────────────────────────────────────────┤
│ Action keys:  │ < Demo >   │ Disp Src ( )│ Trace ( )  │  Run      │ Step Source │
│ <Your Key>    │ Make       │ Disp Src Prev│Run Xfer to( )│ Break   │ Step Asm    │
│ REG KEYS:     │ Sync $MBAR │ Set $MBAR   │ Boot Code  │ Help Reg ( )│** Help 360 **│
│ Pick Util     │Pick Reg 360│Pick Chip 360│ Reg 360 All│ Mod 360 ( )│ Reg 360 ( ) │
│ Run Util ( )  │ ^STEP #1^  │ Set Chip ( )│PRBD 360 All│Mod Memory │ PRBD 360 ( )│
├──────────────────────────────────────────────────────────────────────┤
│ ( ): main                                                     │ Recall │
├──────────────────────────────────────────────────────────────────────┤
│ Software breakpoints  :disabled                                        │
│   address         label                               status           │
│  00000FC2 @sp    main(module)."main.c":        line   96  inactivated  │
│  00000FD6 @sp    main(module)."main.c":        line   97  inactivated  │
│  00000FFA @sp    main(module)."main.c":        line  103  inactivated  │
│  00001714               prog|update_sys                   inactivated  │
│  0000171A        update_sys(module)."update_sys.c": line 47 inactivated│
│                            ┌──────────────────────────────────────┐    │
│                            │ 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        │    │
│                            └──────────────────────────────────────┘    │
├──────────────────────────────────────────────────────────────────────┤
│ STATUS:   M68360--Running in monitor    Software break: 0000171A@sp    │
└──────────────────────────────────────────────────────────────────────┘
```

To set a permanent breakpoint

- When displaying memory in mnemonic format, position the mouse pointer over the program line at which 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 ()**

- Using the command line, enter the **modify software_breakpoints set <address> permanent** command.

Permanent breakpoints are available if your version of HP 64700 system firmware is A.04.00 or greater.

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

Note that you must only set software breakpoints at memory locations which contain instruction opcodes (not operands or data).

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   To set permanent breakpoints using the mnemonic memory display pop-up menu:

Click this line to set a breakpoint.

Click this line to clear a breakpoint. (Asterisks mark set breakpoints.)

Bring up menu and choose this item to set (or clear) a breakpoint on the highlighted line.

To set a temporary breakpoint

- Place an absolute or symbolic address in the entry buffer; then, choose **Breakpoints→Temporary ()** (or **Breakpoints→Set ()** if your version of HP 64700 system firmware is less than A.04.00).

- Using the command line, enter the **modify software_breakpoints set <address> temporary** or **modify software_breakpoints set <address>** command.

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

  Note that you must only set software breakpoints at memory locations which contain instruction opcodes (not operands or data).

  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 all breakpoints

- When displaying the breakpoint list, position the mouse pointer within the breakpoints display screen, press and hold the *select* mouse button, and choose **Set All Breakpoints** from the pop-up menu.

- Choose **Breakpoints→Set All**.

- Using the command line, enter the **modify software_breakpoints set** command.

  Breakpoints must be enabled before being set.

To deactivate a breakpoint

- When displaying breakpoints, position the mouse pointer over the line displaying the active breakpoint and click the *select* mouse button. Or, press and hold the *select* mouse button and choose **Set/Inactivate Breakpoint** from the pop-up menu.

  A deactivated breakpoint remains in the breakpoint list and can be re-activated later. Deactivating a breakpoint is different than clearing a breakpoint because a cleared breakpoint is removed from the breakpoints list.

To re-activate a breakpoint

- When displaying breakpoints, position the mouse pointer over the line displaying the inactivated breakpoint and click the *select* mouse button. Or, press and hold the *select* mouse button and choose **Set/Inactivate Breakpoint** from the pop-up menu.

  The "inactivated" breakpoint either becomes "temporary" (or "pending") if it was set as a temporary breakpoint or "permanent" if it was set as a permanent breakpoint.

Examples   To re-activate breakpoints using the breakpoints display pop-up menu:

Change status with a mouse click on this line (menu and highlight do not appear). ────

Choose this menu item to change the state of the highlighted breakpoint.

```
┌──────────────────────────────────────────────────────────────────────┐
│ ━━         Hewlett Packard Emulator/Analyzer: hplsds2 (m68360)    □ □ │
├──────────────────────────────────────────────────────────────────────┤
│ File  Display  Modify  Execution  Breakpoints  Trace  Settings   Help │
├──────────────────────────────────────────────────────────────────────┤
│ Action keys:  │ < Demo > │ Disp Src ( ) │  Trace ( )   │   Run   │ Step Source │
│ <Your Key>    │   Make   │ Disp Src Prev│ Run Xfer to ( )│ Break │ Step Asm    │
│ REG KEYS:     │ Sync $MBAR │ Set $MBAR  │  Boot Code   │ Help Reg ( ) │ ** Help 360 ** │
│ Pick Util     │ Pick Reg 360 │ Pick Chip 360 │ Reg 360 All │ Mod 360 ( ) │ Reg 360 ( ) │
│ Run Util ( )  │ ^STEP #1^ │ Set Chip ( ) │ PRBD 360 All │ Mod Memory │ PRBD 360 ( ) │
├──────────────────────────────────────────────────────────────────────┤
│ ( ): main                                                      Recall │
├──────────────────────────────────────────────────────────────────────┤
│ Software breakpoints  :enabled                                        │
│   address        label                                   status     ▲ │
│  00000FC2 @sp    main(module)."main.c":        line   96  inactivated │
│  00000FD6 @sp    main(module)."main.c":        line   97  inactivated │
│  00000FE6 @sp    main(module)."main.c":        line  100  permanent   │
│  00000FFA @sp    main(module)."main.c":        line  103  permanent   │
│  00001714                      prog update_sys            inactivated │
│  0000171A        update_sys(module)."upd┌────────────────────────────┐│
│                                         │ 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 │
│                                         └────────────────────────────┘│
├──────────────────────────────────────────────────────────────────────┤
│ STATUS:   M68360--Running in monitor      Software break: 00000171a@sp │
└──────────────────────────────────────────────────────────────────────┘
```

To clear a breakpoint

- When displaying memory in mnemonic format, position the mouse pointer over the program line at which you wish to clear a currently set breakpoint (notice the asterisk at the left of the line) 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.

- When displaying breakpoints, position the mouse pointer over the line displaying the breakpoint you wish to clear, press and hold the *select* mouse button, and choose **Clear (delete) Breakpoint** from the pop-up menu.

- Place an absolute or symbolic address in the entry buffer; then choose **Breakpoints→Clear ()**.

- Using the command line, enter the **modify software_breakpoints clear <address>** command.

  When you clear a breakpoint, it is removed from the breakpoints list.

Examples   To clear a software 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 (m68360)         □ □
 File Display Modify Execution Breakpoints Trace Settings                Help

 Action keys:  │ < Demo >  │ Disp Src ( ) │   Trace ( )  │    Run    │ Step Source
 <Your Key>    │   Make    │ Disp Src Prev│Run Xfer to ( )│   Break   │  Step Asm
 REG KEYS:     │Sync $MBAR │  Set $MBAR   │  Boot Code   │ Help Reg ( )│** Help 360 **
 Pick Util     │Pick Reg 360│Pick Chip 360│  Reg 360 All │ Mod 360 ( ) │ Reg 360 ( )
 Run Util ( )  │ ^STEP #1^ │  Set Chip ( )│ PRBD 360 All │Mod Memory │ PRBD 360 ( )

 ( ): main                                                             Recall

 Software breakpoints  :enabled
    address        label                                     status
   00000FC2 @sp     main(module)."main.c":          line   96  inactivated
   00000FD6 @sp     main(module)."main.c":          line   97  inactivated
   00000FE6 @sp     main(module)."main.c":          line  100  permanent
   00000FFA @sp     main(module)."main.c":          line  103  permanent
   00001714              prog update_sys ┌──────────────────────────────────┐
   0000171A      update_sys(module)."update_│  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     │
                                            └──────────────────────────────────┘

 STATUS:    M68360--Running in monitor       Software break: 00000171a@sp
```

To clear all 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**.

- Using the command line, enter the **modify software_breakpoints clear** command.
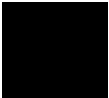
Displaying and Modifying Registers

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

You can display the contents of an individual register or of all the registers.

To display register contents

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

- Using the command line, enter the **display registers <register class>** command.

  The **<register class>** token requires the name of a class of registers, such as **BASIC** or **SIM**. The display will show all registers in the **<register class>** you select, along with the present values of the registers.

Obtaining mnemonic displays of the 68360 registers using the Action Keys

- To set up to display registers for the emulated 68360, use the "Sync $MBAR" Action Key.

- To obtain a record of the present contents of all SIM60 and CPM registers in one listing, press the "Reg 360 All" Action Key.

- To view the contents of a single SIM60 or CPM register, press the "Pick Reg 360" Action Key. Within the appropriate browser window, click on the name of the register to be displayed. Then press the "Reg 360 ()" Action Key.

- To obtain a record of the present content of all parameter RAMs and Buffer Descriptors in one browser, press the "PRBD 360 All" Action Key.

- To view the contents of a single Parameter RAM and its associated Buffer Descriptors, place the name of the desired channel in the entry buffer and press the "PRBD 360 ()" Action Key.

All complex register displays are supported with an "expanded" display to interpret the meaning of the bits that make up the register value. The descriptions of values will change as the register content changes to correspond to the present register values.

Expanded register displays show dependencies. For example, if a bit is masked by another register, the expanded display will show that the bit is masked.

When a bit governs output format (such as AppleTalk or Asynch HDLC), the expanded display will show the present format.

The descriptions in expanded displays correspond to interpretations and descriptions in Motorola manuals.

Expanded register displays are only available for registers whose individual bits have unique configuring values.

The following expanded display of the pepar register was obtained from the
SIM60/CMP Register Browser Window by placing pepar in the entry buffer and
clicking the action key labeled Reg 360 ():

```
CREATED: Fri Mar 31 11:39:27 1995
>addr=0x00001016 pepar=0x0000
Bit     Name                Setting
------  -------------------
-------------------------------------------
15                          Reserved
14-12   SINTOUT             Reserved
11                          Reserved
10-9    CF1MODE             CONFIG1 input function
8       ~|PIPE1/~RAS1DD     ~|PIPE1 output function
7       A28-31/~WE3-~WE0    Address 31-28 input/output functions
6       ~OE/AMUX            ~OE output function
5       PWW                 PEPAR has not been written
4       ~CAS2,3/~IACK3,6    ~CAS2,3 output functions
3                           Reserved
2       ~CAS0,1/~IACK1,2    ~CAS0,1 output functions
1       ~CS7/~IACK7         ~CS7 output function
0       ~AVEC(AVEC0)/~IACK5 ~AVEC input function
```
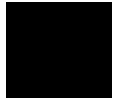
To modify register contents

- 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.

Emulator/Analyzer: Modify Register

Modify Register

Name | PC | Recall

Value | | Recall

Value Type | Hex

☐ Read Current Register Value

| OK | Apply | Cancel |
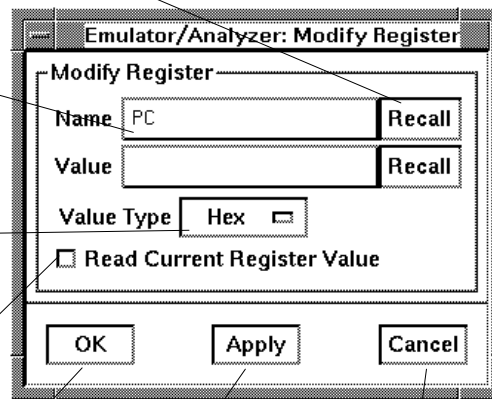
- Using the command line, enter the **modify register <register> to <value>** command.

To modify registers using the Action Keys

- To modify the content of a SIM60 or CPM register, press the "Pick Reg 360"
  Action Key. Within the appropriate browser window, click on the name of the
  register to be modified. Then press the "Mod 360 ()" Action Key. Type the
  desired value in the Define command file parameter dialog box and click OK.

- To modify the contents of a Parameter RAM or Buffer Descriptor, press the "Mod
  Memory" Action Key. The Define command file parameter dialog box will appear
  three times. In the first appearance, enter the desired address; next, enter size; and
  finally, enter value. Click OK after each entry.
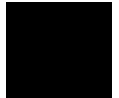
Displaying and Modifying Memory

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** pulldown 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**.

- Using the command line, enter the **display memory mnemonic previous_display** command.

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.

- Using the command line, enter the **display memory <address> blocked <size>**
  command.

  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* <RETURN>

```
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** <RETURN>

```
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 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**.

- Using the command line, enter the **display memory repetitively** command.

  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.

- Using the command line, enter the **modify memory** command.

  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.
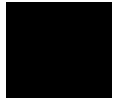
Displaying Data Values

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**.

- Using the command line, enter the **display data** command.

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.

- Using the command line, enter the **display data <address>** command.

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.

- Using the command line, enter the **display data , <address>** command.

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 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 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.

Emulator/Analyzer: Display Modes

┌ 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

| OK | | Apply | | Cancel |

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.
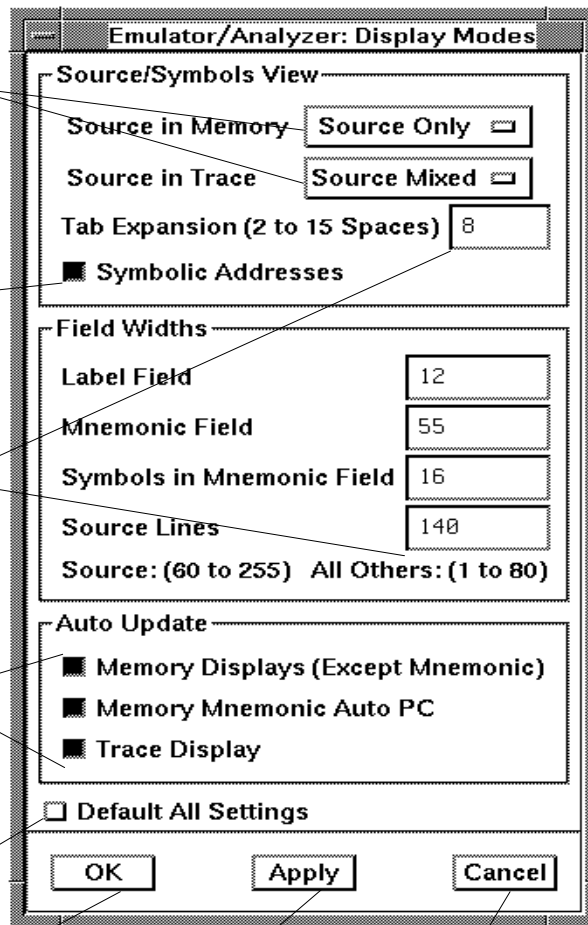
Source/Symbols View

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 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 absolute 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 turn off memory display updates 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.
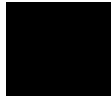
Using System Commands

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 the **set <VAR>** command.

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" <RETURN>

After you set an environment variable from within the Softkey Interface, you can verify the value of it by entering **!set <RETURN>**.

To display the name of the emulation module

- Using the command line, enter 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 <RETURN>
```

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.

- Using the command line, enter the **display event_log** command.

  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.

- Using the command line, enter the **display error_log** command.

  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 At 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.

The interface will choose the "vi" editor as its default editor, unless you specify another editor by setting an X resource. Refer to the Chapter 10, "Setting X Resources" 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.
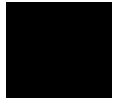
Examples   To edit a file that defines a symbol:

Choosing this menu
item brings up a
terminal window
with an edit session
open on the file
where the
highlighted symbol is
defined.

To edit a file at a source line:

Choosing this menu item brings up a terminal window with an edit session open on the file where the highlighted source line exists.


```
Hewlett Packard Emulator/Analyzer: hplsds2 (m68360)

File  Display  Modify  Execution  Breakpoints  Trace  Settings                Help

Action keys:  | < Demo >     | Disp Src ( )  | Trace ( )      | Run        | Step Source
<Your Key>    | Make         | Disp Src Prev | Run Xfer to ( )| Break      | Step Asm
REG KEYS:     | Sync $MBAR   | Set $MBAR     | Boot Code      | Help Reg ( )| ** Help 360 **
Pick Util     | Pick Reg 360 | Pick Chip 360 | Reg 360 All    | Mod 360 ( ) | Reg 360 ( )
Run Util ( )  | ^STEP #1^    | Set Chip ( )  | PRBD 360 All   | Mod Memory  | PRBD 360 ( )

( ): save_points                                                          Recall

Memory  :@sp :mnemonic :file = update_sys(module)."update_sys.c":
   address  label        data
      262       MAKEBAR(ARG6); /* This is a delay macro; it is used to create SP
      263
      264       old_data[curr_loc].temp = current_temp;
      265       old_data[curr_loc].humid = current_humid;
      266       curr_loc++;
      267       if (curr_loc > NUM_OF_OLD) curr_loc = 0;   /*BUG!!!!!*/
      268                   Choose Action for Highlighted Line
      269       temp_tot=0;
      270       for (i=0;i<  Set/Clear Software Breakpoint
      271          temp_to  Edit Source
      272
      273       old_data[cu Run Until                       t/(float)(NUM_OF_OLD
      274
      275       humid_tot=0  Trace After
      276       for (i=0;i<  Trace Before
      277          humid_t   Trace About
      278                   Trace Until

STATUS:   M68360--Emulation reset       Emulation trace complete        ◄ ►
```

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.

- Using the command line, enter the **copy** command.

  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** pulldown 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.
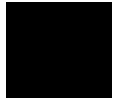
To save peripheral register settings to a file

1 Press the "Pick Util" Action Key.

2 In the browser window, highlight save68360registers and click Done.

3 Press the "Run Util()" Action Key.

4 Type the desired directory/filename to contain register values in the Define command file parameter dialog box, and click OK.

To load peripheral register settings from a file

1 Press the "Pick Util" Action Key.

2 In the browser window, highlight load68360registers and click Done.

3 Press the "Run Util()" Action Key.

4 Type the name of the directory/filename that contains the desired register values into the Define command file parameter dialog box, and click OK.

To remove all temporary files

1 Press the "Pick Util" Action Key.

2 In the browser window, highlight clean68360util and click Done.

3 Press the "Run Util()" Action Key.

To generate boot code for configuring the SIM60 unit

1 Press the "Boot Code" Action Key.

2 When the SIM60 Boot Code browser window opens, press the Save to File... pushbutton and enter the name of the file to contain the generated boot code.

3 Click OK.

4 Assemble and link the file of generated boot code with your code.

To open a terminal emulation window

- Choose **File→Term...**

This command opens a terminal window into the current working directory context.

Using emulator support for the M68360 Companion Mode

This section shows you how to use the M68360 action keys to develop products that use the M68360 Companion Mode. Through the action keys, you can perform such actions as viewing registers, configuring registers, developing boot code, and running programs.

Press the Action Key labeled "**Help 360**". A window will open, providing general information to help you get started using the M68360 Companion Mode through the Action Keys.

Refer to the section on plugging the emulator into the Motorola QUADS Target System in Chapter 2 for an example of plugging into a target system that contains a 68360 master chip and a 68360 slave chip.

Refer to the section on setting up custom action keys in Chapter 10 for an example of how the action keys are developed and what they do in the emulator.

Tasks you may wish to perform when using the M68360 companion Mode

The following paragraphs show you how to perform typical development operations supported in the action keys of the M68360 Graphical User Interface. For further details, refer to the help screen available by pressing the "**Help 360**" Action Key.

- To obtain a record of the present contents of all SIM60 and CPM registers in one listing, press the "Reg 360 All" Action Key.

- To view the contents of a single SIM60 or CPM register, press the "Pick Reg 360" Action Key. Within the Register List browser window, click on the name of the register to be displayed. Then press the "Reg 360 ()" Action Key.

- To modify the content of a SIM60 or CPM register, press the "Pick Reg 360" Action Key. Within the Register List browser window, click on the name of the register to be modified. Then press the "Mod 360 ()" Action Key. Type the desired value in the Define command file parameter dialog box and click OK.

- To obtain a record of the present content of all parameter RAMs and Buffer Descriptors in one browser, press the "PRBD 360 All" Action Key.

- To view the contents of a single Parameter RAM and its associated Buffer Descriptors, place the name of the desired channel in the entry buffer and press the "PRBD 360 ()" Action Key.

- To modify the contents of a Parameter RAM or Buffer Descriptor, press the "Mod Memory" Action Key. The Define command file parameter dialog box will appear three times. In the first appearance, enter the desired address; next, enter size; and finally, enter value. Click OK after each entry.

- To select the M68360 slave module whose registers will be viewed through the Graphical User Interface, press the "Pick Chip 360" Action Key. In the Available M68360 Slaves browser window, click on the name of the desired M68360 slave module, and click Done. Then press the "Set Chip ()" Action Key.
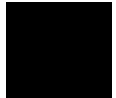
- To assign a new base address to contain the register set of an M68360 chip, press the "Pick Util" Action Key. In the Utilities Selection browser window, highlight assign68360chip, and click Done. Press the "Run Util()" Action Key. Type the new base address in the Define command file parameter dialog box, and click OK.

- To save peripheral register settings to a file. Press the "Pick Util" Action Key. In the Utilities Selection browser window, highlight save68360registers and click Done. Press the "Run Util()" Action Key. Type the desired directory/filename to contain register values in the Define command file parameter dialog box, and click OK.

- To restore peripheral register settings to files, Press the "Pick Util" Action Key. In the Utilities Selection browser window, highlight load68360registers and click Done. Press the "Run Util()" Action Key. Type the name of the directory/filename that contains the desired register values into the Define command file parameter dialog box, and click OK.

- To remove all temporary files that have been created during the development session, press the "Pick Util" Action Key. In the Utilities Selection browser window, highlight clean68360util and click Done. Press the "Run Util()" Action Key.

- To generate boot code for configuring the SIM60 unit, press the "Boot Code" Action Key. When the SIM60 Boot Code browser window opens, press the Save to File... pushbutton and enter the name of the file to contain the generated boot code; then click OK. Assemble and link the file of generated boot code with your code.

For more information

- General information about using the Action Key solution to the M68360 Companion Mode is available by pressing the "**Help 360**" Action Key.

- Detailed information for configuring a particular SIM60 or CPM register can be obtained by placing the name of the register in the entry field and pressing the "Help Reg ()" Action Key.

- Help for understanding how action keys work in the Graphical User Interface is available in Chapter 10, "Setting X Resources", and in the online file named **$HP64000/lib/X11/app-defaults/HP64_Softkey**, under the discussion called XcHotkey:Action Keys.

- Refer to the file named **README040360** in the directory **$HP64000/inst/emul/64780A/compmode** to see files developed to support companion mode use of an HP 64780 M68360 or HP 64783 M68040 emulator system. This file can easily be modified to support M68360 companion mode with the HP 64747/B1489 M68030 emulator system, as well.

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.

    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-Bus Analyzer

How to record program execution in real-time

Power of the Emulation-Bus Analyzer

The *emulation-bus analyzer* is a powerful tool that allows you to view the execution of your program in real-time. Extensive triggering and sequencing capability ensures that the analyzer captures only the information you need so you don't spend time searching through long trace lists to find the information that is of interest.

The Graphical User Interface has menus that let you specify some simple analyzer measurements like tracing after, about, or before an address. You can also specify qualifications for which states get stored and which states can be prestored; the analyzer can prestore up to two states before each qualified store state.

The analyzer has much more capability than is available in the menus. You can access this capability by using the command line to make your trace specifications. Use of the command line is also covered in this chapter.

Once a trace specification command is entered, either with the menus or the command line, it can be recalled, edited if desired, and executed again. Also, trace specifications and trace data can be stored to files and loaded from files.

If you encounter problems when using the emulator/analyzer, refer to the chapter titled "Solving Problems" in the MC68360 Emulator/Analyzer Installation/Service/Terminal Interface Manual.

Making Simple Trace Measurements

You can make simple records of the processor's bus activity using just a few analyzer commands. When you set up the analyzer to record processor bus activity, you are preparing to make a *trace measurement*. During the trace measurement, the analyzer saves a record of the bus activity in trace memory. The display of the trace memory content is called the *trace list*.
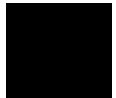
The information captured at the occurrence of each clock is called a state. When a captured state matches your specification for the trigger state, the analyzer identifies it as the trigger state and stores it in trace memory.

The default specification for the trigger state is "any state." When you start a trace measurement using the default trace specification, the analyzer will identify the first state it captures as the trigger state and fill the remaining space in the trace memory with the states that follow it. A trace is said to be complete when the trace memory is filled with captured states, and the trigger state resides at its specified point in the trace memory (the first state captured in memory, by default).

When a trace measurement is started, 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 or when the analyzer hasn't filled its trace memory, 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 keyboard keys to position the trace list on screen. To speed up the display of traces, you can reduce the depth of the trace list. Also, when entering trace commands, you can recall and modify preceding trace commands to speed command entry.

To start a trace measurement

- Choose **Trace→Everything**.

- 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 deep analyzer, 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 *MC68360 Emulator/Analyzer Installation/Service/Terminal Interface 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

Start the demo program and trace from the program start:

```
Startemul
reset
trace
run from transfer_address
```
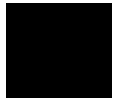
To stop a trace measurement

- Choose **Trace→Stop**.

- 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**.

- Choose **Display→Trace**.

- Using the command line, enter:
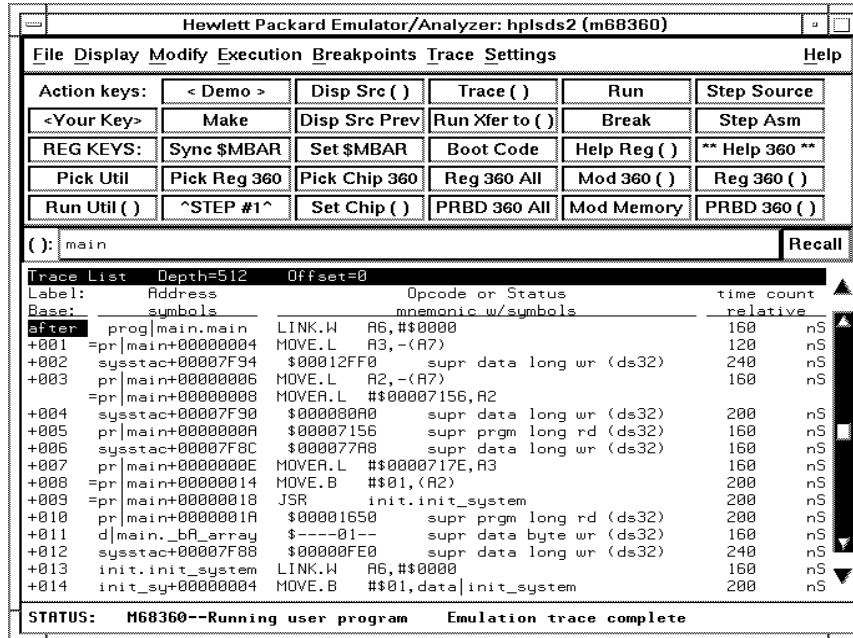
  display trace

  When you complete a trace measurement, you will want to see the results. The
  display trace command shows you the current trace list. The trace display is
  updated each time you enter a new **trace** command, until you display some other
  data using the **display** command. (See the **set update** command in "Emulator
  Commands" for details.)

Whether source lines, disassembled trace states, or symbols are included in the display depends on the modes you choose with the
Settings→Source/Symbols Modes or **Settings→Display Modes** pulldown menu items.

Example   A simple trace list resembles:

```
 ┌──────────────────────────────────────────────────────────────────────────┐
 │ □        Hewlett Packard Emulator/Analyzer: hplsds2 (m68360)      □ │□│   │
 ├──────────────────────────────────────────────────────────────────────────┤
 │ File  Display  Modify  Execution  Breakpoints  Trace  Settings      Help  │
 ├──────────────────────────────────────────────────────────────────────────┤
 │ Action keys:  │ < Demo >   │ Disp Src ( ) │  Trace ( )   │  Run   │ Step Source  │
 │ <Your Key>    │ Make       │ Disp Src Prev│ Run Xfer to ( )│ Break │  Step Asm    │
 │ REG KEYS:     │ Sync $MBAR │ Set $MBAR    │  Boot Code   │ Help Reg ( ) │ ** Help 360 ** │
 │ Pick Util     │ Pick Reg 360│ Pick Chip 360│  Reg 360 All │ Mod 360 ( ) │ Reg 360 ( )  │
 │ Run Util ( )  │ ^STEP #1^  │ Set Chip ( ) │ PRBD 360 All │ Mod Memory │ PRBD 360 ( ) │
 ├──────────────────────────────────────────────────────────────────────────┤
 │ ( ): main                                                        Recall   │
 ├──────────────────────────────────────────────────────────────────────────┤
 │ Trace List   Depth=512    Offset=0                                        │
 │ Label:      Address                Opcode or Status         time count    │
 │ Base:         symbols              mnemonic w/symbols        relative      │
 │ after    prog|main.main   LINK.W   A6,#$0000                  160    nS    │
 │ +001   =pr|main+00000004  MOVE.L   A3,-(A7)                   120    nS    │
 │ +002    sysstac+00007F94  $00012FF0     supr data long wr (ds32)  240 nS   │
 │ +003    pr|main+00000006  MOVE.L   A2,-(A7)                   160    nS    │
 │         =pr|main+00000008 MOVEA.L  #$00007156,A2                            │
 │ +004    sysstac+00007F90  $000080A0     supr data long wr (ds32)  200 nS   │
 │ +005    pr|main+0000000A  $00007156     supr prgm long rd (ds32)  160 nS   │
 │ +006    sysstac+00007F8C  $000077A8     supr data long wr (ds32)  160 nS   │
 │ +007    pr|main+0000000E  MOVEA.L  #$0000717E,A3              160    nS    │
 │ +008    =pr|main+00000014 MOVE.B   #$01,(A2)                  200    nS    │
 │ +009    =pr|main+00000018 JSR      init.init_system           200    nS    │
 │ +010    pr|main+0000001A  $00001650     supr prgm long rd (ds32)  200 nS   │
 │ +011    d|main._bA_array  $----01--     supr data byte wr (ds32)  160 nS   │
 │ +012    sysstac+00007F88  $00000FE0     supr data long wr (ds32)  240 nS   │
 │ +013    init.init_system  LINK.W   A6,#$0000                  160    nS    │
 │ +014    init_sy+00000004  MOVE.B   #$01,data|init_system      200    nS    │
 ├──────────────────────────────────────────────────────────────────────────┤
 │ STATUS:   M68360--Running user program    Emulation trace complete        │
 └──────────────────────────────────────────────────────────────────────────┘
```

To display the trace status

- Choose **Display**→**Status.**

- Using the command line, display the trace status with the **display status** command.

When you complete a trace measurement, you'll want to see the results.

The commands above show the current emulator and analyzer status. The analyzer status shows:

- whether the trace has completed (trace memory is full)
- analyzer arm condition
- whether the trigger has been found
- number of states captured
- current sequencer state and occurrence count

Example   In the following example trace status display, the screen shows that the emulation trace has completed, an analyzer arm (a condition to activate the analyzer) was not defined for this measurement, the analyzer trigger was captured in memory before the analyzer trace completed, 512 trace states were captured (511 states plus the trigger state), and one analyzer sequence term was needed to satisfy the analyzer trigger.

```
Status

_____

Emulator Status

  M68360--Running user program

Trace Status

  Emulation trace complete
  Arm ignored
  Trigger in memory
  Arm to trigger ?
  States 512 (512) 0..511
  Sequence term 2
  Occurrence left 1
```

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.

- 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 deep analyzer, 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 *MC68360 Emulator/Analyzer Installation/Service/Terminal Interface 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
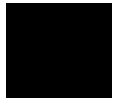
- Choose **Trace**→**Trace Spec** and use the dialog box to select and edit a trace command.

- Using the command line, enter:

`trace modify_command`

The Trace Specification Selection dialog box contains a list of trace specifications executed during the emulation session as well as any predefined trace specifications present at interface startup.

You can predefine trace specifications and set the maximum number of entries for the dialog box by setting X resources (see Chapter 10, "Setting X Resources").

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 define a simple trigger qualifier

- Enter your trigger qualifier (such as, **address 1000h**) in the entry buffer. Then in the menu bar, click on: **Trace→After()**, **Trace→Before(),** or **Trace→About().**

- When displaying memory in mnemonic format, position the mouse pointer over the program line that you wish to use as a trigger, press and hold the *select* mouse button, and choose **Trace After, Trace Before,** or **Trace About** from the pop-up menu.

- Using the command line, use the **trace** command to specify a trigger.

The default option for the analyzer is to begin to fill trace memory immediately after the start of the trace. The trace completes when trace memory is full and the trigger has been captured.

The trigger is a reference event in a trace list. You select trigger position to see activity leading up to the trigger event, or following the trigger event, or both.

Example

To trigger a trace measurement after the demo program executes the Init_system procedure, place init_system in the entry buffer and choose **Trace→After()**, or on the command line, enter:

```
trace after long_aligned init_system
```

The "long_aligned" option ensures that if the address of the trigger event is fetched as part of a long-aligned access, the analyzer will still be able to recognize it.

To capture a trace leading up to the address of gen_ascii_data, and then break to the monitor when that trigger event occurs, place gen_ascii_data in the entry buffer and choose **Trace→Until()**, or on the command line, enter:

```
trace before long_aligned gen_ascii_data
break_on_trigger
```

To capture a trace of activity both preceding and following the write_hdwr symbol in the update_sys module, place update_sys.write_hdwr in the entry buffer and choose **Trace→About()**, or on the commad line, enter:

trace about long_aligned update_sys.write_hdwr

To specify a trigger and set the trigger position

- Place the trigger specification desired (such as **address 1000h**) in the entry buffer, and then choose **Trace→After()**, **Trace→Before()**, or **Trace→About()**.

- When displaying memory in mnemonic format, position the mouse pointer over the program line that you wish to use as the trigger, press and hold the *select* mouse button, and choose **Trace After, Trace Before,** or **Trace About** from the pop-up menu.

- Using the command line, select **trace after**, **trace before**, or **trace about** to set the trigger position.

Normally the analyzer begins to save processor activity whenever the trace is started. By selecting trigger position, you can specify which portion of processor activity you will view in the trace list.

The **trace after** command causes the analyzer to fill its trace memory with processor activity that occurred after the trigger event.

The **trace before** command causes the analyzer to fill its trace memory with processor activity that occurred before the trigger event.

The **trace about** command causes the analyzer to fill its trace memory with processor activity that occurred before and after the trigger event. With this command, the trigger event is positioned at the center of the trace.

The actual trigger position in the trace list is within +/-3 states of the position specified.

When you enter a **trace about** command, the trigger state (line 0) is normally labeled "about". However, if there are three or fewer states before the trigger, the trigger state is labeled "after", and if there are three or fewer states after the trigger, the trigger state is labeled "before".

Example

To trace on states before the demo program accesses the current humidity, enter:

```
trace before address current_humid status write
set symbols on
display trace
```

To define a simple storage qualifier

- Place your storage qualifier in the entry buffer (such as **status read**), and then choose **Trace→Only().**

- Using the command line, use the **only** option in the **trace** command.

All captured states are stored by default. However, you can qualify which states get stored with the **only** option to the **trace** command.

Example

When you are running the demo program, to store only accesses to the address "target_temp", place target_temp in the entry buffer, and then choose **Trace→Only(),** or on the command line, enter:

```
trace only target_temp
```

If you are having problems tracing

- Check the emulator configuration. Choose Modify→Emulator Config.... Then in the Emulator Pod Settings dialog box:

  - You may need to select Clock 01 Drive to Target = Buffered.

  - You may need to select Buffer AS, DS and R/W = Yes.

  Interaction problems between the emulator and the target system may affect tracing. These problems may be overcome by selecting proper buffering of the emulator signals.

- To obtain a trace, the analyzer must receive CLK01, and the clock must meet normal clock specifications.

  - Perhaps your target system is degrading the clock so that it does not meet specifications.

  - Perhaps the target system is interfering with proper operation of CLK01.

  Without CLK01, no trace can be taken by the analyzer.

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 the column widths and set options for disassembly of the trace list.

This section covers many of the options available for controlling the trace display. 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 on the command line to obtain a desired trace display. See the **display trace** and **set** command descriptions in Chapter 11, "Emulator/Analyzer Interface Commands", for more information.
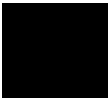
Examples   To use the Trace Options dialog box:

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.
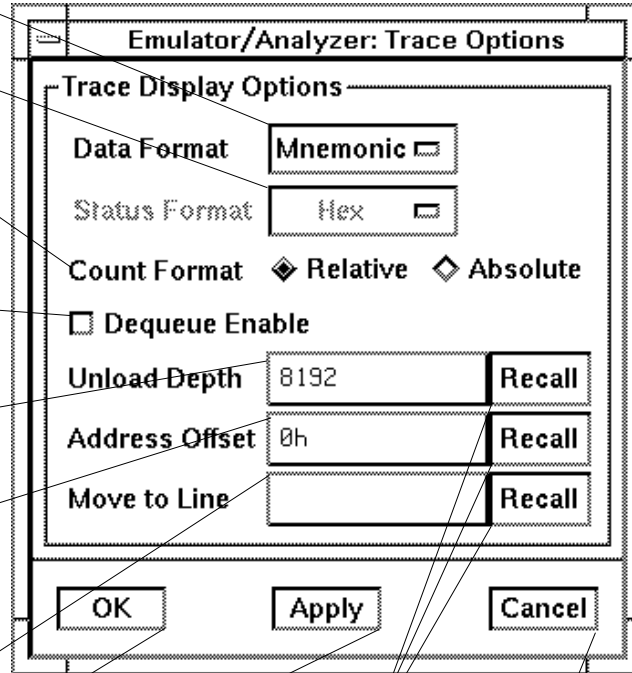
Emulator/Analyzer: Trace Options

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

Click OK   Click Apply   Click these   Click this
to specify   to specify   pushbuttons   pushbutton
the trace   the trace   to select   to cancel
options and   options and   predefined   the entries
close the   leave the   or   and close
dialog box.   dialog box   previously   the dialog
                  open.   specified   box.
                                   entries.
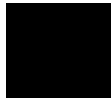
Examples   To use the trace list pop-up menu:

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.

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.

- 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:
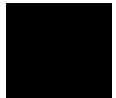
  display trace disassemble_from_line_number <LINE #>

  <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 pulldowns 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.

- 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

  - To start instruction disassembly from the upper word of the bus, enter:

  display trace disassemble_from_line_number <LINE#>
  high_word

  - To start instruction disassembly from the lower word of the bus, enter:

  display trace disassemble_from_line_number <LINE#>
  low_word

Normally, the MC68360 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.

Each analyzer bus state may have two data words. An opcode can appear in either word. You can force disassembly to begin with the lower word of the first trace state by using the **low_word** option. If the disassembled trace list isn't what you expected, try using this option.

The disassembly options remain in effect until you specify a new disassembly option.

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
```

Start instruction disassembly from the upper word of the bus:

```
display trace disassemble_from_line_number 100 high_word
```

Start instruction disassembly from the lower word of the bus

```
display trace disassemble_from_line_number 100 low_word
```

To specify trace dequeueing options

- Choose **Trace→Display Options...** and in the dialog box, select Dequeue Enable. Then click the OK or Apply pushbutton.

- 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`

    - To tell the analyzer which data operand is aligned with the first opcode, enter:

    `display trace disassemble_from_line_number` <LINE#>
    `align_data_from_line` <STATE#>

    <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 also need to use the **low_word** option.)

    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. You

will need to include the **low word** option if the destination opcode is in the low word at the destination address. You may need to resynchronize alignment of operand cycles with the instruction at the branch address, using the **align_data_from_line** option.

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

Tell the analyzer which data operand should be aligned with the first opcode:

display trace disassemble_from_line_number 40
align_data_from_line 42
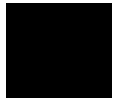
To display the trace without disassembly

- 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.

- 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.

- 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.

- 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.

- 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>

- To set the column width for the symbols column in the trace list, enter:

  set width symbols <WIDTH>

  <WIDTH> is an integer 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.

- 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

The count information in the trace list is always displayed if it 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
deep analyzer, 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."
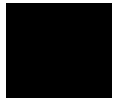
Examples   Count time and store only each iteration of the update_sys symbol in the demo
program (if using the 1K analyzer, make sure the clock speed is set to "Slow" in the
configuration):

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.

- 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 obsolete, 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
```
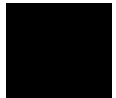
To reset the trace display defaults

- 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.

- Using the command line, enter:

  set default

  This turns off all symbolics and source references in the interface.

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 display the trace list around a specific 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.

- Center the trace display about a particular state given by <LINE #> by entering

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
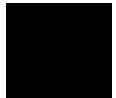
To change the number of states available for display

- 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.

- Using the command line, set the depth of the trace list with:

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 deep analyzer, 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 *MC68360 Emulator/Analyzer Installation/Service/Terminal Interface 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 pulldown 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.

Making Complex Trace Measurements

You can have the analyzer record bus activity by simply using the trace command without any options. But this doesn't use the analyzer effectively for two reasons:

- the trace memory may fill before the program reaches the states of interest.

- you may have to search through a long trace list to find a few states pertinent to your measurement problem.

The HP 64700 analyzer has trigger and sequence capabilities that help solve these problems. These tools act as a filter for processor bus activity that allows the analyzer to capture only the states you want to see in the measurement.

A *trigger* tells the analyzer to identify a certain bus state as a point of reference in the trace of states. A *sequence* is a more complex specification that specifies a series of bus states that must be found to satisfy the trigger.

This section tells you how to get the most out of the HP 64700 analyzer by using trigger and sequence specifications. It also describes additional measurement tools to help you get more information from the trace.

Many of the options in this section can be combined one or more times. See the trace syntax in Chapter 11, "Emulator/Analyzer Interface Commands", for more information.

Expressions are an important part of trace specifications because they specify the numeric or logical values that the analyzer matches for trigger and storage. Expressions are represented by the <expression> symbol in this chapter. Refer to Chapter 11, "Emulator/Analyzer Interface Commands", for specifics on expression syntax.

Expressions in Trace Commands

When modifying the analysis specification, you can enter expressions that 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. Don't care digits are represented by the letters **X** or **x**. A zero must precede any numerical value that begins with an "X". Example: xxx1b must be 0xxx1b.

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-Bus Analyzer Trace Signals

The emulation-bus analyzer has 80 channels available for capturing information: 64 of those channels are used for the instruction bus and data bus, and the remaining 16 channels monitor other processor signals or

Emulation-Bus Analyzer Trace Signals		
Emulation-Bus	**Signal Name (Bits)**	**Signal Description**
Bits 0..31	Address A0-A27	68360 Address Lines A0-A27
	Address A28-A31	68360 Address Lines A28-A31 or reconstructed address
Bits 32..63	Data D0-D31	68360 Data Lines D0-D31
Bits 64..79	Status 0	Reserved
	Status 1..3	CPU function code FC0..FC2, respectively, directly from 68360 processor
	Status 4	R/W directly from 68360 processor
	Status 5..6	SIZ0..SIZ1, respectively
	Status 7..8	DSACK0..DSACK1, respectively (active low)
	Status 9	Bus error (active low)
	Status 10	Halt (active low) on CPU halt line
	Status 11	Show cycle (active low) generated cycle (will occur if show cycles are enabled)
	Status 12	Flush (active low) indicates first fetch following pipeline flush
	Status 13	External DMA cycle indicated when this bit is low
	Status 14	CPU function code FC3
	Status 15	Fetch (active low) indicates an instruction word fetch

synthesized signals, and are collectively called the status lines. You can use status values as trigger or storage qualifiers. For example, you may want to capture processor reads to a certain address, but not processor writes. You can use a status value to qualify only processor read cycles to the memory location.

A number of status values have already been defined for you. They are collectively known as the status equates and cover most common processor operations. Status equates appear on softkeys at the appropriate time so you can include the status you want in your command line.

The following table lists the predefined status values associated with each of the status softkeys.

68360 Status Softkeys and Associated Values

```
Softkey Label   Status Bits (79..64)      Description
---------       --------------------      -----------------------
berr            xxxx xx0x xxxx xxxx       port bus error
cpu             x0xx xxxx xxxx 111x       cpu space access
data            x0xx xxxx xxxx x01x       data cycle
ds_byte         xxxx xxx1 0xxx xxxx       8-bit port
ds_long         xxxx xxx0 0xxx xxxx       32-bit port
ds_word         xxxx xxx0 1xxx xxxx       16-bit port
fc3             x1xx xxxx xxxx xxxx       function code 3
fetch           0xxx xxxx xxxx xxxx       instruction fetch
flush           xxx0 xxxx xxxx xxxx       pipeline flush
halt            xxxx x0xx xxxx xxxx       processor halted
program         x0xx xxxx xxxx x10x       program cycle
read            xxxx xxxx xxx1 xxxx       read cycle
retry           xxxx x00x xxxx xxxx       port not ready
siz_3byt        xxxx xxxx x11x xxxx       3 byte request
siz_byte        xxxx xxxx x01x xxxx       byte request
siz_long        xxxx xxxx x00x xxxx       long request
siz_word        xxxx xxxx x10x xxxx       word request
sup             x0xx xxxx xxxx 1xxx       supervisor cycle
supdata         x0xx xxxx xxxx 101x       supervisor data cycle
supprog         x0xx xxxx xxxx 110x       supervisor program cycle
user            x0xx xxxx xxxx 0xxx       user cycle
userdata        x0xx xxxx xxxx 001x       user data cycle
userprog        x0xx xxxx xxxx 010x       users program cycle
write           xxxx xxxx xxx0 xxxx       write cycle
xdma            xx0x xxxx xxxx xxxx       external DMA cycle
```
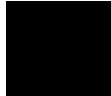
To use address, data, and status values in trace expressions

- Enter the value(s) desired in the entry buffer (such as **address 1000h**). Then Choose **Trace→After(), Trace→Before(),** or **Trace→About(),** as desired.

- 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 Chapter 11, "Emulator/Analyzer Interface Commands", 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.

- 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 *MC68360 Emulator/Analyzer Installation/Service/Terminal Interface 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 use the sequencer

- 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.

- Using the command line, specify a trace sequence by entering:

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 trace sequencer allows you to specify up to seven sequence terms (including the trigger) that must be satisfied to trigger the analyzer. If you use the windowing specification, the sequence specification is limited to four sequence terms.

Example   Use the analyzer sequencer to trigger after finding a series of events:

trace find_sequence main **then** update_sys.get_targets
trigger after proc_spec

To specify a restart term

- 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.

- Using the command line, restart the search for the trace sequence terms by including the restart parameter in

    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.

Example

Use the analyzer sequencer to trace a series of events and then restart the sequencer if the restart term is found while searching for the events:

trace find_sequence update_sys.get_targets **then** update_sys.write_hdwr **restart** update_sys.set_outputs **trigger after** current_humid

To specify trace windowing

- 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.

- 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.

  The trace window specification makes it easy to trace only the occurrences of a particular routine. This is especially useful in high-level languages, where storing only the accesses to a particular address range may miss several function calls within the routine.

Examples   Trace states occurring after the start of the example program:

trace enable main

Trace states occurring between the start of the example program and the call to the message interpreter:

trace enable main ***disable*** proc_spec

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.

- Using the command line, enter commands as follows:

- Specify a trace sequence that starts with a window and ends with a trigger by entering:

trace enable <bus_state> **disable** <bus_state>
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.

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 long_aligned**
update_sys.write_hdwr **trigger after** current_humid

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.

- 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 deep analyzer, 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 MC68360 emulator defaults to **counting off**. To count states or time, you must configure the analyzer clocks correctly. See "To configure the analyzer clock" in Chapter 5, "Configuring the Emulator", 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 configuration):

trace counting 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().**

- Using the command line, store only certain states in the trace list by entering:

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 writen to it. Using a
  prestore measurement, you can find out which module is writing the bad data.

Store-qualify writes to the variable, and use 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 prog and read only**
long_aligned gen_ascii_data

Disable prestore qualification:

trace prestore anything

To trace activity leading up to a program halt

- Choose **Trace→Until Stop**.

- Using the command line, trace on a program halt by entering:

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 modify the trace specification

- Choose **Trace→Trace Spec...** You can recall, modify, and enter your trace specification in the dialog box.

- Using the command line, enter:

  ```
  trace modify_command
  ```

  Then use the command line editing features to change the trace command specifications.

  If you made an error in a trace command or want to change the measurement results slightly, it's often easier to recall the previous trace command and edit it than it is to enter a new trace 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.

  Predefine entries for the Trace Specification Selection dialog box and define the maximum number of entries by setting X resources (refer to Chapter 10, "Setting X Resources").

  See the "To use dialog boxes" description in Chapter 4, "Entering Commands", for information about using selection dialog boxes.

Example   Recall the last trace command with **Trace→Trace Spec...**, or by entering:

`trace modify_command`

Then edit the trace command as you desire.

To repeat the previous trace command

- Choose **Trace→Again.**

- To continually repeat the last trace, choose **Trace→Repetitively.**

- Using the command line, repeat the previous trace command (including its complete trace specification) by entering:

`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 capture a continuous stream of program execution no matter how large your program

The following example can be performed in emulation systems using the deep analyzer (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:

Settings→Pod Command Keyboard
tgout trig1 -c <states before end of memory> (trigger output trig1 before trace complete)
bc -e trig1 (break conditions enabled on trig1)
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:

Settings→Pod Command Keyboard
trng addr=<address range occupied by your monitor> (trigger on range address
= <address range>)
    where <address range> is expressed as <first addr>..<last addr>
tsto !r (trace store not range)
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 command, **Trace→Again**

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).
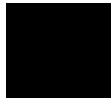
Note that when you set a delay specification using **tgout -c** or **tgout -t** (trigger output delay before trace complete/after trigger), the trace will indicate complete as soon as the analyzer has captured the state specified, even though the entire trace memory has not been filled.

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.

Saving and Restoring Trace Data and Specifications

The emulator/analyzer can save your trace data and trace specifications in a file for later use. This can help you record measurement results that you can use for comparison with other tests, and it is useful 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).

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.

- Using the command line, store the current trace data by entering:

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 into the interface and displayed like any other trace listing.

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.

- Using the command line, load an existing trace specification from a file by entering:

  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
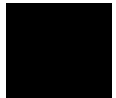
To load trace data

- 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.

- Using the command line, load trace data from a file by entering:

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.

```
 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
```

This section describes how to:

- Set up the trace command for activity measurements.

- Initialize activity performance measurements.

- Interpret activity measurement reports.

To set up the trace command for activity measurements

1 Specify a trace display depth of 512 if using the 1K analyzer. If using the deep analyzer, ignore this step.

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 <RETURN>

To trace after any state, store all states, and count time:

trace counting time <RETURN>

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:

- 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 reentered).

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.
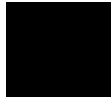
You can also use the **local_symbols_in** option with procedure symbols; this allows you to measure activity related to the symbols defined in a single function or procedure.

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 reentered emulation, you can add traces to a performance measurement simply by entering another **performance_measurement_run** command. However, if you exit and reenter 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 <RETURN>

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 <RETURN>
trace counting time <RETURN>
performance_measurement_initialize addr_ranges <RETURN>
performance_measurement_run 20 <RETURN>
performance_measurement_end <RETURN>
!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
```
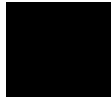
```
              Address Range       101EH thru      10A8H



              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
```
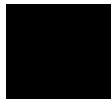
```
          Program Activity
                  State Percent  Rel =   0.00  Abs =   0.00
                                 Mean =   0.00  Sdv =   0.00
                  Time  Percent  Rel =   0.00  Abs =   0.00

 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.

```
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        ⌟
```
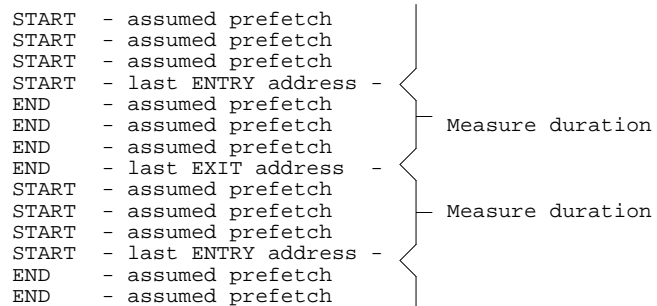
When measuring a recursive function, module duration will be measured between the last recursive call and the true end of the recursive execution. This will affect the accuracy of the measurement.

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 reentered from 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 <RETURN>
```

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*** <RETURN>

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 <RETURN>

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*** <RETURN>

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 reentered).

  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 reentered emulation, you can add traces to a performance measurement simply by entering another **performance_measurement_run** command. However, if you exit and reenter 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 <RETURN>

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 <RETURN>
trace after interrupt_sim start **or** interrupt_sim end
only interrupt_sim start **or** interrupt_sim end **counting
time** <RETURN>
performance_measurement_initialize duration <RETURN>
performance_measurement_run 10 <RETURN>
performance_measurement_end <RETURN>
!perf32 | more

```
        Time Interval Profile


From Address      10A8
        File main(module)."/users/guest/demo/debug_env/hp64749/main.c"
        Symbolic Reference at interrupt_sim+8A
To Address        101E
        File main(module)."/users/guest/demo/debug_env/hp64749/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%
```
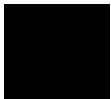
```
From Address      101E
        File main(module)."/users/guest/demo/debug_env/hp64749/main.c"
        Symbolic Reference at main.interrupt_sim
To Address        10A8
        File main(module)."/users/guest/demo/debug_env/hp64749/main.c"
        Symbolic Reference at interrupt_sim+8A
Number of intervals   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 <RETURN>

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 <RETURN>

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 <RETURN>
perf32 -hpf perf1.out <RETURN>
```

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

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 (that is, 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.

CMB Connector

BNC Connector

64700E20
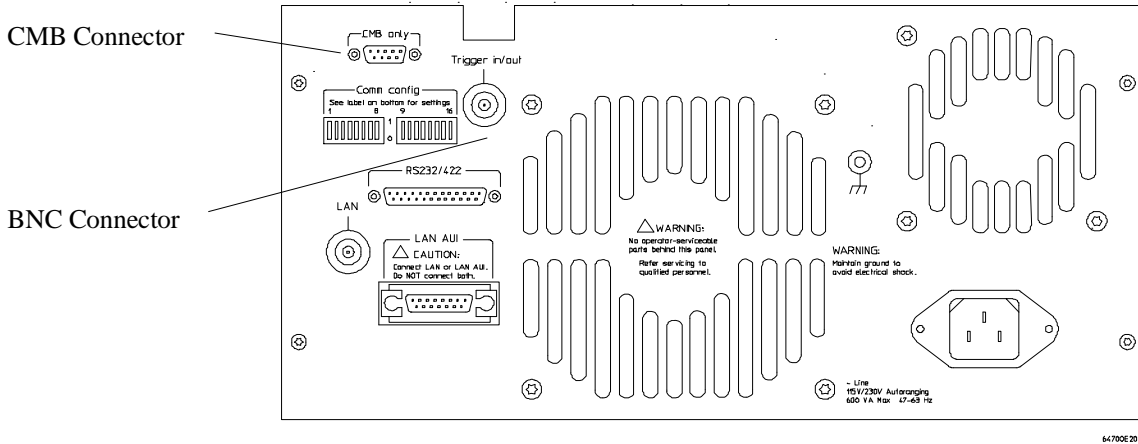
Signal Lines on the CMB

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-232C communications. Applying RS-232C 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

TWO EMULATORS

THREE EMULATORS, ETC.

64700E14
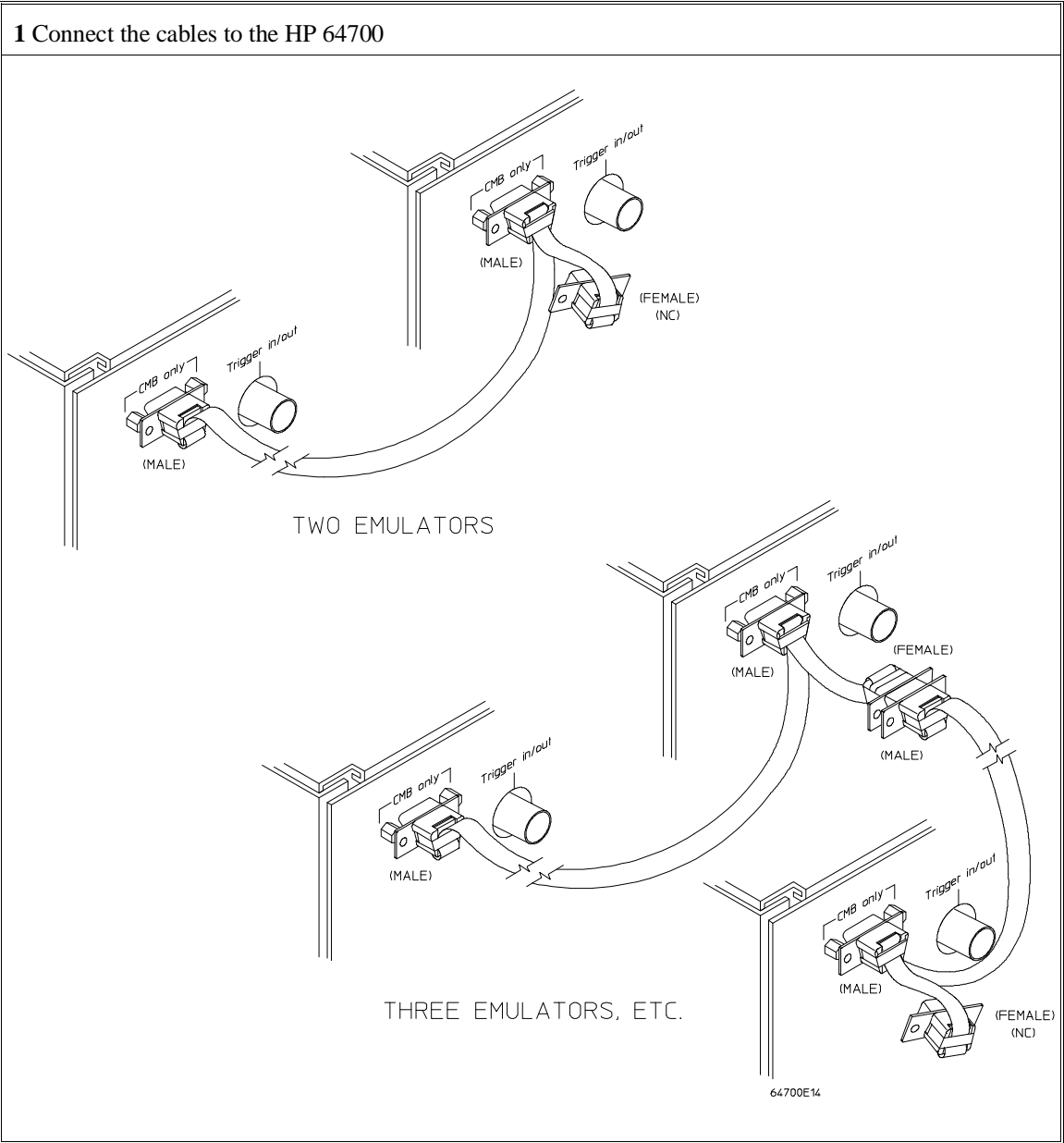
Number of HP 64700 Series Emulators	Maximum Total Length of Cable	Restrictions on the CMB Connection
2 to 8	100 meters	None.
9 to 16	50 meters	None.
9 to 16	100 meters	Only 8 emulators may have rear panel pullups connected. *
17 to 32	50 meters	Only 16 emulators may have rear panel pullups connected. *

* 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
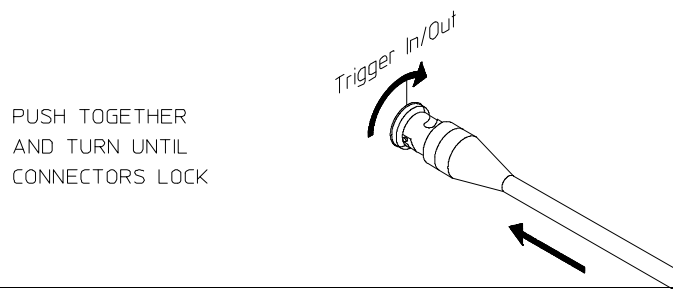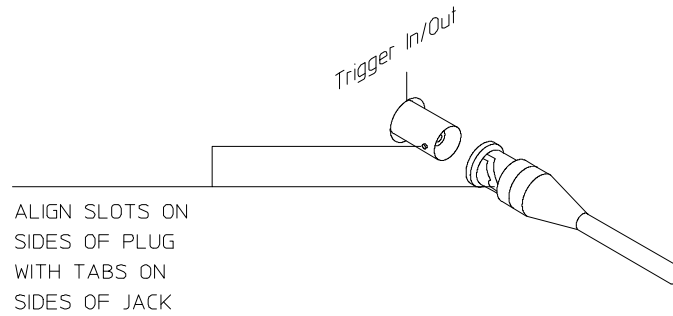
64700E15

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 <RETURN>

To trace when synchronous execution begins:

specify trace after address main <RETURN>

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, 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.
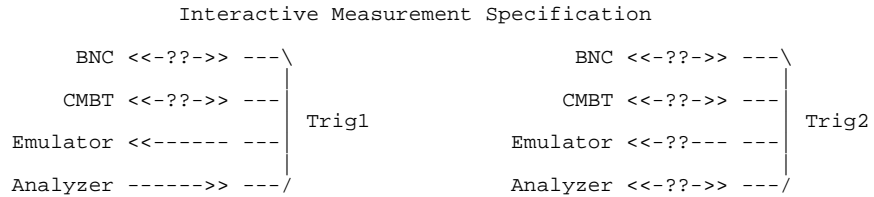
```
                    Interactive Measurement Specification

           BNC <<-??->> ---\                       BNC <<-??->> ---\
                           |                                       |
           CMBT <<-??->> ---|                      CMBT <<-??->> ---|
                           | Trig1                                 | Trig2
       Emulator <<------ ---|                   Emulator <<-??--- ---|
                           |                                       |
       Analyzer ------>> ---/                   Analyzer <<-??->> ---/

NOTES:
  1. The connections marked "??" are set up here in configuration.
  2. drive = ---->>  receive = <<----  (The display won't change, however.)
```

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.

You can use the trig1 or trig2 line to make a connection between the emulator break input and the CMB connector, BNC connector, or 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.

You can use the trig1 and trig2 lines to make several types 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 Chapter 5, "Configuring the Emulator").

```
┌─────────────────────────────────────────────────────┐
│ ═  │ 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 ▭│                  │ │
│  └─────────────────────────────────────────────────┘ │
│                                                       │
│   ┌──────┐        ┌────────┐         ┌──────┐         │
│   │  OK  │        │ Cancel │         │ Help │         │
│   └──────┘        └────────┘         └──────┘         │
└─────────────────────────────────────────────────────┘
```
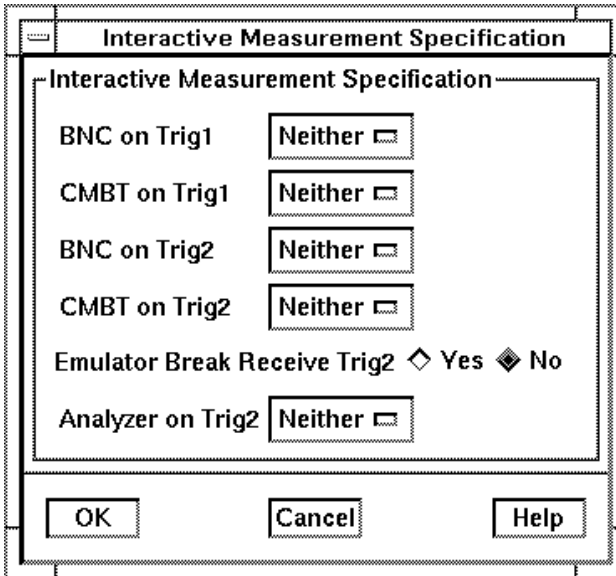
If you're using the Softkey Interface from a terminal or terminal emulation window, you don't get a dialog box from which to choose configuration sections; however, you have access to the same configuration options through a series of configuration questions. To access the questions in the "Interactive 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.

- Break emulator execution on signal from CMB.

- Break emulator execution on signal from BNC.

- Arm the emulation analyzer on signal from CMB.

- Arm the emulation analyzer on signal from BNC.

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 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 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.

Part 3

Reference

Reference

In This Part

This part provides detailed information on aspects of using the Graphical User Interface and the Softkey Interface for the HP 64780A product.

Chapter 10, "Setting X Resources," shows how you can change the appearance or behavior of certain elements in the Graphical User Interface.

Chapter 11, "Emulator/Analyzer Interface Commands," lists and describes each of the commands available in the emulator/analyzer.

Chapter 12, "Emulator Error Messages," lists each of the messages that you may see while using the MC68360 emulator/analyzer, and describes conditions that may cause the message to appear, and suggests actions you can take to correct problems indicated by the messages.

For a thorough analysis of possible problems and solutions, refer to the Hewlett-Packard *M68360 Emulator/Analyzer Installation/Service/Terminal Interface User's Guide.*

Note that Specifications and Characteristics for the Hewlett-Packard MC68360 Emulator/Analyzer are listed in the *M68360 Emulator/Analyzer Installation/Service/Terminal Interface User's Guide.*

10

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.

To modify 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.

<table>
<tr><th colspan="3" align="center">Application Resources for Schemes</th></tr>
<tr><th>Resource</th><th>Values</th><th>Description</th></tr>
<tr>
<td>HP64_Softkey.platformScheme</td>
<td>HP-UX
SunOS
(custom)</td>
<td>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.</td>
</tr>
<tr>
<td>HP64_Softkey.colorScheme</td>
<td>BW
Color
(custom)</td>
<td>Names the color scheme file.</td>
</tr>
<tr>
<td>HP64_Softkey.sizeScheme</td>
<td>Small
Large
(custom)</td>
<td>Names the size scheme file which defines the fonts and the spacing used.</td>
</tr>
<tr>
<td>HP64_Softkey.labelScheme</td>
<td>Label
$LANG
(custom)</td>
<td>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.</td>
</tr>
<tr>
<td>HP64_Softkey.inputScheme</td>
<td>Input
(custom)</td>
<td>Specifies mouse and keyboard operation.</td>
</tr>
</table>

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 Chapter 3, "Starting and Exiting HP 64700 Interfaces".

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 in the
  $HP64000/lib/X11/app-defaults/HP64_Softkey file.

The "actionKeysSub.keyDefs" resource defines a list of paired strings. The first
string defines the text to appear on the action key pushbutton. The second string
defines the command that will be sent to the command line area and executed when
the action key is pressed.

Command files can be executed by placing the name of the command file in the
command definition.

A pair of parentheses with no spaces (that is "()") in the command definition
indicates that text from the entry buffer will 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 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.

An empty action ("") causes the emulator to repeat the previous operation, whether
it came from a pulldown, a dialog, a pop-up, or another action key.

Examples

To set up custom action keys when the graphical interface is used with a 68360
emulator, modify the "actionKeysSub.keyDefs" resource:

```
*m68360*actionKeysSub.keyDefs: \
    "Make"                "cd /users/project2/68360; !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" \
    " Display Special"        "mycommandfilename"
    "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."

Pop-up Recall Buffer Resources		
Recall Pop-up	**Resources**	**Description**
File→Context→Directory ...	*dirSelect.textColumns *dirSelect.listVisibleItemCount *dirSelectSub.entries	The default number of text columns in the pop-up is 50.
File→Context→Symbols ...	*symSelect.textColumns *symSelect.listVisibleItemCount *symSelectSub.entries	The default number of visible lines in the pop-up is 12.
Trace→Trace Spec ...	*modtrace.textColumns *modtrace.listVisibleItemCount *modtraceSub.entries	The "entries" resource is defined as a list of strings (see the following example).
Entry Buffer ():	*recall.textColumns *recall.listVisibleItemCount *recallSub.entries	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").
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 a 68360 emulator, modify the "*m68360*dirSelectSub.entries" resource:

```
*m68360*dirSelectSub.entries: \
    "$HOME" \
    ".." \
    "/users/project1" \
    "/users/project2/68360"
```

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 pulldown 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 pulldown 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/64780A).

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/64780A).

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.

11
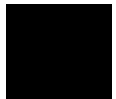
Emulator/Analyzer Interface Commands
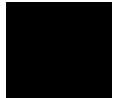
Emulator/Analyzer Interface Commands

This chapter describes the emulator/analyzer interface commands in alphabetical order. First, the syntax conventions are described and the commands are summarized.

How Pulldown Menus Map to the Command Line

The following table shows the items available in the pulldown menus and the command line commands to which they map.

Pulldown	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→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

Pulldown	Command Line
File→Log→Playback	<command file>
File→Log→Record	log_commands to
File→Log→Stop	log_commands off
File→Emul700→High-Level Debugger	N/A
File→Emul700→Performance Analyzer	N/A
File→Emul700→Emulator/Analyzer	N/A
File→Emul700→Timing Analyzer	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>

Pulldown	Command Line
Display→Configuration Info	display configuration_info
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→Upper Address Mode	display configuration_info upper_address
Display→Configuration Info→Clock Input Mode	display configuration_info clock_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

Pulldown	Command Line
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_68360_to_config
Modify→SIM Registers→Copy Emulator SIM to Processor SIM	sync_sim_registers to_68360_from_config
Modify→SIM Registers→Default Emulator SIM	sync_sim_registers default_config
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→from Soft Reset	run from soft_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
	modify software_breakpoints set --EXPR-- temporary
Breakpoints→Temporary ()	modify software_breakpoints set
	modify software_breakpoints clear --EXPR--
Breakpoints→Set All	modify software_breakpoints clear
Breakpoints→Clear ()	
Breakpoints→Clear All	

Pulldown	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)
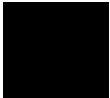
How Pop-up Menus Map to the Command Line

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
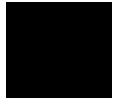
Status Line Pop-up	Command Line
Remove Temporary Message	N/A
Command Line On/Off	(toggles command line)
Display Error Log	display error_log
Display Event Log	display event_log

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)
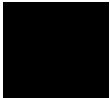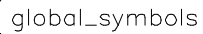
Syntax Conventions

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:

$$\left(\text{global\_symbols}\right)$$

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:

| <REGISTERS> | | --EXPR-- |

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:

$$\left(\,,\,\right)$$

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 interface commands are summarized in the table below and described in the following pages.

!UNIX_COMMAND	display error_log	modify memory[4]
bbaunload	display event_log	modify register[1]
break	display global_symbols	modify software_breakpoints[1]
cd (change directory)[3]	display local_symbols_in	name_of_module[3]
cmb_execute	display memory[4]	performance_measurement_end
<command file>[3]	display pod_command	performance_measurement_init
copy configuration_info	display registers[1]	performance_measurement_run
copy data[4]	display simulated_io[2]	pod_command
copy display	display software_breakpoints	pwd (print working directory)[3]
copy error_log	display status	pws (print working symbol)[3]
copy event_log	display trace	reset
copy global_symbols	end	run
copy help	forward	set
copy local_symbols_in	help[3]	specify
copy memory[4]	load <absolute_file>	step
copy pod_command	load configuration	stop_trace
copy registers[1]	load emul_mem	store memory
copy software_breakpoints	load trace	store trace
copy status	load trace_spec	store trace_spec
copy trace	load user_memory	sync_sim_registers[1]
cws(change working symbol)[3]	log_commands[3]	trace
display configuration_info	modify configuration	wait[3]
display data[4]	modify keyboard_to_simio[2]	

[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.

break

```
( break )─────────────────────────────▶ <RETURN>
```

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.
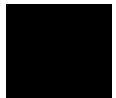
bbaunld

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.

cmb_execute

```
( cmb_execute )————————————▶ <RETURN>
```
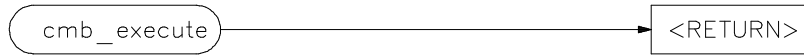
The **cmb_execute** command causes the emulator to emit an EXECUTE pulse on its rear panel Coordinated Measurement Bus (CMB) connector. All emulators connected to the CMB (including the one sending the CMB EXECUTE pulse) and configured to respond to this signal will take part in the measurement.

See Also   The **specify run** and **specify trace** commands.
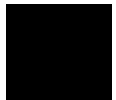
copy
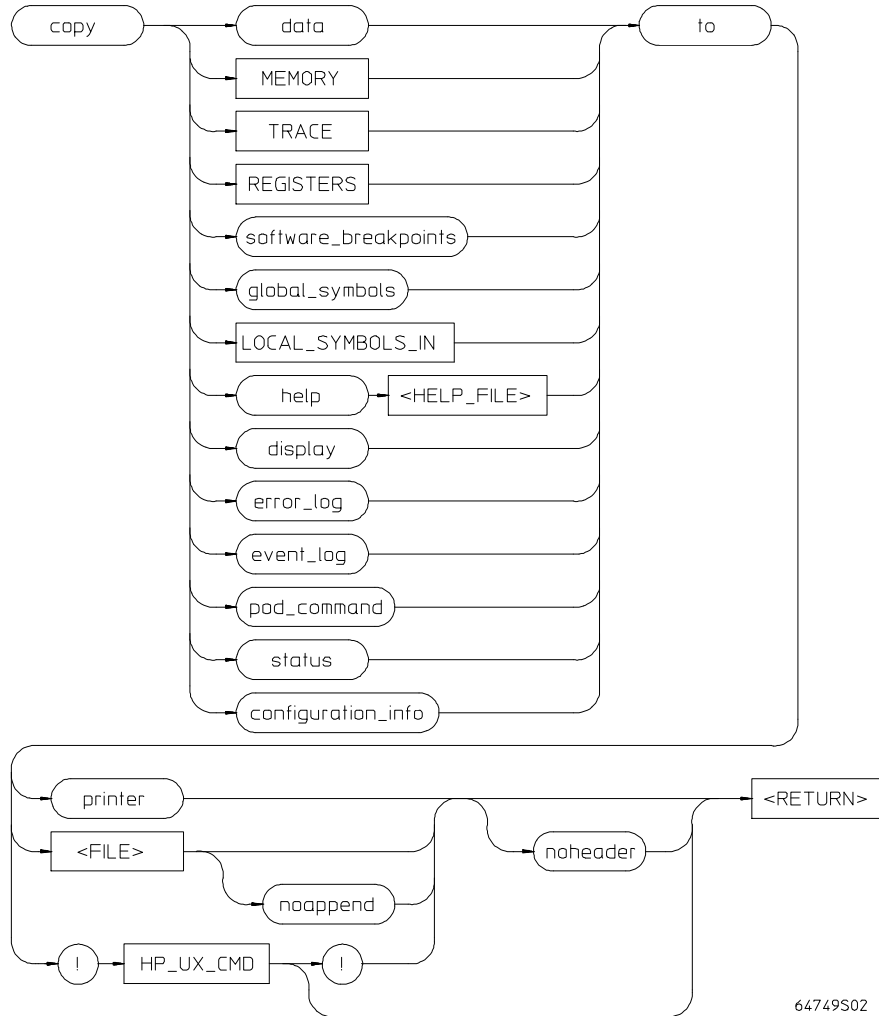
Use this command with various parameters to save or print emulation and analysis information.

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 <RETURN>**
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.

307

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   Copies a list of the current software breakpoints to a selected destination.
_breakpoints

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.

!   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
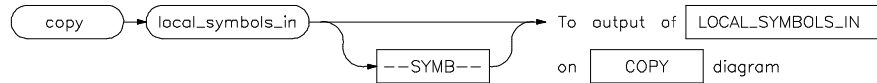
This command lets you copy local symbols contained in a source file and relative segments (program, data, or common) to the selected destination.

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** <RETURN>

    copy local_symbols_in mod_name: **to** linenumfile <RETURN>

See Also   The **display local_symbols_in** command.

copy memory
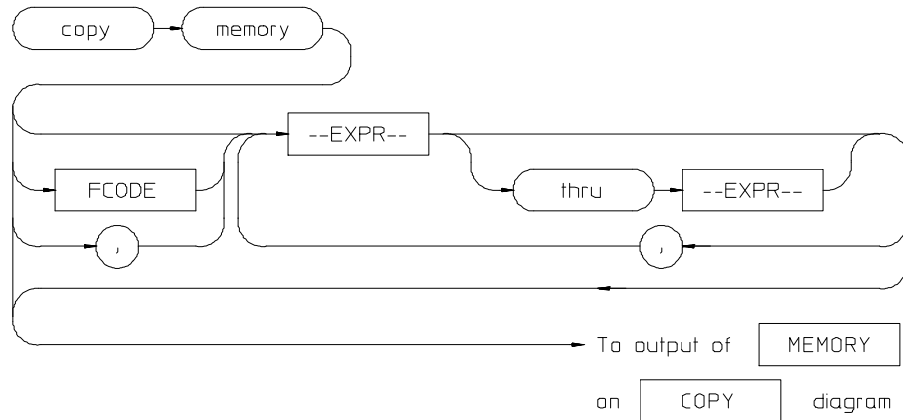
This command copies the contents of a memory location or series of locations to the specified output.

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:

1 The address refers to guarded memory.

2 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* <RETURN>

> *copy memory* 0 *thru* 100h , start *thru* +5 , 500H ,
> target2 *to* memlist <RETURN>

> *copy memory* 2000h *thru* 204fh *to* memlist <RETURN>

See Also   The **display memory**, **modify memory**, and **store memory** commands.
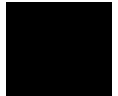
copy registers
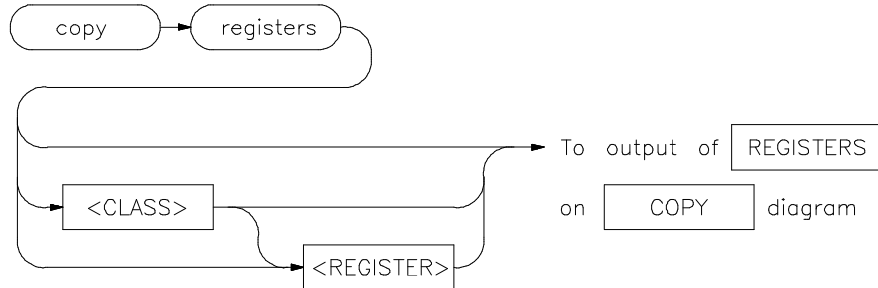
This command copies the contents of the processor registers to a file or printer.

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.

The parameters are as follows:

<CLASS>   Specifies a particular class of the emulator registers.

<REGISTER>

Examples

copy registers EMSIM **to printer** <RETURN>

copy registers to reglist <RETURN>

See Also   The **display registers** and **modify registers** commands.

copy trace
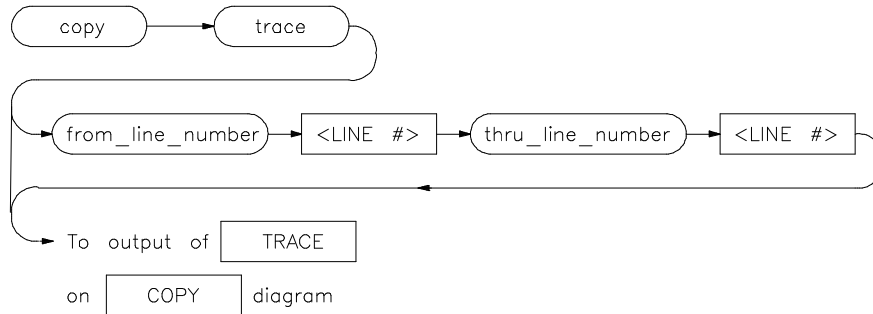
This command copies the contents of the trace buffer to a file or to the printer.

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 <RETURN>

copy trace from_line_number 0 *thru_line_number* 5
to longtrac <RETURN>

See Also   The **display trace** and **store trace** commands.

display
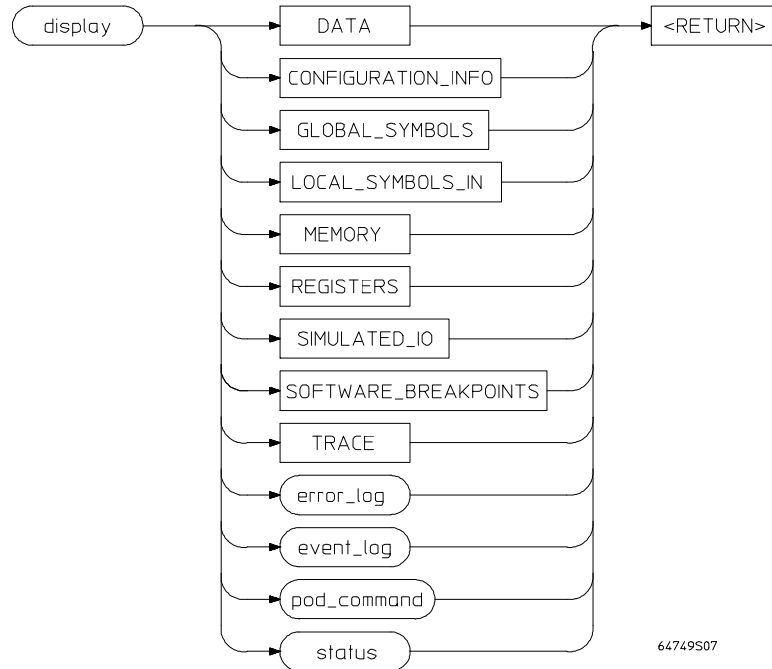
This command displays selected information on your screen.

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.
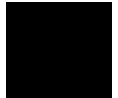
Examples

> *display event_log* <RETURN>

> *display local_symbols_in* mod_name <RETURN>

See Also   The **copy** command description and the following pages which describe the various **display** commands.

display configuration_info

64780S01
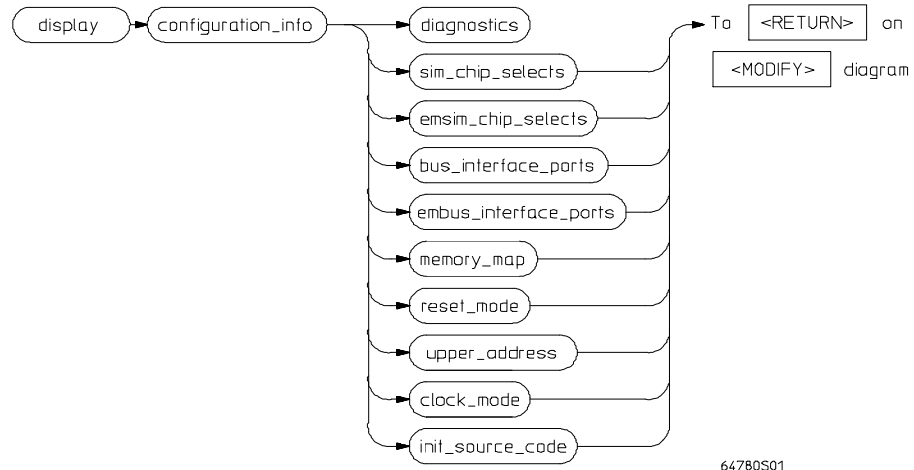
The **display configuration_info** command displays information about emulator configuration and processor SIM programming. You can also display diagnostic information about inconsistencies found in the emulator configuration.

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 values 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 port E.
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 information about the location of 68360 resources.
reset_mode	Displays information about the reset mode configuration value, whether it is generated internally by the emulator or externally by the target system.
upper_address	Display the present address mode, including size of the address bus and whether the upper address bits are used as A31-A28 or WE3-WE0. This display also describes the distribution of address information for the address mode in use.
clock_mode	Display the present mode of clock for the 68360 target system. This mode is set by installation of a clock module in the clock module socket on the emulation probe. Refer to the Hewlett-Packard *MC68360 Installation/Service/Terminal Interface User's Guide* manual for details.
init_source_code	Displays the assembly language program that will initialize the processor as defined by the current EMSIM register contents.
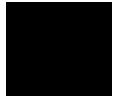
Examples

> `display configuration_info diagnostics` <RETURN>

> `display configuration_info memory_map` <RETURN>

See Also   The **sync_sim_registers** and **modify configuration** commands. Also, see the
"Verifying the Emulator Configuration" section in Chapter 5, "Configuring the
Emulator".

display data

```
display ── data ──
              └─ , ─┘

                                        to  <RETURN>  on
                                        display  diagram

  ──EXPR──                                          byte
        └─ thru ── ──EXPR── ─┘                       word
                                                     long
                                                     int8
                                                     int16
                                                     int32
                                                     u_int8
                                                     u_int16
                                                     u_int32
                                                     char
                                          , 
```
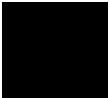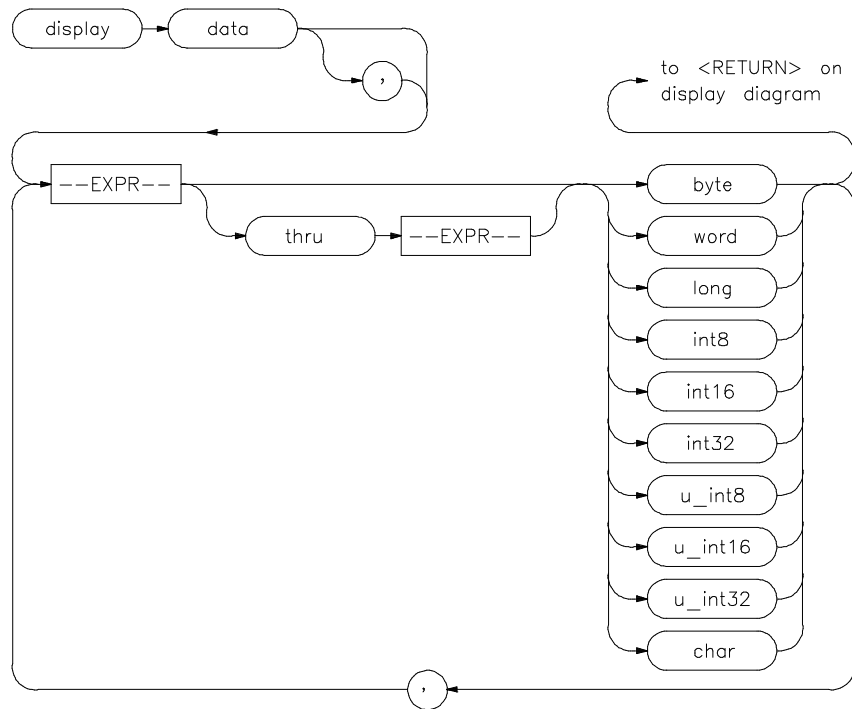
The **display data** command can display the values of simple data types in your program. Using this command can save you time; otherwise, you would need to search through memory displays for the location and value of a particular variable.

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* <RETURN>
>
> *set symbols on* <RETURN>
>
> *set width label* 30 <RETURN>
>
> *display data* , Msg_B *thru* +17 *char*, Msg_Dest *thru* +17
> *char* <RETURN>

See Also   The **copy data** and **set** commands.

display global_symbols
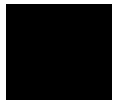
display → global_symbols → To <RETURN> on

DISPLAY diagram
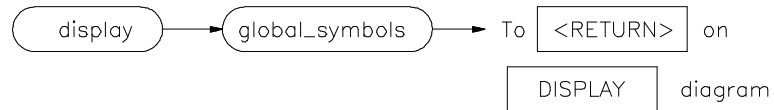
This command displays the global symbols defined for the current absolute file.

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

```
   ( display )──→( local_symbols_in )──┬──────────────────→ To │ <RETURN> │ on
                                        │                       ├──────────┤
                                        └──→┤ ──SYMB── ┤──┘      │ DISPLAY  │ diagram
```
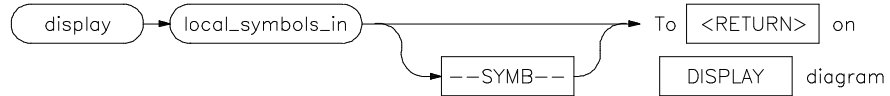
Displays the local symbols in a specified source file and their relative segment (program, data, or common).

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 <RETURN>

display local_symbols_in mod_name:main <RETURN>

See Also   The **copy local_symbols_in** command.
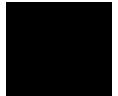
display memory
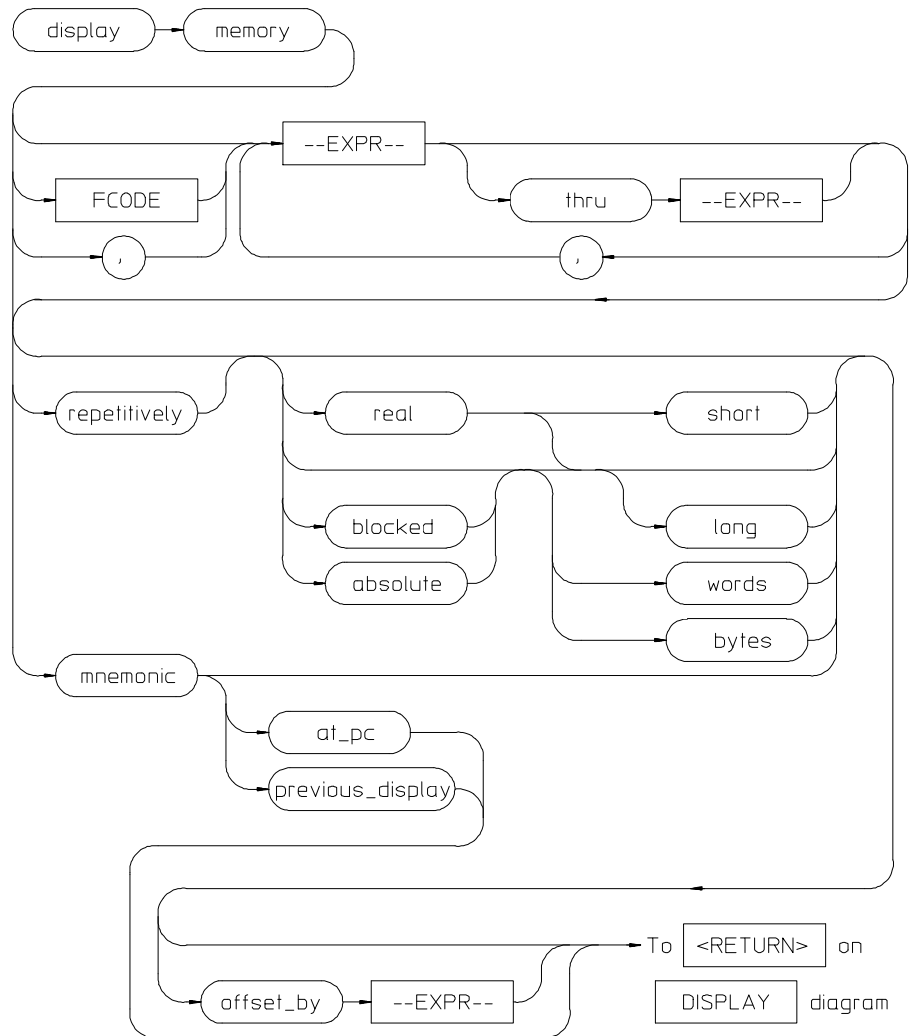
This command displays the contents of the specified memory location or series of locations.

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***
<RETURN>

display memory 400h ***mnemonic*** <RETURN>

set symbols on <RETURN>
set source on <RETURN>
display memory main mnemonic <RETURN>

See Also   The **copy memory**, **modify memory**, **set**, and **store memory** commands.
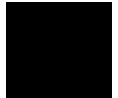
display registers
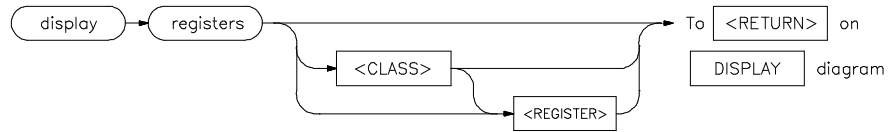
This command displays the current contents of the emulation processor registers.

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 <RETURN>

display registers BASIC D2 <RETURN>

See Also   The **copy registers**, **modify registers**, **set**, and **step** commands.

display simulated_io

display → simulated_io → To | <RETURN> | on

DISPLAY | diagram
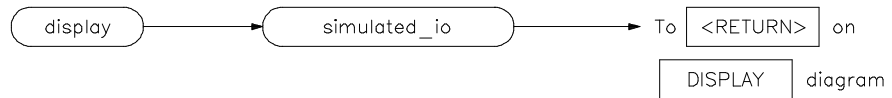
This command displays information written to the simulated I/O display buffer.

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 <RETURN>

See Also   The **modify configuration** and **modify keyboard_to_simio** commands.
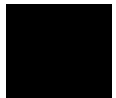
display software_breakpoints
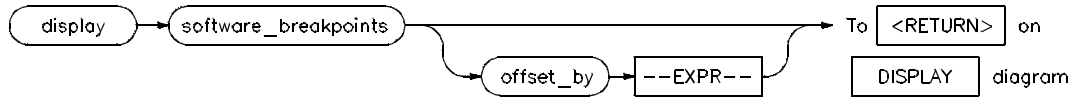
This command displays the currently defined software breakpoints and their status.

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 <RETURN>

display software_breakpoints offset_by 1000H <RETURN>

See Also   The **copy software_breakpoints**, **modify software_breakpoints**, and **set** commands.

display trace

To `<RETURN>` on `DISPLAY` diagram   * available when external labels are in use
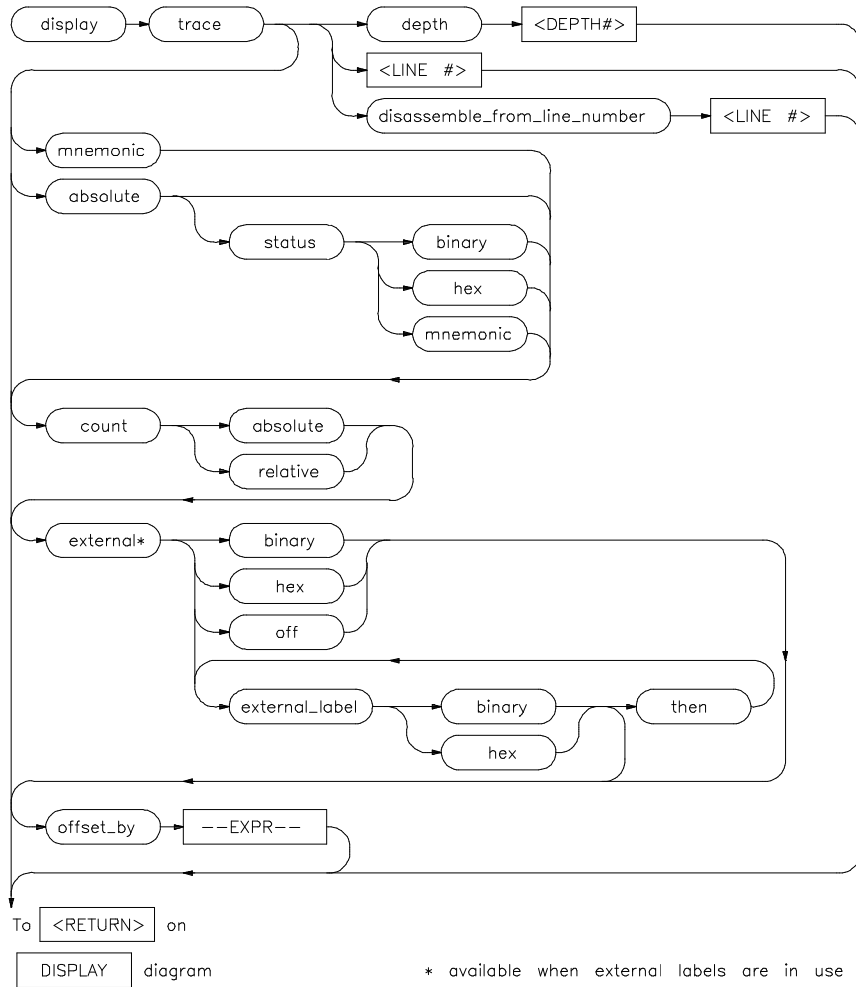
This command displays the contents of the trace buffer.

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
`<RETURN>`

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.
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 <RETURN>

            display trace absolute status binary <RETURN>

            display trace mnemonic <RETURN>

See Also   The **copy trace**, **store trace**, and **set** commands.

end
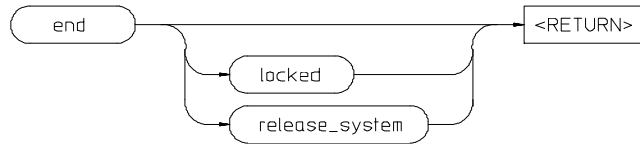
This command terminates the current emulation session.

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 <RETURN>

end release_system <RETURN>

See Also   The "Exiting the Emulator/Analyzer Interface" section in Chapter 3, "Starting and Exiting HP 64700 Interfaces".

--EXPR--
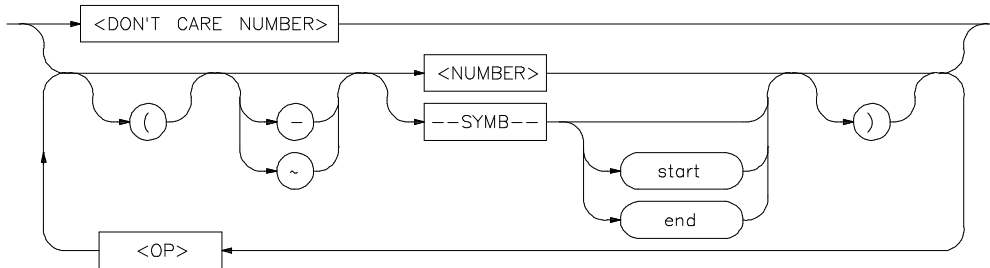
An expression is a combination of numeric values, symbols, operators, and parentheses used to specify address, data, status, executed address, or any other value used in the emulation commands.

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.
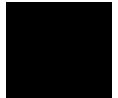
Note that when a **thru** softkey has been entered, a <+ EXPR> prompt appears.
This saves you from tedious repeated entry of long symbols and expressions. For
example:

```
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
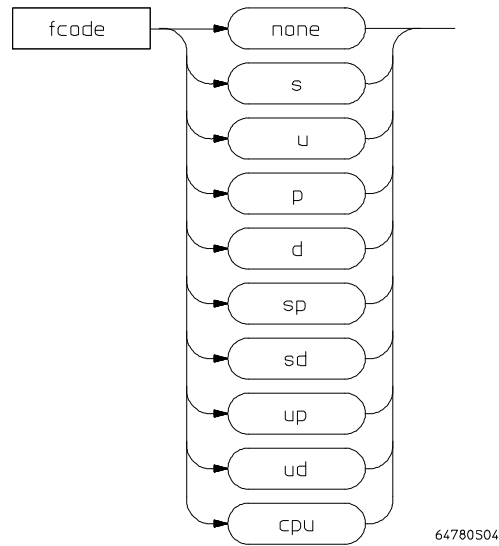
The function code is used to define the address space being referenced. Select the appropriate function code from those listed below.

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.

cpu   Supervisor CPU space.
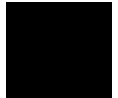
Examples   To copy a portion of user data memory to a file:

copy memory fcode ud 1000H **thru** 1fffH **to** mymem <RETURN>

To modify a location in program memory:

modify memory fcode p 5000h **long to** 12345678h <RETURN>
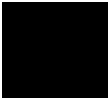
forward
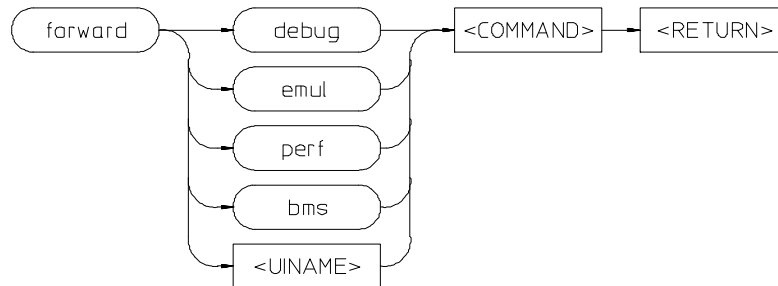
This command lets you forward commands to other HP 64700 interfaces that use the "emul700dmn" daemon process to coordinate actions between the interfaces.

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.

debug   Forwards command to the high-level debugger 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 "Program Run" command to the debugger:

forward debug "Program Run" <RETURN>

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

forward debug "profile" <RETURN>

See Also   The *User's Guide* for the interface to which you are forwarding commands.
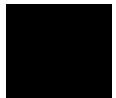
help

```
( help ) ─────▶ <HELP_FILE> ─────▶ <RETURN>
```

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 <RETURN>
```

```
? run <RETURN>
```

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
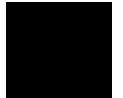
load
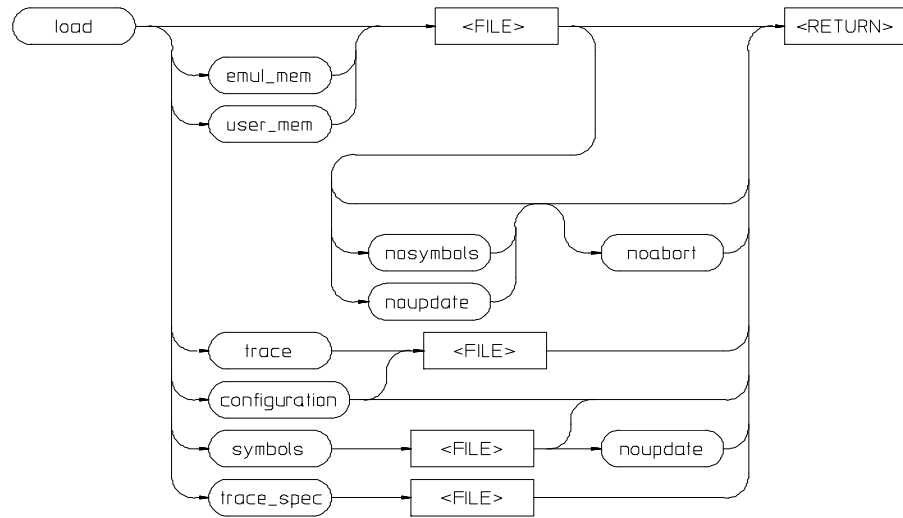
This command transfers absolute files from the host computer into emulation or target system RAM. With other parameters, the load command can load emulator configuration files, trace records, trace specifications, or symbol files.

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 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.

<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).

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 <RETURN>

> *load configuration* config3 <RETURN>

See Also   The **display trace** command.

log_commands

```
┌──────────────┐     ┌──────┐     ┌────────┐                      ┌──────────┐
│ log_commands │────▶│  to  │────▶│ <FILE> │──────────────┬──────▶│ <RETURN> │
└──────────────┘     └──────┘     └────────┘      │       │       └──────────┘
            │                              ┌──────────┐   │
            │                              │ noappend │───┤
            │        ┌──────┐              └──────────┘   │
            └───────▶│ off  │────────────────────────────┘
                     └──────┘
```
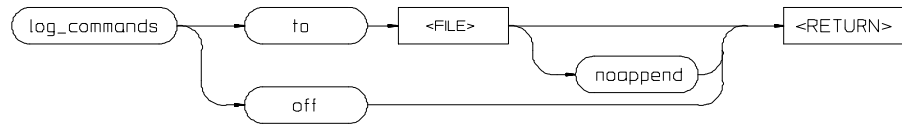
This command allows you to record commands that are executed during an emulation session.

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 <RETURN>

log_commands *off* <RETURN>

See Also   The **wait** command.

346

modify


```
┌─────────┐        ┌──────────────────────┐      ┌──────────┐
(  modify  )───────►│       MEMORY         │─────►│ <RETURN> │
└─────────┘     │  └──────────────────────┘      └──────────┘
                │  ┌──────────────────────┐
                ├─►│      REGISTER        │──────┐
                │  └──────────────────────┘
                │  ┌──────────────────────┐
                ├─►│   CONFIGURATION      │──────┤
                │  └──────────────────────┘
                │  ┌──────────────────────┐
                ├─►│ SOFTWARE_BREAKPOINTS │──────┤
                │  └──────────────────────┘
                │  ┌──────────────────────┐
                └─►│  KEYBOARD_TO_SIMIO   │──────┘
                   └──────────────────────┘           64749S05
```
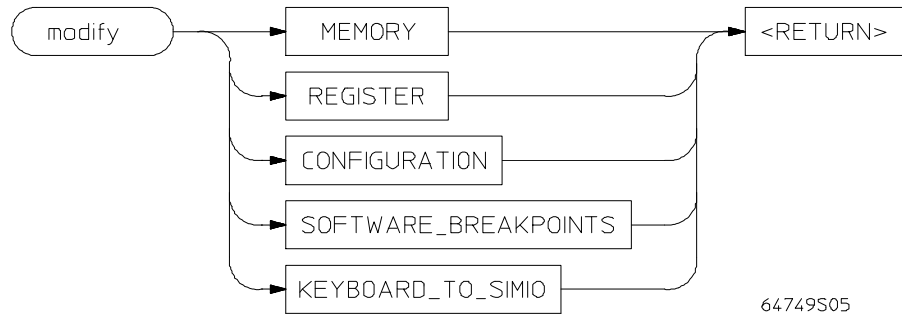
This command allows you to observe or change information specific to the
emulator.

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 software breakpoints table.

The following pages contain detailed information about the various **modify** syntax
diagrams.

modify configuration

```
 ┌──────────┐    ┌──────────────┐    ┌───────────┐
 │  modify  ├───►│configuration ├─►To│ <RETURN>  │
 └──────────┘    └──────────────┘    └───────────┘
                                  ┌───────────┐
                               on │  MODIFY   │ diagram
                                  └───────────┘
```
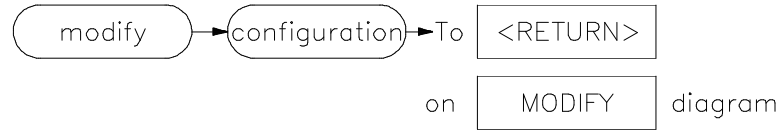
This command allows you to view and edit the current emulation configuration items.

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 <RETURN>

See Also   The **load configuration** command.

modify keyboard_to_simio

```
  ( modify )──►( keyboard_to_simio )──►  To  output  of
                                        ┌─────────────────────┐
                                        │ KEYBOARD _ TO _ SIMIO │
                                        └─────────────────────┘
                                        ┌───────────┐
                                  on    │  MODIFY   │  diagram
                                        └───────────┘
```
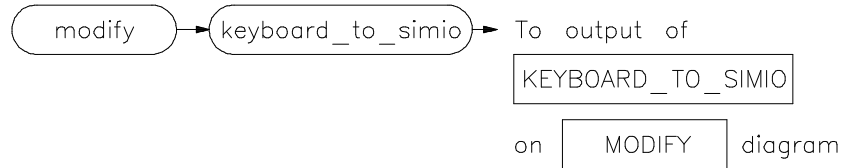
This command allows the keyboard to interact with your program through the simulated I/O software.

When the keyboard is activated for simulated I/O, its normal interaction with emulation is disabled. The emulation softkeys are blank and the softkey labeled "suspend" 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.
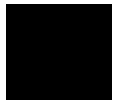
modify memory
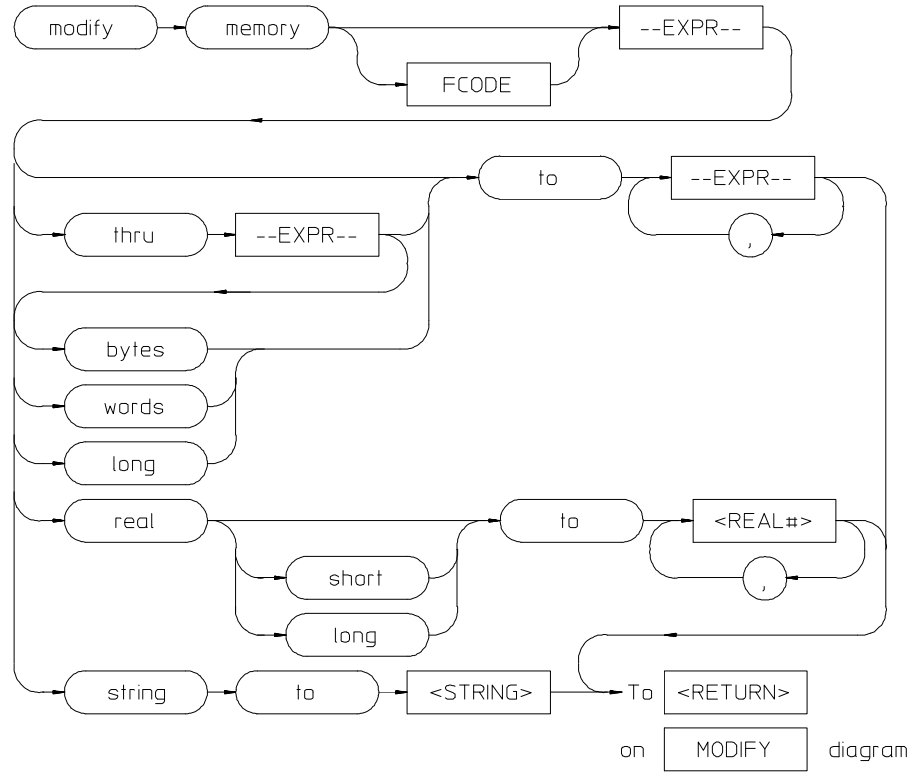
This command lets you modify the contents of selected memory locations.

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>.

351

<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 <RETURN>

modify memory data1 *thru* DATA100 *to* 0FFFFH <RETURN>

modify memory 0675H *real to* -1.303 <RETURN>

modify memory temp *real long to* 0.5532E-8 <RETURN>

modify memory buffer *string to* "Test \n\0" <RETURN>

See Also   The **copy memory**, **display memory**, and **store memory** commands.

modify register
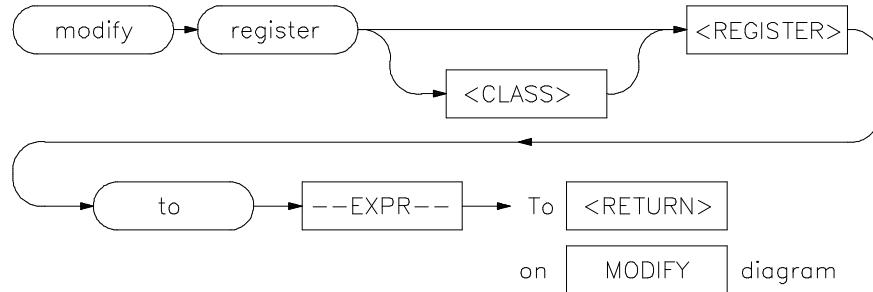
This command allows you to modify the contents of the emulation processor internal registers.

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 <RETURN>

See Also   The **copy registers**, **display registers**, and **modify registers** commands.

modify software_breakpoints
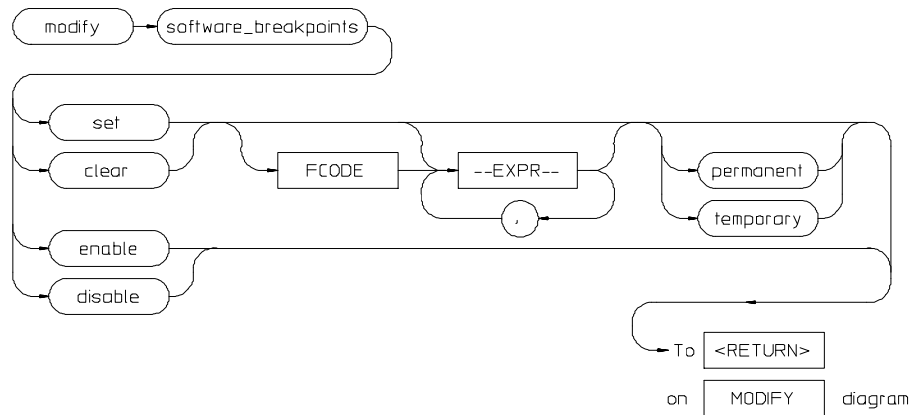
This command changes the specification of software breakpoints.

Software breakpoints provide a way to accurately stop the execution of your program at one or more instruction locations. When a software breakpoint is set, the instruction that is normally at that location is replaced with a TRAP instruction. When the software breakpoint is executed, control is passed to the emulator's monitor program, and the original instruction is restored in the user program. Thus, execution is interrupted before the instruction at the specified address is executed.

Operation of the program can be resumed after the breakpoint is encountered, by specifying either a **run** or **step** command.

If you modify software breakpoints while the memory mnemonic display is active, the new breakpoints are indicated by a "*" in the leftmost column of the instruction containing the breakpoint.

The software breakpoint facility may be completely disabled or enabled via the "modify software_breakpoints" command. The default is "enabled".

The parameters are as follows:

clear	This option erases the specified breakpoint address. If no breakpoints are specified in the command, all currently specified breakpoints are cleared.
disable	This option turns off the software breakpoint capability.
enable	This option allows you to modify the software breakpoint specification.
--EXPR--	An expression is a combination of numeric values, symbols, operators, and parentheses, specifying a software breakpoint address. See the EXPR syntax diagram.
permanent	Sets a permanent breakpoint. The software breakpoint instruction remains in the program until the breakpoint is inactivated or removed.
set	This option allows you to activate software breakpoints in your program. If no breakpoint addresses are specified in the command, all breakpoints that have been inactivated (executed) are reactivated.
temporary	Sets a temporary breakpoint. When the break occurs, the original opcode is replaced in the program.
,	A comma is used as a delimiter between specified breakpoint values.

Examples

modify software_breakpoints enable <RETURN>

modify software_breakpoints set loop1 end , loop2 end , 0E40H <RETURN>

modify software_breakpoints clear <RETURN>

modify software_breakpoints set <RETURN>

See Also   The **copy software_breakpoints**, **display memory mnemonic**, and **display software_breakpoints** commands.

performance_measurement_end

```
( performance_measurement_end )                              ►[ <RETURN> ]
```
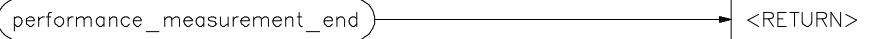
This command stores data previously generated by the
performance_measurement_run command, in a file named "perf.out" in the
current working directory.

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 <RETURN>

See Also   The **performance_measurement_initialize** and
performance_measurement_run commands.

Refer to Chapter 8, "Making Software Performance Measurements" for examples
of performance measurement specification and use.

performance_measurement_initialize
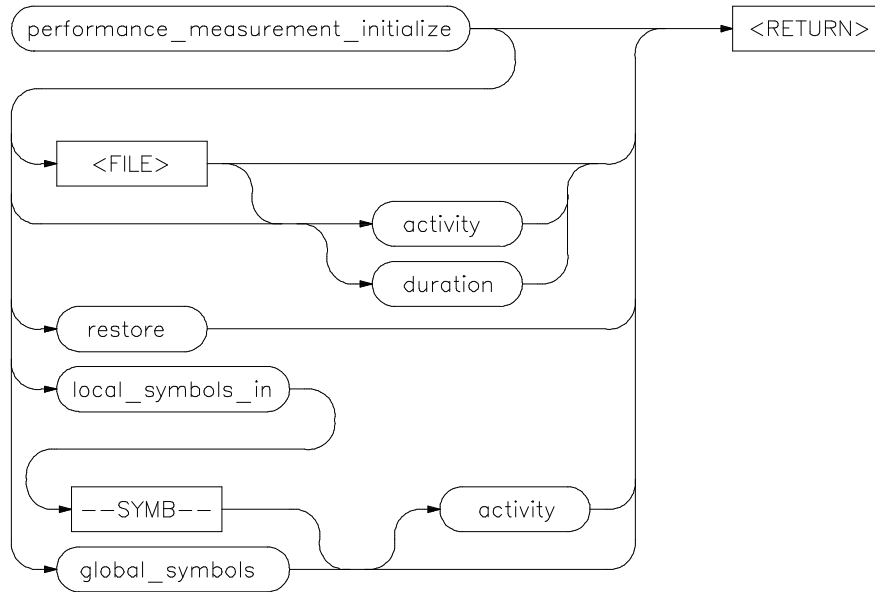
This command sets up performance measurements.

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 <RETURN>

performance_measurement_initialize duration <RETURN>

performance_measurement_initialize local_symbols_in mod_name <RETURN>

See Also

The **performance_measurement_run** and **performance_measurement_end** commands.

Refer to Chapter 8, "Making Software Performance Measurements" for examples of performance measurement specification and use.

performance_measurement_run
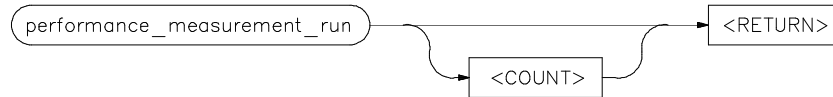
This command begins a performance measurement.

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 <RETURN>**).

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 <RETURN>

performance_measurement_run <RETURN>

See Also   The **performance_measurement_end** and **performance_measurement_initialize** commands.

Refer to Chapter 8, "Making Software Performance Measurements" for examples of performance measurement specification and use.

pod_command
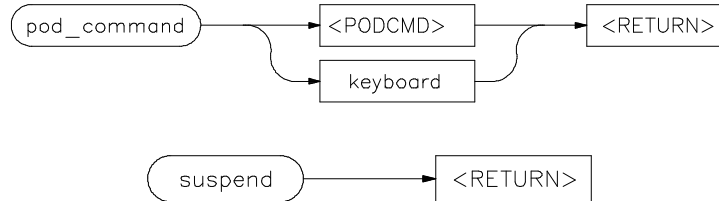
Allows you to control the emulator through the direct HP 64700 Terminal Interface.

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 Hewlett-Packard *M68360 Emulator/Analyzer Installation/Service/Terminal Interface User's Guide* for Terminal Interface use.

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 <RETURN>

pod_command keyboard <RETURN>

cf <RETURN>

tsq <RETURN>

tcq <RETURN>

Enter **suspend** to return to the Graphical User Interface or Softkey Interface.

See Also   The **display pod_command** command.

Also see the Hewlett-Packard *M68360 Emulator/Analyzer Installation/Service/Terminal Interface User's Guide* and the Terminal Interface on-line help information.
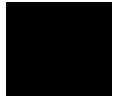
QUALIFIER
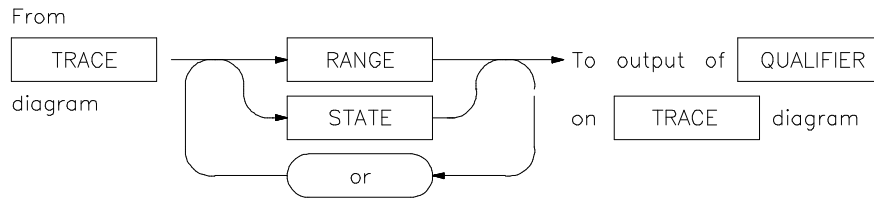
The **QUALIFIER** parameter is used with **trace only**, **trace prestore**, and **TRIGGER** to specify states captured during the trace measurement.

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 <RETURN>

trace only address range mod_name:read_input *thru*
output <RETURN>

trace only address range mod_name:clear *thru* read_input
<RETURN>

See Also   The **trace** command.

RANGE

From
```
RANGE
```
on
```
QUALIFIER
```
diagram

address

data

status

<external_label>

not

range

--EXPR--   thru   --EXPR--   To output of   RANGE

on   QUALIFIER   diagram
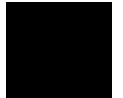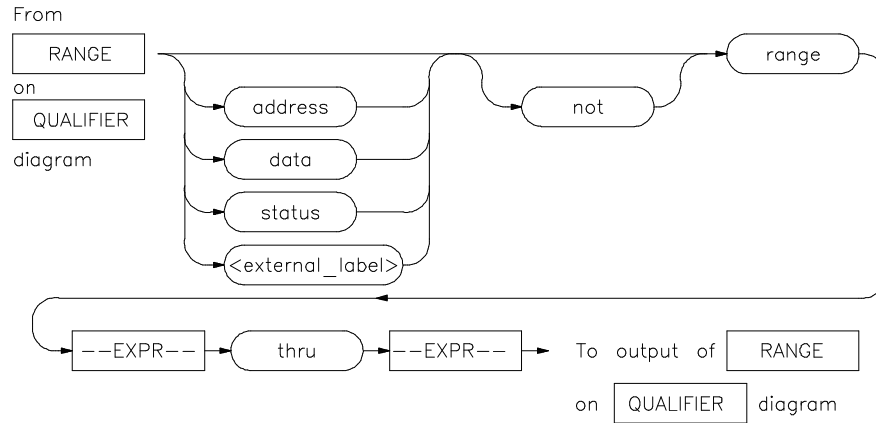
The **RANGE** parameter allows you to specify a condition for the trace measurement, made up of one or more values.

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 Chapter 7, "Using the Emulation-Bus Analyzer" 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.
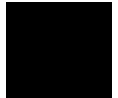
reset

```
  ┌──────────┐        ┌───────────┐
  │   reset  │───────▶│ <RETURN>  │
  └──────────┘        └───────────┘
```
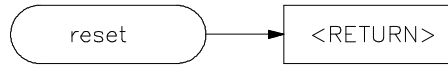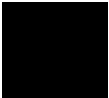
This command suspends target system operation and reestablishes initial emulator operating parameters, such as reloading control registers.

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
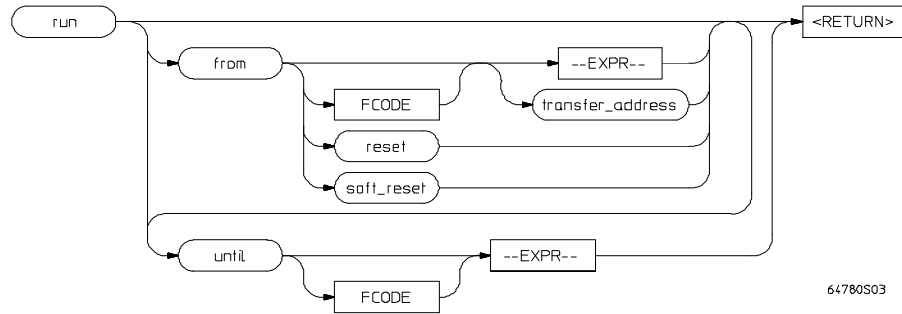
This command causes the emulator to execute a program.

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 is 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.

A **run from soft_reset** command will pulse the 68360 soft reset line to cause a soft reset.

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.

soft_reset
Causes a soft reset.

Examples

run <RETURN>

run from 810H <RETURN>

run from COLD_START <RETURN>

run from transfer_address until 910H <RETURN>

See Also   The **step** command.

SEQUENCING

From trace
syntax diagram

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
                         <RETURN>

See Also   The **trace** command and the QUALIFIER and WINDOW syntax descriptions.
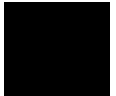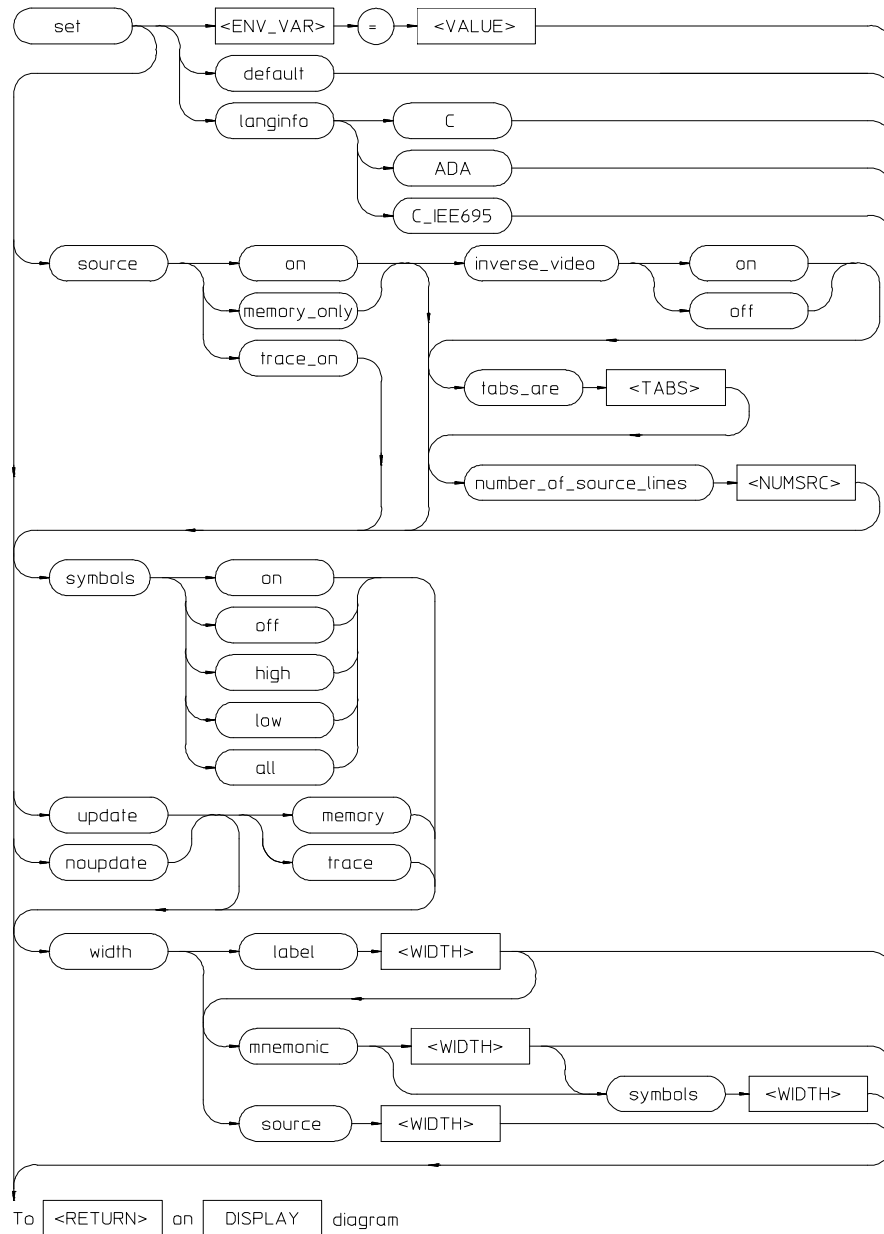
set

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.

Note   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_   This allows you to specify the number of source lines displayed for the actual
source_lines   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

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.

   symbols   This lets you specify the maximum width of symbols in the mnemonic field of the
                trace list, memory mnemonic, and register step mnemonic displays.

   <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 <RETURN>

        set symbols on width label 30 *mnemonic* 20 <RETURN>

        set PRINTER = "lp -s" <RETURN>

        set HP64KSYMBPATH=".file1:proc1
        .file2:proc2:code_block_1" <RETURN>

See Also   The **display data**, **display memory**, **display software_breakpoints**, and **display
                trace** commands.
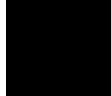
specify
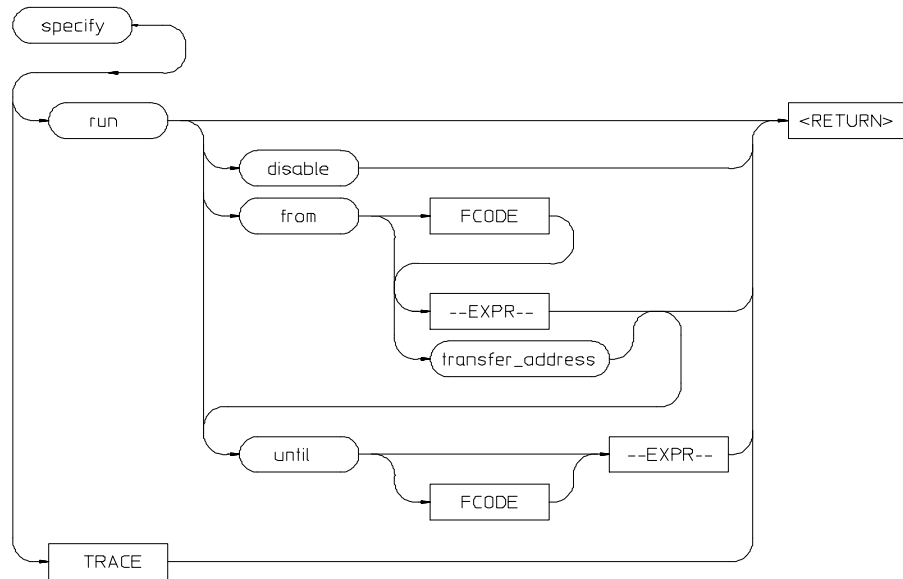
This command prepares a **run** or **trace** command for execution, and is used with the **cmb_execute** command.

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 <RETURN>

    specify trace after address 1234H <RETURN>

See Also   The **cmb_execute** command.

STATE

From

```
STATE    on

QUALIFIER  diagram              To output of   STATE

                                on  QUALIFIER  diagram


                                                --EXPR--

          address                 not

          data

        <external_label>

          status        not        <STATUS>

                                   --EXPR--

                                   <STATUS>

                                   --EXPR--

                         and
```
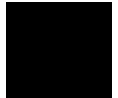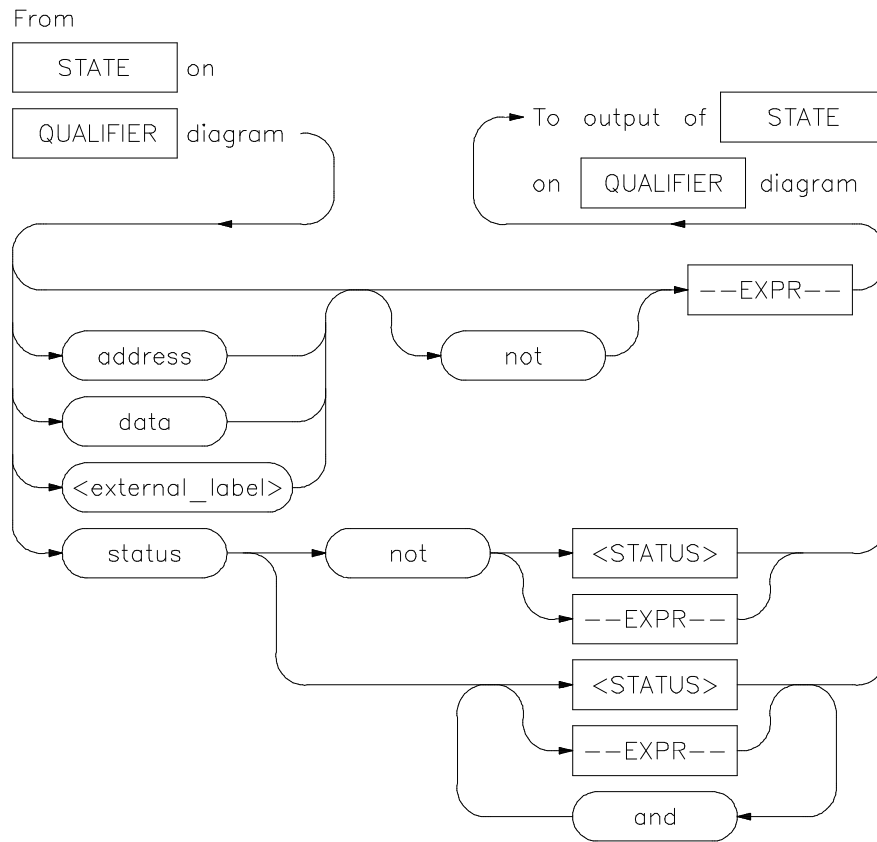
This parameter lets you specify a trigger condition as a unique combination of address, data, status, and executed address values.

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 Chapter 7, "Using the Emulation-Bus Analyzer" 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 <RETURN>

trace about address 1000H status write <RETURN>
```

See the **trace** command examples.

See Also   The **trace** command and the QUALIFIER syntax description.

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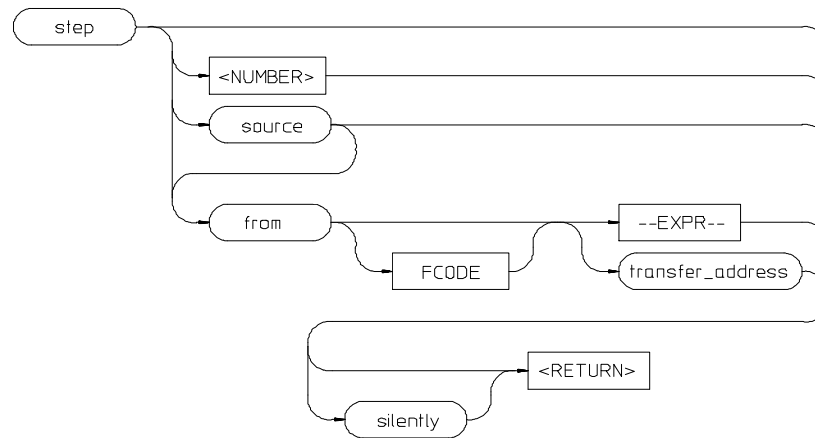
The **step** command allows sequential analysis of program instructions by causing the emulation processor to execute a specified number of assembly instructions or source lines.

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 <RETURN>

step from 810H <RETURN>

step 5 *source* <RETURN>

step 20 *silently* <RETURN>

step 4 *from* main <RETURN>

See Also   The **display registers**, **display memory mnemonic**, and **set symbols** commands.

stop_trace

```
( stop_trace ) ─────────────▶ [ <RETURN> ]
```
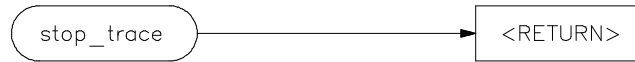
This command terminates the current trace and stops execution of the current measurement.

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
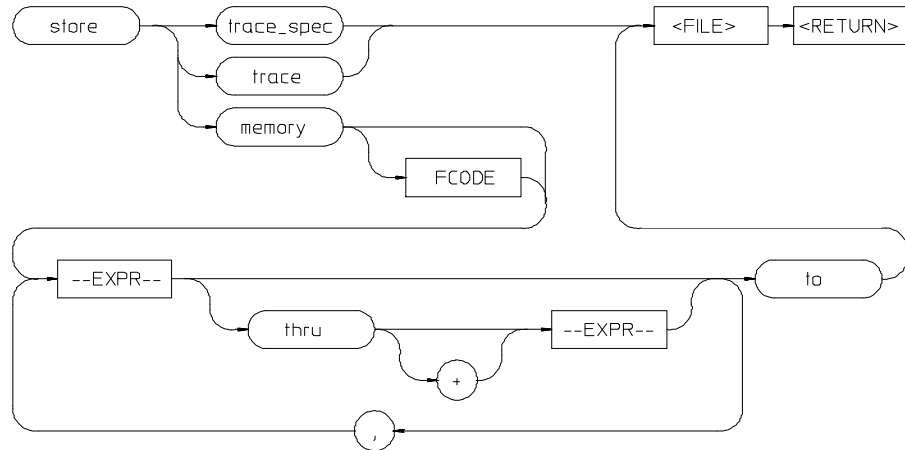
This command lets you save the contents of specific memory locations in an absolute file. You also can save trace memory contents in a trace file.

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.
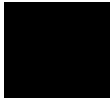
memory	This causes selected memory locations to be stored in the specified HP64000 format file with a **.X** extension.
thru	This allows you to specify that ranges of memory be stored.
to	Use this in the **store memory** command to separate memory locations from the file identifier.
trace	This option 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 <RETURN>
>
> *store memory* EXEC *thru* DONE *to* \12.10 <RETURN>
>
> *store trace* TRACE <RETURN>
>
> *store trace_spec* TRACE <RETURN>

See Also   The **display memory**, **display trace**, and **load** commands.

--SYMB--

--SYMB--

FILE

<SYMB>
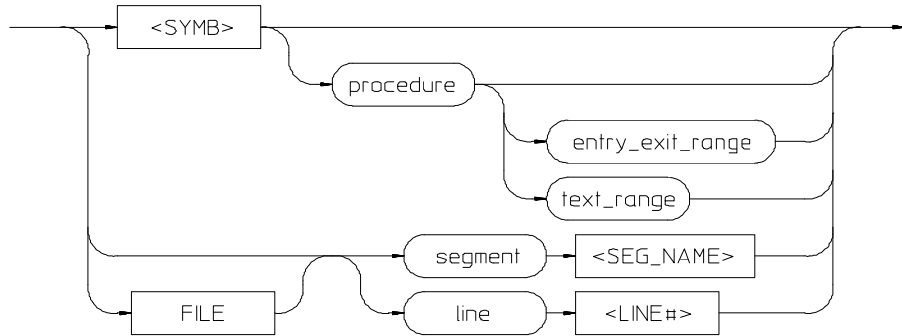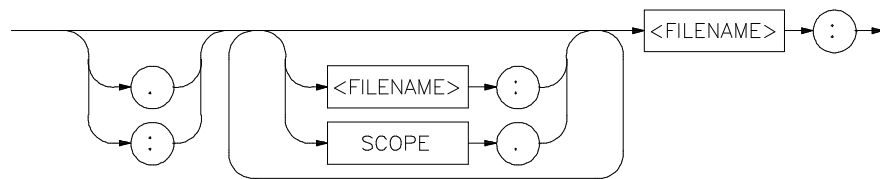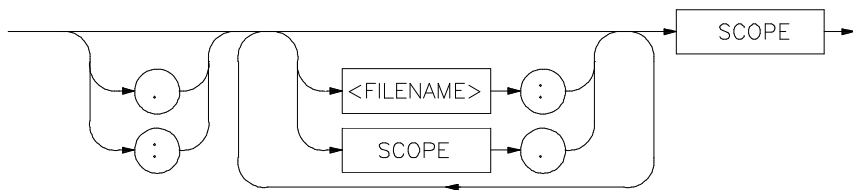
SCOPE
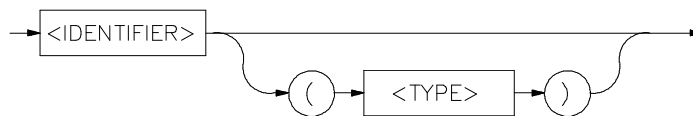
This parameter is a symbolic reference to an address, address range, file, or other value.

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 reenter 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:
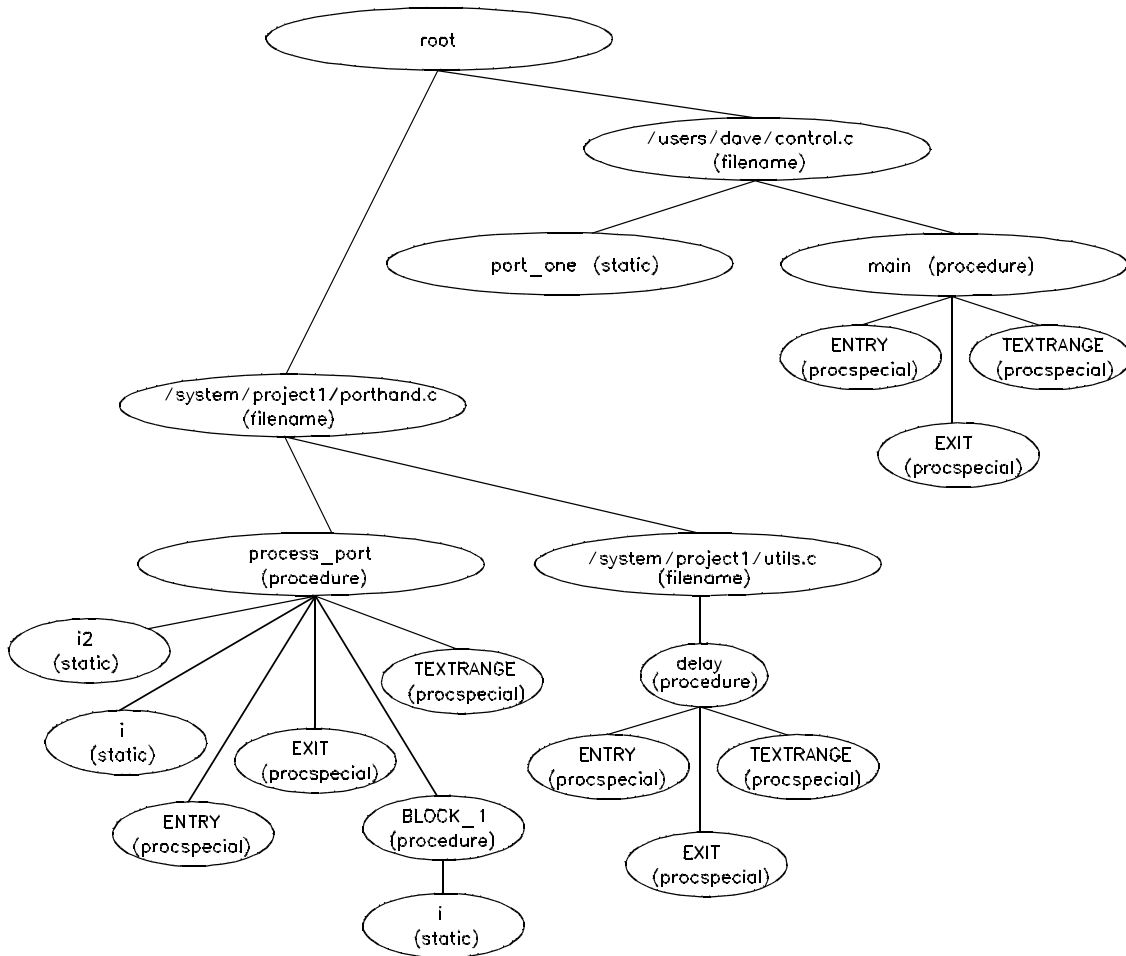
Note that SRU does not build tree nodes for variables that are dynamically allocated on the stack at run-time, such as i and j within the delay () procedure.

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.
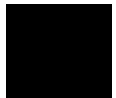
sync_sim_registers
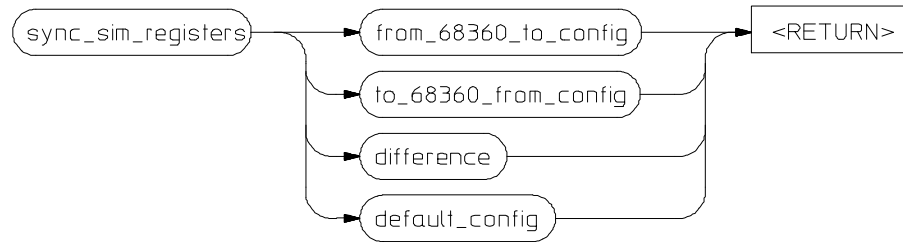
64780S02

The **sync_sim_registers** command synchronizes the 68360's system integration module (SIM) registers to the emulator's EMSIM registers.

The parameters are as follows:

from_68360
_to_config
Copies the microprocessor's SIM registers into the emulator's EMSIM registers.

to_68360
_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.

default_config
Restores the EMSIM registers to their default (power-up) values. This has no effect on the SIM registers.

See Also
The **modify register** commands.

trace
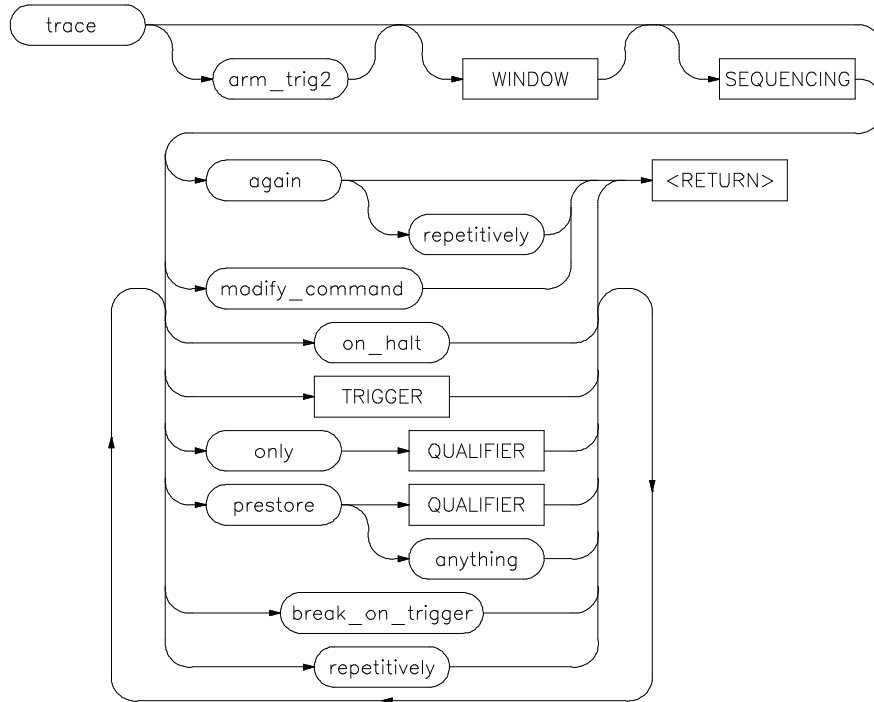
This command allows you to trace program execution using the emulation analyzer.

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 Chapter 9, "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 <RETURN>

> **trace only address range** 1000H **thru** 1004H <RETURN>

> **trace after address** 1000H **occurs** 2 **only address range** 1000H **thru** 1004H **break_on_trigger** <RETURN>

See Also   The **copy trace**, **display trace**, **load trace**, **load trace_spec**, **specify trace**, **store trace**, and **store trace_spec** commands.
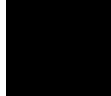
TRIGGER
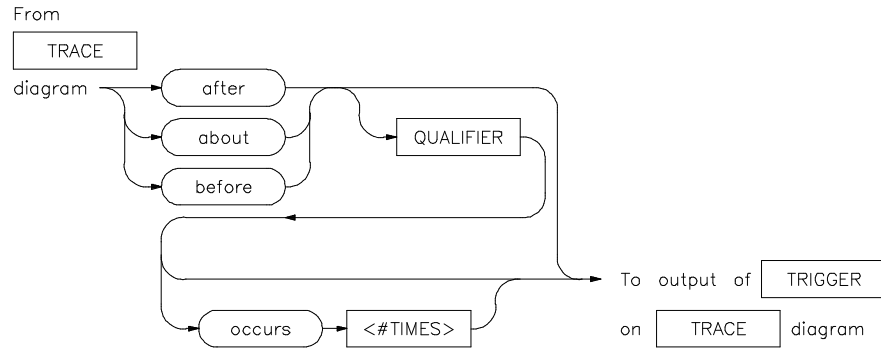
This parameter lets you define where the analyzer will begin tracing program information during a trace measurement.

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 <RETURN>

trace after 1000H *then data* 5 <RETURN>

Also see the **trace** command examples.

See Also   The **trace** command.

Also, refer to Chapter 9, "Making Coordinated Measurements".
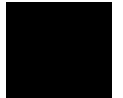
wait
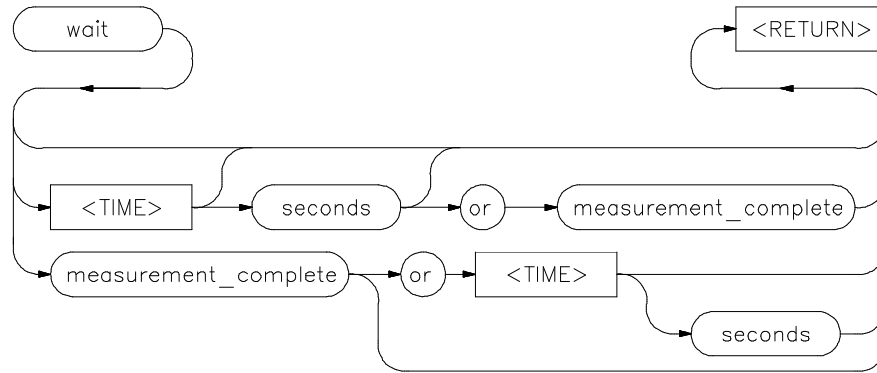
This command allows you to present delays to the system.

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 <RETURN>

wait 5; *wait measurement_complete* <RETURN>
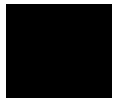
WINDOW

From trace
syntax diagram
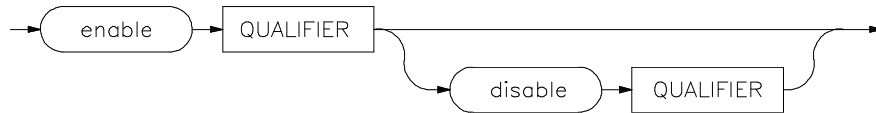
Lets you select which states are stored by the analyzer.

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 <RETURN>

See Also   The **trace** command and the SEQUENCING and QUALIFIER syntax descriptions.

12
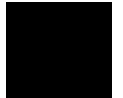
Emulator Error Messages

This chapter lists error and status messages that you may see when using the emulator. The causes of the messages are given along with actions you can take to overcome error conditions.

The emulator/analyzer interface provides feedback to the user through messages that are displayed on the STATUS line.

The messages in this chapter are listed in alphabetical order.

Some messages have error numbers assigned to them. These error numbers are shown in parenthesis at the end of the message text in this chapter.

The error log records error messages received during the emulation session. You may want to display the error log to view the error messages. When several messages are generated for a single error condition, you will have to view the error log to see the complete list of messages. Only the last error message in the sequence will remain in the status line display area.

The error log can hold up to 100 messages. To prevent overrun, the error log purges the oldest messages to make room for the new ones.

Emulator error messages

Analyzer Break (Async_Stat 613)

Cause: Status message. No action necessary.

Analyzer SIMMs are not all the same size; using smallest size (Status 1002)

Cause: Plug-in SIMMs are used to expand the trace depth to 64k or 256k states in the deep analyzer. Four SIMMs, all of the same size must be used. If they are not all the same size, the smallest SIMM size in the set of four will be used for trace depth.

Action: No action necessary.

Arm term used more than once (Error 1250)

Cause: This error occurs when you attempt to use the "arm" qualifier more than once in a sequencer branch expression.

Action: Reenter the trace command and specify the "arm" qualifier only once.

Ascii symbol download failed (Error 881)

Cause: This error occurs because the system is out of memory.

Action: You must either reduce the number of symbols to be loaded, or free up additional system space and try the download again.

Attempt to load code outside of allocated bounds (Error 850)

Cause: This error occurs when you attempt to load an absolute file that contains code or data outside the range allocated for system code.

BDM communication failed (Error 177)

Cause: The emulator aborted your command when a BDM cycle failed.

Action: Try your command again.

BDM cycle aborted due to target reset (Error 173)

Cause: During execution of your command, the emulator BDM cycle was aborted due to detection of a target reset.
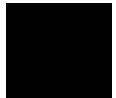
Action: Try your command again.

BNC trigger break (Async_Stat 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 (Error 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. No action is necessary (status only).

Break condition configuration aborted (Error 653)

Cause: Occurs when <CTRL> c is entered during **bc** display.

Break condition must be specified (Error 652)

Cause: You tried to define a breakpoint without specifying the break condition to enable or disable.

Action: Reenter the breakpoint command along with the enable/disable flag and the break condition you wish to modify.

Break due to cause other than step (Error 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.

Breakpoint code already exists: <address> (Error 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: Remove the breakpoints from your program code and try to insert breakpoints again.

Breakpoint disable aborted (Error 671)

Cause: Occurs when <CTRL> c is entered when disabling software breakpoints.

Breakpoint enable aborted (Error 670)

Cause: Occurs when <CTRL> c is entered when setting software breakpoints.

Breakpoint not added: <address> (Error 668)

Cause: The emulator tried to insert a breakpoint in a memory location which could not be accessed.

Action: Insert breakpoints only within memory ranges mapped to emulation or target RAM or ROM.

Breakpoint remove aborted (Error 669)

Cause: Occurs when <CTRL> c is entered when clearing a software breakpoint.

Bus activity required to access emulation memory (Error 148)

Cause: You entered a command that requires the emulator to access emulation memory, but there is no bus activity so the emulator cannot access emulation memory. When the processor is not in the reset state and the bus is not active, which happens during periods like the wait state, the emulator cannot access emulation memory.

Action: Enter a break command. This causes the emulator to begin execution in the monitor program. In this state, the emulator can access emulation memory.

Cannot modify program counter to an odd value

Cause: The emulator will not allow you to modify the content of the program counter to an odd value.
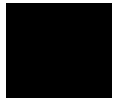
Cannot modify stack pointer to an odd value

Cause: The emulator will not allow you to modify the stack pointer to an odd value.

Can't access module regs, addr space mask (@sd bit 6) is set (Error 168)

Cause: The value of register emmbar or register mbar has bit 6, AS5 (mask supervisor data address space) set to 1. Neither the EMSIM nor the SIM register sets can allow the supervisor data address space to be masked.

Action: Write a new value into register embar or register mbar (as applicable) to set bit 6=0. The EMSIM and SIM register sets cannot function properly when the supervisor data address space is masked by bit 6 being set.

Chip select 0 must have tcyc >=1 for map term 1 (Error 147)

Cause: You tried to enter a TCYC3-0 specification that is incompatible with the present maximum bus speed and memory type. For the type of memory specified in the emulation configuration (cf memtype) and the present bus speed specification (cf maxbusspeed) all chip selects mapped into emulation memory must be programmed for TCYC3-0 >= 1.

For max. bus speed 10MHz: Any chip select into emulation memory may be programmed for any TCYC3-0.

For max. bus speed 25MHz: Any chip select into emulation memory must be programmed for TCYC3-0 >=1.

For max. bus speed 33MHz: Memory type HP64172   Any chip select into emulation memory must be programmed for TCYC3-0 >=1.

Memory type HP64173 or for both type HP64172 and HP64173: Any chip select into emulation memory must be programmed for TCYC3-0 >=2.

For max. bus speed 40MHz: Any chip select into emulation memory must be programmed for TCYC3-0 >=2.

Action: Respecify the value of emor0 to obtain the correct TCYC3-0 programming.

Chip select 0 must have tcyc >=2 for map term 1 (Error 147)

Cause: You tried to enter a TCYC3-0 specification that is incompatible with the present maximum bus speed and memory type. For the memory specified in the emulation configuration (cf memtype) and the present bus speed specification (cf maxbusspeed) all chip selects mapped into emulation memory must be programmed for TCYC3-0 >= 2.

For max. bus speed 10MHz: Any chip select into emulation memory may be programmed for any TCYC3-0.

For max. bus speed 25MHz: Any chip select into emulation memory must be programmed for TCYC3-0 >=1.

For max. bus speed 33MHz, and Memory type HP64172: Any chip select into emulation memory must be programmed for TCYC3-0 >=1.

Memory type HP64173 or for both type HP64172 and HP64173: Any chip select into emulation memory must be programmed for TCYC3-0 >=2.

For max. bus speed 40MHz: Any chip select into emulation memory must be programmed for TCYC3-0 >=2.

Action: Respecify the value of emor0 to obtain the correct TCYC3-0 programming.

Chip 0 has DRAM access into Map Term 1 (not allowed) (Status 169)

Cause: The setup of register emor0 allows DRAM access into emulation mapped memory. The emulator does not allow this access.

Action: Write the correct value into register emor0 to prevent DRAM access into emulation mapped memory.

Chip 0 has Parity enable into Map Term 1 (not allowed) (Status 170)

Cause: The setup of register embr0 enables parity checking into emulation mapped memory. The emulator does not allow parity checking into emulation mapped memory.

Action: Write the correct value into register embr0 to prevent parity checking into emulation mapped memory.
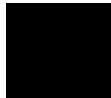
Clock speed not available with current count qualifier (Error 1239)

Cause: This error occurs when you attempt to specify a fast (F) or very fast (VF) maximum qualified clock speed when the analyzer is counting time. This error also occurs when you attempt to specify a very fast (VF) maximum qualified clock speed when the analyzer is counting states.

Action: Change the count qualifier; then reenter the command. See Chapter 7, "Using the Emulation-Bus Analyzer", for more information.

CMB execute break (Error 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. This is a status message; no action is required.

CMB execute; emulation trace started (Error 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 (Async_Stat 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.

CMB trigger break (Async_Stat 617)

Cause: This status message will be displayed if you have configured the emulator to break on a CMB trigger and the CMB trigger line is activated during a program run. The emulator is broken to the monitor.

Command line too complex (Error 814)

Cause: There was not enough memory for the expressions in the command line.

Action: Split up the command line, or use fewer expressions.

Command line too complex (Error 816)

Cause: Too many expression operators are used.

Action: Split up the command line, or use fewer expressions.

Command line too complex (Error 818)

Cause: A maximum nesting level has been exceeded for nested command execution.

Action: Reduce the number of nesting levels.

Command line too long; maximum line length: <number of characters> (Error 813)

Cause: This error occurs when the command line exceeds the maximum number of characters.

Action: Split the command line into two command lines.

Configuration aborted (Error 642)

Cause: Occurs when a <CTRL> c is entered while emulator configuration items are being set.

Configuration failed; setting unknown: <item>=<value> (Error 626)

Cause: Target condition or system failure while trying to change configuration item.

Action: Try to reset. Then reenter your **cf** command. Check target system, and run performance verification (**pv** command).

Conflict between expected and received symbol information (Error 880)

Cause: The information you supplied in a symbol definition is not what the HP 64700 expected to receive.

Action: Make sure that all symbols in the symbol file are defined correctly. Verify that there are no spaces in the address definitions for the symbols in the symbol file being downloaded.

Conflicting disassembler option: <option> (Error 1000)

Cause: This error occurs when you attempt to specify inverse assembly options that are not allowed with each other.

Action: Do not use conflicting inverse assembly options in the same trace list command.

Copy memory aborted; next destination: <address> (Error 752)

Cause: One of these messages is displayed if a break occurs during processing of the **copy memory**, or **modify memory** commands. The break could result from any of the break conditions 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.

Copy target image not supportee (Error 161)

Cause: The cim (copy target image memory) command cannot be used in this emulator. Normally, the cim command would be used to copy a target system memory range to emulation memory so you could set breakpoints or patch code.

Action: To do this without the cim command, save the target system memory range to an absolute file using the copy command. Then remap the target memory range to emulation memory, and load the absolute file into emulation memory using the load command. Refer to Chapter 6, "Using the Emulator", for information on saving and loading absolute files.

Count out of bounds: <number> (Error 318)

Cause: You specified an occurrence count less than 1 or greater than 65535 for a **trace trigger** or **trace find sequence** command.

Action: Reenter the command, specifying a count value from 1 to 65535.

Count qualifier not available with current clock speed (Error 1240)

Cause: This error occurs when you attempt to specify the "time" count qualifier when the current maximum qualified clock speed is fast (F) or very fast (VF). This error also occurs when you attempt to specify a "state" count qualifier when the maximum qualified clock speed is fast (F).

Action: Change the clock speed; then change the count qualifier. See Chapter 7, "Using the Emulation-Bus Analyzer", for more information.

Coverage not supported (Error 160)

Cause: The memory coverage command cannot be used in this emulator because there is no supporting hardware.

Disable breakpoint failed: <address> (Error 604)

Cause: System failure or target condition.

Action: Emulator was unable to write previously saved opcode to target memory. Check target memory system.

Disable breakpoint failed: <address> (Error 666)

Cause: System failure or target condition.

Action: Check memory mapping and configuration questions. This message is usually accompanied by other messages. Look at those messages to better understand the error and know which actions to take.

Display register failed: <register> (Error 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 is likely that the emulator was unable to break to the monitor to perform the register display.

Display truncated to <NUM> lines

Cause: This is a status message. It indicates that the display could not contain all of the information available from the emulator.

Double bus fault occurred (Error 162)

Cause: A double bus fault occurred because of multiple address or bus errors. When this occurs, the emulator breaks into the monitor.

DRAM access into emulation memory is not allowed (Async_Err 164)

Cause: The program has tried to access emulation memory configured as DRAM, which is not supported in emulation memory.

Action: Refer to "If emulation memory addressing appears incorrect" in the "Solving Problems" chapter of the MC68360 Emulator/Analyzer (HP 64780A) Installation/Service/Terminal Interface User's Guide.

Emulation memory access failed (Error 702)

Cause: This message is displayed if the emulator was unable to perform the requested operation on memory mapped to the target system. In most cases, the problem results from the emulator's inability to break to the monitor to perform the operation. Usually there are other error messages. Refer to them to fully understand the cause of the error.

Action: See message "Unable to Break".

Emulator can not respond to chip 0 without ADDR[31:28] (Status 172)

Cause: The emulator is programmed for chip select mode (where the upper address lines are used as WEx[3-0]), but at least one chip select is programed to use ADDR[31-28].

Action: This is a status message. No action required.

Emulator does not support upper address mode selected (Error 144)

Cause: You tried to configure register empepar for WE[3-0] and set the emulator to reconstruct the WEx lines. This mode of emulator operation is not allowed.

Action: Change register empeper to support ADDR[31-28].

Emulator terminated hung bus cycle: 010000000@sp byte read (Error 167)

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, target memory) and make corrections required.

Enable breakpoint failed: <address> (Error 665)

Cause: System failure or target condition.

Action: Check memory mapping and configuration questions. This message is usually accompanied by other messages. Look at those messages to better understand the error and know which actions to take.

Event "expr" cannot be combined with expression definition (Error 1256)

Cause: The terminal interface **tgout** (trigger output) command of the deep analyzer may use an arbitrary expression as an event to drive the trig1 and/or trig2 signals to the emulator. This expression can be set up in two ways. One way uses two **tgout** commands; the first command defines the signals and type of events, and the second command defines the expression. This is most useful when defining complicated expressions. The other way uses one **tgout** command which defines the expression as the event. This error message indicates that you have tried to combine the two methods.

Action: Reenter your **tgout** command using the correct format for the command. Refer to the **tgout** command description in the online help screen available through Pod_Commands for correct formats for the **tgout** command.

Failed to disable step mode (Error 684)

Cause: System failure.

Action: Run performance verification (**pv** command).
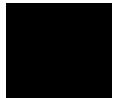
FATAL SYSTEM SOFTWARE ERROR (Error 204)
FATAL SYSTEM SOFTWARE ERROR (Error 205)
FATAL SYSTEM SOFTWARE ERROR (Error 208)

Cause: The system has encountered an error from which it cannot recover.

Action: Write down the sequence of commands that caused the error. Cycle power on the emulator and reenter the commands. If the error repeats, call your local HP Sales Office for assistance.

File could not be opened

Cause: The file cannot be opened or created for writing.

Action: Check to make sure that the parent directory for the file has correct permissions set.

File transfer aborted (Error 410)

Cause: A **transfer** operation was aborted due to a break received, most likely a <CTRL> c from the keyboard. If you typed <CTRL> c, you probably did so because you thought the transfer was about to fail.

Action: Retry the transfer, making sure to use the correct command options. If you are unsuccessful, make sure the data communications parameters are set correctly on the host and on the HP 64700; then retry the operation.

Foreground monitor handled an exception: vector offset 8 (Async_Err 152)

Cause: The foreground monitor handled an exception while performing a monitor command (such as read/write user memory).

Action: Verify memory operation and retry your command.

Foreground monitor not mapped into emulation memory (Error 141)

Cause: You tried to map the foreground monitor into memory space that is not mapped as emulation memory. The foreground monitor must be mapped into emulation memory space.

Action: Respecify map term 1 to be located in emulation memory space.

Foreground monitor range(4Kbytes) not within map term 1 (Error 141)

Cause: You tried to allocate too small a space for map term 1 or you tried to place map term 1 in target memory when you were using a foreground monitor. All of the foreground monitor must be contained in emulation memory.

Action: Respecify map term 1 to contain at least 4 Kbytes and be sure it is located in emulation memory.

Guarded mem break: <guarded memory address> (Async_Stat 628)

Cause: This status message indicates that the target program accessed memory mapped as guarded and the emulator interrupted target execution and began running in the monitor.

Handled target exception: <exception> (Error 628)

Cause: The vector base register points to the exception vector table in the foreground monitor and the target program generated an exception that was caught by the monitor.

Illegal base for count display (Error 1130)

Cause: When specifying the trace format, counts may only be displayed relative or absolute. When counting states, the count is always displayed as a decimal number.

Action: Respecify the trace format without using a base for the count column. Also, you can use "**,A**" to specify that counts be displayed absolute, or you can use "**,R**" to specify that counts be displayed relative.

Illegal base for mnemonic disassembly display (Error 1131)

Cause: When specifying the trace format, you cannot specify a number base for the column containing mnemonic information.

Action: Respecify the trace format without using a base for the mnemonic column.

Illegal base for sequencer display (Error 1132)

Cause: When specifying the trace format, you cannot specify a number base for the column containing sequencer information.

Action: Respecify the trace format without using a base for the sequencer column.

Illegal width for symbol display: <width> (Error 1138)

Cause: This error occurs when the value specified for the trace format address field width is not valid.

Action: Enter your command again, and specify the width of the address field for symbol display within the range of 4 to 55.

Incompatibile signal out events: <Incompatible Event Name> (Error 1254)

Cause: The terminal interface **tgout** (trigger output) command may be used to drive the trig1 and/or trig2 signals to the emulator in response to several different events. The events are trigger recognition, measurement complete, finding a specified expression, delay after trigger recognition, and delay before measurement complete. Some of these events may be ORed together, but a delay specification may not be ORed with trigger recognition or mesaurement complete events.

Action: Examine your **tgout** specification and modify it to remove ORing of delay specifications with trigger recognition or measurement complete events.

Insufficient emulation memory (Error 21)

Cause: You tried to map more emulation memory than is available.

Action: Check your map specification. Do not try to map more emulation memory than is available in your system. You can install up to 2 Mbytes of memory in your system. For a detailed explanation that may explain why you got this message, refer to the message titled, "Request cannot be satisfied with remaining map resources" in this chapter.

Insufficient emulation memory (Error 21)

Cause: You tried to map more emulation memory than is available.

Action: Check your map specification. Do not try to map more emulation memory than is available in your system. You can install up to 2 Mbytes of memory in your system.

Invalid address: <address> (Error 310)

You specified an invalid address value as an argument to a command (other than an analyzer command). For example, you may have specified digits that don't correspond to the base specified, or you forgot to precede a hexadecimal letter digit with a number (even zero (0)).

Action: Reenter the command and the address specification. Use online help by typing **help --EXPR--** and **help --SYMB--**. See the <ADDRESS> and the <EXPRESSION> syntax pages in this manual for information on address specifications.

Invalid address range: <address_range> (Error 311)

Cause: You specified an invalid address range as an argument to a command (other than an analyzer command). For example, you may have specified digits that don't correspond to the base specified, or you forgot to precede a hexadecimal letter digit with a number, or the upper boundary of the range you specified is less than the lower boundary.

Action: Reenter the command and the address specification. Use online help by typing **help --EXPR--** and **help --SYMB--**. See the <ADDRESS> and <EXPRESSION> syntax pages in this manual for information on address specifications. Also, make sure that the upper boundary specification is greater than the lower boundary specification (the lower boundary must always precede the upper boundary on the command line).

Invalid answer in ascii config file; configuration aborted

Cause: A configuration file (filename.EA) being loaded into the emulator has at least one invalid answer to a configuration question.

Action: Display the emulator error log to see which answer(s) were invalid. Edit the configuration file and correct the invalid answer(s) or create a new configuration file by modifying the emulator configuration and storing it.

Invalid attribute for memory type : <attribute> (Error 140)

Cause: You tried to specify a memory attribute for target memory. Attibutes are valid for emulation memory only (memory types eram and erom), not for target memory or guarded memory.

Action: Reenter your specification and use attributes only when specifying emulation memory space.
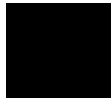
Invalid base: <base> (Error 319)

Cause: This error occurs if you have specified an invalid base when entering a command to change the format of the trace list.

Action: Use the help screens to view the valid base options.

Invalid clock channel: <name> (Error 1207)

Cause: Valid clock channels are L, M, and N.

Action: Respecify the command using valid clock channels.

Invalid command group: <group name> (Error 801)

Cause: This error occurs when you specify an invalid group name in the **help <group>** command.

Action: Enter the **help** command for a listing of the valid group names.

Invalid configuration item: <item> (Error 627)

Cause: You specified a non-existent configuration item.

Action: Use the help screen to see valid items. Reenter the command, specifying only configuration items that are supported by your emulator. Refer to Chapter 5, "Configuring the Emulator", in this manual.

Invalid count: <count> (Error 315)

Cause: This error occurs when the emulation system expects a certain number (of arguments, for example), but you specify a different number.

Action: Enter the number the system expects to receive.

Invalid disassembler option: <option> (Error 1001)

Cause: You specified an invalid option for the disassembler. The disassembler can display all bus cycles, display only instruction cycles, dequeue the trace list, not dequeue the trace list, and disassemble starting with the lower word of the instruction.

Action: Use valid inverse assembly options in your command.

Invalid expression: <expression> (Error 307)

Cause: You have entered an expression with incorrect syntax; therefore, it cannot be evaluated. <expression> is the bad expression.

Action: Use online help. Reenter the expression, following the syntax rules for that type of expression. Refer to Chapter 11, "Emulator/Analyzer Interface Commands", to determine the expression type and the correct syntax for that type.

Invalid map address range: <address range> (Error 723)

Cause: You specified an invalid address range. For example, you may have specified digits that don't correspond to the base specified, or you forgot to precede a hexadecimal letter digit with a number, or the upper boundary of the range you specified is less than the lower boundary.

Action: Reenter your command and the address specification. See the <ADDRESS> and the <EXPRESSION> syntax pages in this manual for information on address specifications. Also, make sure that the upper boundary specification is greater than the lower boundary specification (the lower boundary must always precede the upper boundary on the command line).

Invalid memory map attribute: <attribute> (Error 731)

Cause: The only valid memory map attributes for the MC68360 emulator are Processor DSACKs, Target DSACKs, 32 Bit Emulation DSACKs, 16 Bit Emulation DSACKs, and 8 Bit Emulation DSACKs.

Action: Reenter your command, using only valid memory map attributes.

Invalid memory map type: <type> (Error 730)

Cause: You specified a memory type while mapping that is not one of the supported types: **Emul RAM, Emul ROM, Target RAM, Target ROM, Guarded.**

Action: Reenter your command, specifying only one of the five supported types, listed above.
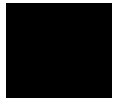
Invalid number of arguments (Error 308)

Cause: You either entered too many options to a command or an insufficient number of options.

Action: Reenter the command with correct syntax. Use online help by typing **help <command>**. Refer to Chapter 11, "Emulator/Analyzer Interface Commands", in this manual for more information.

Invalid occurrence count: <number> (Error 1234)

Cause: Occurrence counts may be from 1 to 65535.

Action: Reenter the command with a valid occurrence count.

Invalid option or operand (Error 300)
Invalid option or operand: <option> (Error 305)

Cause: You have specified an incorrect option to a command. <option>, if printed, indicates the incorrect option.

Action: Use online help by typing **help <command>** or **? <command>**. Reenter the command with the correct syntax. Refer to Chapter 11, "Emulator/Analyzer Interface Commands", for more information.

Invalid pod number: <pod#> (Error 1253)

Cause: This error message occurs when you attempt to specify a slave clock for a non-existent analyzer pod.

Action: Use the trace activity command to display the valid pod numbers, and use only these numbers when entering commands.

Invalid qualifier resource or operator: <expression> (Error 1241)

Cause: When specifying complex expressions, you have either specified an illegal pattern or used an illegal operator.

Action: See Chapter 7, "Using the Emulation-Bus Analyzer", for more information.

Invalid question in ascii file; configuration aborted

Cause: A configuration file (filename.EA) being loaded into the emulator has at least one question that is not valid for this emulator.

Action: Display the emulator error log to see which question(s) were invalid. Edit the configuration file and remove the invalid question(s) or create a new configuration file by modifying the emulator configuration and storing it.

Invalid syntax for global or user symbol name: <symbol> (Error 875)

Cause: This error occurs when you enter a global or user symbol name with incorrect syntax.

Action: Make sure that you enter the global or user symbol name using the correct syntax. When specifying a global symbol, make sure that you precede the global symbol with a colon (for example, **:global_symbol**). When specifying a symbol you created, make sure that you enter the name correctly without a colon.

Invalid syntax for local symbol or module: <symbol/module> (Error 876)

Cause: This error occurs when you enter a local symbol or module name with incorrect syntax.

Action: When entering a local symbol name, make sure you specify the module name, followed by a colon, and then the symbol name (for example **module:local_symbol**). Make sure you specify the module name correctly.

Invalid time: <time> (Error 842)

Cause: You have incorrectly specified the time format in the command.

Action: Reenter the command with the correct time format. See the command syntax pages in this manual for the correct format.

Label not defined: <label> (Error 321)

Cause: You entered an analyzer expression in which the label was not present in the analyzer label list. For example, if the label list includes **address**, **data**, and **status**, you might have entered something such as **lowerdata=24t**. This error also occurs if you try to delete a label that does not exist.

Action: You can reenter the command, using one of the previously defined labels, and adjust the expression as necessary to accommodate the fit of that label to the analyzer input lines. You can also define a new label using the **tlb** command, and then reenter the analyzer command using the newly defined label.

Map range overlaps with term: <term number> (Error 734)

Cause: You entered a map term whose address range overlaps with one already mapped.

Action: Reenter the map term so that ranges do not overlap, or combine terms and change the memory type.

Macro buffer full; macro not added (Error 809)

Cause: This error occurs when the memory reserved for macros is all used up.

Action: You must delete macros to reclaim memory in the macro buffer.

Map term 1 type conflict with foreground monitor, must be eram (Error 141)

Cause: You tried to specify map term 1 as emulation ROM when you were using a foreground monitor. The foreground monitor must be mapped into emulation RAM type memory (not emulation ROM).

Action: Respecify map term 1 to ensure it is RAM memory space in emulation memory.

Maximum argument buffer space exceeded (Error 826)

Cause: You exceeded the space limits for argument lists.

Action: Reenter the command with less arguments, or simplify the expressions in the arguments.

Maximum number of arguments exceeded (Error 824)

Cause: You exceeded the limit of 100 arguments per command.

Action: Reduce the number of arguments in the command.

Memory modify aborted; next address: <address> (Error 754)

Cause: One of these messages is displayed if a break occurs during processing of
the **copy memory**, or **modify memory** commands. The break could result from
any of the break conditions 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 search aborted; next address: <address> (Error 756)

Cause: One of these messages is displayed if a break occurs during processing of
the **copy memory**, or **modify memory** commands. The break could result from
any of the break conditions 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.

Message overflow (Status 153)

Cause: The **display configuration_info diagnostics** command may emit more
messages than the HP 64700 will allow (16). This status message occurs when
there are more than 16 messages.

Missing option or operand (Error 313)

Cause: You have omitted a required option to the command.

Action: Reenter the command with the correct syntax. Use online help by typing
help <command>. Refer to Chapter 11, "Emulator/Analyzer Interface
Commands", in this manual for further information on required syntax.

Modified register(s) caused change in emulator mode (Status 155)

Cause: You changed the value of the empepar register that affects whether the upper address bits will be ADDR[31-28] or WEx[3-0] while the emulator was not in the reset state. The emulator mode of operation is dependent on the value of empepar.

Action: This is a status message. You can reset the emulator and continue operation.

Monitor must be mapped on a 4K byte boundary (Error 146)

Cause: You tried to map the emulation foreground monitor to a base address that is not a 4-Kbyte boundary. The emulation foreground monitor must be mapped on a 4-Kbyte boundary (address XXXXX000H).

Action: Respecify mapping of the emulation foreground monitor to ensure that it is on a 4-Kbyte boundary.

No map terms available; maximum number already defined (Error 7212)

Cause: You tried to add more mapper terms than are available for this emulator. For example, with the MC68360 emulator, there are only eight terms. If you had already defined memory types for these terms, then tried to map another term, you would see the above error message.

Action: Either combine map ranges to conserve on the number of terms or delete mapper terms that aren't needed.

No module specified for local symbol (Error 882)

Cause: This error occurs because you tried to specify a local symbol name without specifying the module name where the symbol is located.

Action: Enter the module name where the local symbol is located, followed by a colon, and then the local symbol name.

Number must be a multiple of 1000H

Cause: A number other than a multiple of 1000H was entered for the base address of the foreground monitor during configuration.

Action: Use a number that is a multiple of 1000H for the base address of the foreground monitor.

One sequence term required (Error 1228)

Cause: This error occurs when you attempt to delete terms from the sequencer when only one term exists.

Action: At least one term must exist in the sequencer. Do not attempt to delete sequence terms when only one exists.

Out of system memory (Error 201)

Cause: Macros and equates that you have defined have used all of the available system memory.

Action: Delete some of the existing macros and equates. This will free additional memory.

Program counter is located in guarded memory (Error 150)

Cause: The address contained in the program counter is an address in guarded memory.

Action: Write a valid address into the program counter.

Program counter is odd (Error 84)

Cause: You attempted to modify the program counter to an odd value using the **modify registers** command on a processor that expects even alignment of opcodes.

Action: Modify the program counter only to even numbered values.
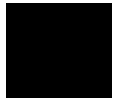
Range resource in use (Error 1221)

Cause: This error occurs when you attempt to redefine the "complex" configuration range resource while it is currently being used as a qualifier in the trace specification.

Action: In the "complex" configuration, display the sequencer specification to see where the range resource is being used and remove it; then, you can redefine the range resource.

Range term used more than once (Error 1248)

Cause: This error occurs when you attempt to use the range resource more than once in a sequencer branch expression.

Action: Do not try to use the range resource more than once in a sequencer branch expression.

Read PC failed during break (Error 603)

Cause: The monitor is not responding.

Action: Check your target system configuration, the emulator configuration and memory map, or reinitialize the emulator. Then try the command sequence again.

Record checksum failure (Error 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> (Error 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 the data communications parameters are set correctly on the host and on the HP 64700. See the *HP 64700-Series Card Cage Installation/Service Guide* for details.

Register access aborted (Error 630)

Cause: Occurs when a <CTRL> c is entered during register display.

Register class cannot be modified: <register class> (Error 637)

Cause: You tried to modify a register class instead of an individual register. You can only modify individual registers.

Action: See the **display** and **modify** syntax pages in Chapter 11, "Emulator/Analyzer Interface Commands", in this manual for a list of register names.

Register emmbar=000000000H; valid bit not set (Error 149)

Cause: You entered a command to copy the values of the EMSIM register set into the target SIM registers. The EMSIM register set can be copied into the processor SIM set only when register emmbar is valid (that is, the valid bit of emmbar, bit 0, must equal 1).

Action: Write a valid content into emmbar and set emmbar bit 0=1. Then try your command again.

Register mbar=000000000H; valid bit not set (Error 149)

Cause: You entered a command that must access the SIM register set. The SIM register set can only be accessed when register mbar is valid (that is, the valid bit of mbar, bit 0, must equal 1).

Action: Write a valid content into mbar and set mbar bit 0=1. Then try your command again.

Request access to guarded memory: <address> (Error 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: Reenter the command and specify only addresses or address ranges within emulation or target RAM or ROM. You can also remap memory so that the desired addresses are no longer mapped as guarded.
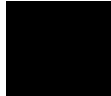
Restricted to real time runs (Error 40)

Cause: While the emulator is restricted to real-time execution, you have attempted to enter a command that requires a temporary break to the monitor for processing (such as a request to display target system memory locations). The emulator will not allow temporary breaks while the emulator is in the reset state or while the target program is running.

Action: Break to the monitor using the **break** command, and then execute the desired command or disable the real time mode.

Retry limit exceeded, transfer failed (Error 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, line noise may cause the failure.

Run failed during CMB execute (Async_Error 694)

Cause: System failure or target condition.

Action: Run performance verification (**pv** command), and check target system.

Sequence term not contiguous: <term> (Error 1225)

Cause: This error occurs when you attempt to insert a sequence term that is not between existing terms or after the last term.

Action: Be sure that the sequence term you enter is either between existing sequence terms or after the last sequence term.

Sequence term not defined: <term> (Error 1227)

Cause: This error occurs when you attempt to delete or specify a primary branch expression for a sequence term number that is possible, but is not currently defined.

Action: Insert the sequence term, and respecify the primary branch expression for that term.

Sequence term number out of range: <term> (Error 1224)

Cause: This error occurs when a sequencer qualification command specifies a non-existent sequence term. The easy configuration sequencer may have a maximum of four sequence terms. Eight sequence terms exist in the complex configuration sequencer.

Action: Reenter the command using an existing sequence term.

Severe error detected, file transfer failed (Error 411)

Cause: An unrecoverable error occurred during a **transfer** operation.

Action: Retry the transfer. If it fails again, make sure the data communications parameters are set correctly on the host and on the HP 64700. Also make sure you are using the correct command options, both on the HP 64700 and on the host.

Software breakpoint: <breakpoint address> (Async_Stat 615)

Cause: This status message indicates that the target program executed a software breakpoint instruction (an execution breakpoint, either in software or provided by one of the eight hardware breakpoint resources). The emulator stopped the target program and began running in the monitor.

Software breakpoint break condition is disabled (Error 661)

Cause: You disabled the software breakpoint feature. Breakpoints are enabled by default. Then you attempted to set a breakpoint, or you attempted to single step with the foreground monitor (either the built-in or custom foreground monitor).

Action: Re-enable the software breakpoint feature and try again.

Specified breakpoint not in list: <address> (Error 663)

You tried to enable a software breakpoint that was not previously defined. <address> prints the address of the breakpoint you attempted to enable. Insert the breakpoint into the table and memory.

Stack pointer is odd (Error 80)

Cause: You tried to modify the stack pointer to an odd value and the emulator expects the stack to be aligned on a word boundary.

Action: Modify the stack pointer to an even value.

Step display failed (Error 688)

Cause: System failure or target condition.

Action: Check memory mapping and configuration questions.

Stepping aborted (Error 685)

Cause: This message is displayed if a break was received during a **step** command with a stepcount of zero (0). The break could have been due to any of the break conditions or a <CTRL> c break.

Stepping aborted; number steps completed: <steps completed> (Error 686)

Cause: This message is displayed if a break was received during a **step** command with a stepcount 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 failed (Error 680)

Cause: Stepping has failed for some reason. For example, this message will appear if the emulator can't modify the trace vector, which is used to implement the step function. Usually, this error message will occur with other error messages.

Action: Refer to the descriptions of the accompanying error messages to find out more about why stepping failed.

Symbol cannot contain text after the wildcard (Error 879)

Cause: You tried to include text after the wildcard specified in the symbol name (for example, **symbol*text**).

Action: Enter the symbol again, but do not include text after the wildcard (*).

Symbol cannot contain wildcard in this context (Error 878)

Cause: You tried to enter a global, local, or user symbol name using the wildcard (*) incorrectly.

Action: When you enter the symbol name again, include the wildcard (*) at the end of the symbol.

Symbol not found: <symbol> (Error 877)

Cause: This occurs when you try to enter a symbol name that doesn't exist.

Action: Enter a valid symbol name.

Target memory access failed (Error 700)

Cause: The emulator was unable to perform the requested operation on memory mapped to the target system. This message is displayed in conjunction with other error messages that further clarify the problem that occurred. In most cases, the problem results from the emulator's inability to break to the monitor to perform the operation.

Action: See other error messages in the error log to further understand the cause of the error.

Target reset and run while in monitor(BDM) (Error 143)

Cause: The emulator had placed the target processor in Background Debug Mode (BDM), and during execution in BDM, the target system was reset.

Action: Reissue your command after target reset.

Trig1, trig2 delay spec out of bounds: <Entered Numeric Value> (Error 1255)

Cause: The terminal interface **tgout** (trigger output) command of the deep analyzer provides a delay feature that allows for driving of the trig1 and/or trig2 signals a specified number of states after trigger or before trace complete. The delay value must be in the range 0 through "current analyzer depth - 1". The current analyzer depth is controlled by the terminal interface command **tcf**. Note: Use of this delay feature may cause modification of the current trigger position value.

Action: Correct the delay value in your specification so that it is within the range of 0 through "current analyzer depth -1".

Trigger position changed to accomodate trig1, trig2 delay spec (Status 1203)

Cause: The terminal interface **tgout** (trigger output) command provides a delay feature that allows for driving of the trig1 and/or trig2 signals a specified number of states after trigger or before trace complete. The setup of this delay feature interacts with the trigger position specification. The trigger position specification may be automatically modified by the deep analyzer in order to make the delay feature work in the expected manner.

Action: You can use the terminal interface command **tp** (trigger position) to examine the new trigger position value.

Trigger term cannot be term 1 (Error 1251)

Cause: This error occurs when you attempt to specify the first sequence term as the trigger term. The trigger term may be any term except the first.

Action: Respecify the trigger term as any other sequence term.

Too many sequence terms (Error 1226)

Cause: This error occurs when you attempt to insert more than four sequence terms.

Action: Do not attempt to insert more than four sequence terms.

Trace error during CMB execute (Error 692)

Cause: System failure.

Action: Run performance verification (**pv** command).

Trace format command failed; using old format (Error 1133)

Cause: This error occurs when the trace format command fails for some reason.

Action: This error message always occurs with another error message. Refer to the description for the other error message displayed.

Trigger position out of bounds: <bounds> (Error 1202)

Cause: This error occurs when you attempt to specify a number of lines to appear either before or after the trigger which is greater than the number of lines allowed. The <bounds> string indicates the incorrect range you typed (not the correct limits on the range).

Action: Be sure that the trigger position specified is within the range -1024 to 1023 (or -512 to 511 if counting is enabled).

trig1 break (Async_Stat 618)

Cause: This status message will be displayed if you used 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.

Trig1 signal cannot be driven and received (Error 1302)

Cause: This error occurs when you attempt to specify the internal trig1 signal as the trace arm condition while the same analyzer's trigger output is currently driving the trig1 signal. This error also occurs if you attempt to specify that the trigger output drive the internal trig1 signal while that signal is currently specified as the arm condition for the same analyzer.

Action: You can either change the arm or the trigger output specification; in either case, make sure they do not use the same internal signal.

trig2 break (Async_Stat 619)

This status message will be displayed if you have used the internal **trig2** line to connect the analyzer trigger output to the emulator break input and the analyzer has found the trigger condition. The emulator is broken to the monitor.

Trig2 signal cannot be driven and received (Error 1303)

Cause: This error occurs when you attempt to specify the internal trig2 signal as the trace arm condition while the same analyzer's trigger output is currently driving the trig2 signal. This error also occurs if you attempt to specify that the trigger output drive the internal trig2 signal while that signal is currently specified as the arm condition for the same analyzer.

Action: You can either change the arm or the trigger output specification; in either case, make sure they do not use the same internal signal.
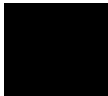
Unable to modify trace vector to <value> for single stepping (Error 156)

Cause: You tried to single step, and the emulator detected the trace vector was not set properly and the emulator was unable to modify the vector table because it was not located in emulation memory or target RAM. This usually occurs when the vector table is located in target ROM.

Action: Copy or relocate the vector table in emulation memory or target RAM, or change your ROM image so that it contains the proper value for the trace vector for single stepping. Refer to stepping information in Chapter 6, "Using the Emulator".

Unable to break (Error 608)

Cause: This message is normally used with other messages that further describe the error. It is displayed if the emulator is unable to break to the monitor because the emulation processor is reset, halted, or the monitor is not responding for some reason.

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 target 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 delete label; used by emulation analyzer: <label> (Error 1105)

Cause: This error occurs when you attempt to delete an emulation trace label that is currently being used as a qualifier in the emulation trace specification or is currently specified in the emulation trace format.

Action: Display the emulation trace sequencer specification in the configuration, display the emulation trace patterns in the complex configuration, or display the trace format to see where the label is used. Also, you should check **tcq** and **tpq** for uses of that label. You must change the pattern or format specification to remove the label before you can delete it.

Unable to load new memory map; old map reloaded (Error 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> (Error 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 is likely that the emulator was unable to break to the monitor to perform the register modification.

Unable to read registers in class: <name> (Error 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.

Unable to redefine label; used by emulation analyzer: <label> (Error 1108)

Cause: This error occurs when you attempt to redefine an emulation trace label that is currently used as a qualifier in the emulation trace specification.

Action: Display the emulation trace sequencer specification in the easy configuration, display the emulation trace patterns in the complex configuration, or display the emulation trace format to see where the label is used. You must change the pattern or format specification to remove the label before you can redefine it.

Unable to reload old memory map; hardware state unknown (Error 726)

Cause: Error occurred while trying to modify the emulation memory map.

Action: Usually there are other error messages present. Refer to their descriptions to more fully understand the cause and action to take for this error.

Unable to reset (Error 640)

Cause: Target condition or system failure.

Action: Check target system, and run performance verification (**pv** command).

Unable to run (Error 610)

Cause: Run has failed for some reason. For example, this message will appear if the emulator cannot write to stack, which is required to run. Usually, this error message will occur with other error messages.

Action: Refer to the descriptions of the accompanying error messages to find out more information about why the run failed. Look at the emulator prompt to know the emulator status. Take a trace with the analyzer to see where the emulator is executing.

Unable to run after CMB break (Error 606)

Cause: System failure or target condition.

Action: Run performance verification (**pv** command), and check target system.

Unexpected software breakpoint (Error 620)
Unexpected step break (Error 621)

Cause: System failure.

Action: Run performance verification (**pv** command).

Undefined software breakpoint: <address> (Error 605)

Cause: The emulator has encountered a BKPT instruction in your program that was not inserted with the **breakpoint** command.

Action: Remove the breakpoints inserted in your code before assembly and link, and then reinsert them using the **breakpoint** command.

Undefined software breakpoint: <breakpoint address> (Async_Stat 605)

Cause: This status message indicates a breakpoint instruction was executed and the emulator stopped target execution and started running in the monitor. The emulator had no record of a breakpoint being set at this address.

Unmatched quote encountered (Error 820)

Cause: In entering a string, such as with the **echo** command, you didn't properly match the string delimiters (either **'** or **"**). For example, you might have entered

echo "set S1 to off

Action: Reenter the command and string, making sure to properly match opening and closing delimiters. Note that both delimiters must be the same character. For example: **echo "set S1 to off"**.

Update HP64740 firmware to version A.02.02 or newer (Error 163)

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 15, "Installing/Updating Emulator Firmware". You must update the firmware to the version number specified in the message, or newer firmware version number. Your system is not usable with its present firmware.

Write to ROM break:<ROM address> (Async_Stat 628)

Cause: This status message indicates the target program accessed memory mapped as either emulation ROM or target ROM; the emulator interrupted target execution and began running in the monitor. This only occurs if you enabled breaks on writes to ROM.
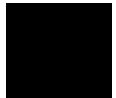
Part 4

Concept Guide

Concept Guide

In This Part

Part 4 of this book explains concepts and shows you how to apply them to advanced tasks.

13

Concepts of the EMSIM and EMRAM
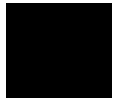
Concepts of the EMSIM and EMRAM

This chapter provides conceptual information on the EMSIM and EMRAM

Concepts of the EMSIM and EMRAM

The 68360 processor provides an array of on-chip peripherals which are configured and used via memory mapped registers. These registers directly control many aspects of the external operation of the processor. The most notable of these on-chip peripherals is the SIM (System Integration Module). For example, address bits A28 through A31 can be configured as either address bits or byte write-enable signals. Selection of these alternative uses drastically changes the external behavior of the processor. Internal to the processor, the full 32-bit address and bus control signals are always maintained. What is seen external to the processor is determined by the current contents of the SIM register set.

The 68360 processor also has internal, on-board, static RAM which can be configured to be addressable anywhere within the address range. The processor RAM register set is used to enable this on-board RAM and to define where it is currently positioned in the 32-bit address space.

The emulator needs access to the full 32-bit address, function codes, and other control signals for proper operation of the emulation-bus analyzer and the emulation memory system. To provide this access when these signals are not available external to the processor, an external bus decoder is designed into the emulator; it recreates these signals.

The following is a view of the emulator implementation:
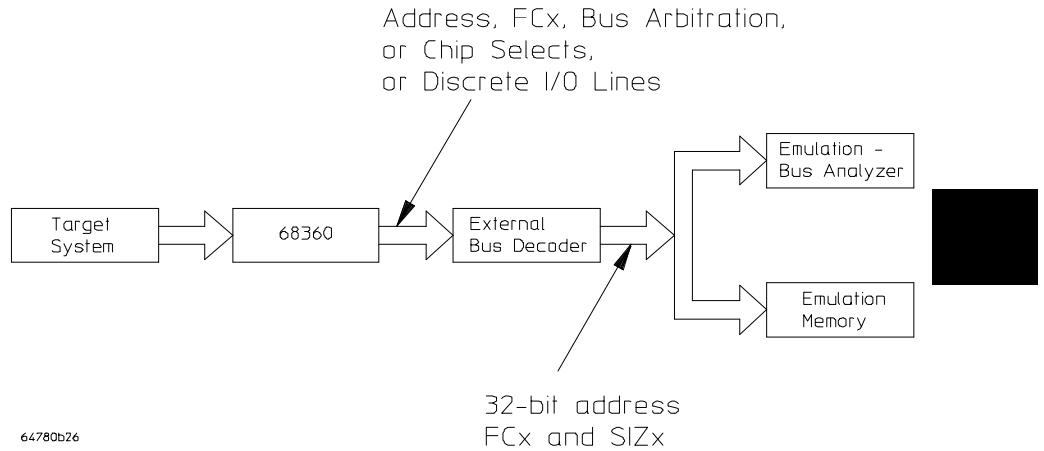
The emulator has been designed to ensure that the emulation-bus analyzer and the emulation memory system will have access to the equivalent of the internal processor 32-bit address bus, function codes, and size information. The external bus decoder circuitry can recreate these signals for all possible combinations of processor pin usage that make sense to the target system. The external bus decoder must be given knowledge of how the processor pins will be used; this knowledge is defined by the EMSIM and EMRAM register sets.

Because the emulation-bus analyzer always receives the full 32-bit address, the user can trace activity based upon the way the code was written, not on the chip selects that are used to access the code. The analyzer can display address symbols in the trace list and accept symbolic address information entered in trace commands.

Because the emulation memory system also receives the full 32-bit address, memory can be allocated (mapped) between the target system and emulation (overlay) memory based upon the full address, not upon chip selects and a subset of the full address bus. When a program download is performed, the program information can be properly directed to emulation or target memory based upon the full 32-bit address contained in the executable file.

The concept of register copies has been implemented in order to accomplish external bus decoding. As part of the emulator configuration, the user can indicate the desired SIM and RAM values by loading the EMSIM and EMRAM copy registers. Once these register copies have been loaded, memory resources (either emulation or target memory) can be accessed in the same manner that the processor will access them when running target code. Note that the default programming of the EMSIM and EMRAM registers is exactly the same as the reset values of the SIM and RAM registers, as defined by the *Motorola 68360 User Manuals*.

In addition to providing the programming knowledge for the external bus decoder, the EMSIM and EMRAM registers provide another helpful feature. Suppose you want to load target memory RAM that has been implemented to be accessible via processor chip selects. In order to access this memory, the processor SIM registers must typically be changed from the reset default values. This can be done by individually modifying each SIM register or by running some processor initialization code. If the EMSIM registers hold the desired values, you can access this memory by simply transferring the EMSIM registers into the SIM registers. As a convenience, this transfer is performed automatically each time the monitor is entered from emulation reset. This is the only time that this transfer is performed automatically; you can manually transfer the EMSIM to the SIM or the SIM to the EMSIM or display their differences at any time.

Concepts of the EMRAM

The 68360 processor has internal RAM modules. These internal RAM modules can be used like any other system RAM; that is, their memory hardware can be assigned to support any desired address range within the 68360 address space.

The address range that a particular internal RAM module will support, and the qualifiers that might additionally be assigned to that address range are defined by the values contained within registers in the Memory Controller of the processor. The values of these registers are supplied to the emulator as part of the EMRAM copy. The emulator uses this information to determine where memory accesses should be routed, based on the current emulator memory map.

The emulator cannot emulate internal RAM modules when the internal RAM is enabled. Internal RAM accesses will typically not be seen outside the processor, and therefore emulation memory will be ignored. The only way to get the processor to access emulation memory which has been mapped to the same address range as internal RAM is to disable the internal RAM, by resetting the processor. Note that internal RAM accesses may be seen externally by the analyzer using the show cycles feature, discussed below.

By default, the internal RAM modules in 68360 processor are turned off. The internal RAMs are enabled and positioned by loading the EMRAM registers. The contents of the EMRAM registers are automatically copied to the RAM registers each time the emulator enters the monitor from emulation reset.

Concepts of Show Cycles

Typically when the processor accesses internal resources (either the Module Control Block, or internal RAM) the bus cycles are not available external to the processor. These bus cycles can be made available by enabling a feature of the 68360 processor called show cycles. In order to capture a trace of activity involving these internal resources, the 68360 processor's show cycles feature is used to make activity available to the analyzer. Two control bits in the SIM_MCR register must be set to enable the show cycles feature. Specifically, these control bits are bits 8 and 9 of the Module Configuration Register. These two bits control external bus arbitration in addition to show cycles. Refer to the *Motorola 68360 User Manual* for detailed information of how to program these bits.

The external-bus decoder within the emulator will automatically decode these "show" bus cycles if the following two conditions are met.

Condition 1: Show cycles are enabled as described above.

Condition 2: The /DS signal is available external to the processor.

If the pin that carries the /DS signal is programmed as a portC I/O pin, the processor is not able to indicate a show cycle and the analyzer will not be able to display show cycles in a trace.

EMSIM/EMRAM 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.

Modify→SIM Registers→Default Emulator SIM

This sets all registers in the SIM and EMSIM to their default values. The default programming of the registers is exactly the same as the reset values defined by the *Motorola 68360 User Manuals*.

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 Port E.

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.
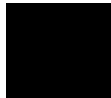
Display→Configuration Info→Reset Mode Value

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→Upper Address Mode

This displays the present address mode, including the size of the address bus and whether the upper address bits are used as A31-A28 or WE3-WE0. This display also describes the distribution of address information for the address mode in use.

Display→Configuration Info→Clock Input Mode

This displays the present mode of clock for the 68360 target system. This mode is set by installation of a clock module in the clock module socket on the emulation probe. Refer to the Hewlett-Packard *MC68360 Installation/Service/Terminal Interface User's Guide* for details.

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 and Service Guide

Installation and Service Guide

In This Part

Part 5 of this book shows you how to:

- Chapter 14, "Installation," shows ou how to install emulation and analysis interface software that supports the Graphical User Interface and the Softkey Interface. Included are instructions for installing this software on HP 9000 systems and Sun SPARCsystems.

- Chapter 15, "Installing/Updating Emulator Firmware," shows you how to update your emulator/analyzer firmware with the progflash command, and display current firmware version information.

The Hewlett-Packard *M68360 Emulator/Analyzer Installation/Service/Terminal Interface User's Guide* shows you how to:

- Connect the emulator into an MC68360 target system and overcome the differences between the specifications and characteristics of the target microprocessor and those of the emulator.

- Install the emulator hardware into the card cage. It also shows how to install the demo board power cable, SRAM modules, rivets and covers, and the emulator probe cable. Then it shows you how to connect the probe to the demo board, and verify performance of the hardware

Refer to the Hewlett-Packard For a thorough analysis of possible problems and solutions, refer to the Hewlett-Packard *M68360 Emulator/Analyzer Installation/Service/Terminal Interface User's Guide*.

14

Installation
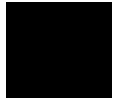
Installation

This chapter shows you how to install emulation and analysis interface software. Installation of emulation and analysis hardware, and performance verification procedures, are shown in Hewlett-Packard's *MC68360 Emulator/Analyzer Installation/Service/Terminal Interface User's Guide*. These installation tasks are described in the following sections:

- Installing HP 9000 software.

- Installing Sun SPARCsystem software.

- Verifying the installation.

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 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. The 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.
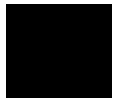
Connecting the HP 64700 to a Computer or LAN

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.)
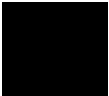
Installing HP 9000 Software

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:

1 Install the software from the media.

2 Verify the software installation.

3 Start the X server and the Motif Window Manager (mwm), or start HP VUE.

4 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 will use the Softkey Interface instead of the Graphical User Interface, do not install the XUI suffixed filesets in the "64700 Operating Environment" and "<processor-type> Emulation Tools" partitions. (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. 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 **/usr/hp64000/lib/X11/app-defaults** subdirectory by entering **ls /usr/hp64000/lib/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 **/usr/hp64000/lib/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 68360 emulator, resource name strings will have **m68360** 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 3a of these instructions. If you plan to run HP VUE, skip to Step 3b of these instructions.
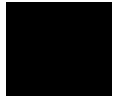
Step 3a. Start the X server and the Motif Window Manager (mwm)

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 4 of these instructions.

Step 3b. 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.

Step 4. 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.

For example, if you installed the HP 64000 software relative to the root directory, "/", you would enter

HP64000=/usr/hp64000; export 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. 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 **usr/hp64000/man** and
usr/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 on-line
"man" page information included with the software.

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:

1 Install the software from the media.

2 Start the X server and OpenWindows.

3 Set the necessary environment variables.

4 Verify the software installation.

5 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 will use the Softkey Interface instead of the Graphical User Interface, do not
install the XUI suffixed filesets. (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.
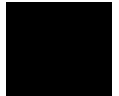
Step 2. Start the X server and OpenWindows

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.

Note that if you see windows on screen, skip this step. The X server must be running in order for windows to be displayed on screen.

- 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.
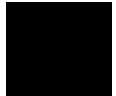
4 Set the MANPATH environment variable to include the **usr/hp64000/man** and **usr/hp64000/contrib/man** directories by entering

setenv MANPATH ${MANPATH}:${HP64000}/man
setenv MANPATH ${MANPATH}:${HP64000}/contrib/man

Including these directories in your MANPATH variable lets you access the on-line "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 68360 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:      em68360     m68360        21.17.9.143
```

A typical entry for a 68360 emulator connected to an RS-422 port would appear as follows:

```
#-----------------------------------------------------------------------------
#                                                      |Xpar|Parity|Flow|Stop|Char
#                                                      |Mode|      |    |Bits|Size
# Channel| Logical  |  Processor | Host |  Physical   |    |      |XON |    |
#  Type  |   Name   |    Type    | Name |  Device     |    |      |    |    |
#                                                      |OFF | NONE |RTS | 2  | 8
#-----------------------------------------------------------------------------
  serial:   em68360    m68360    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 68360 emulator connected to the LAN is "em68360". To start the Graphical User Interface and begin communicating with that emulator, enter (assuming your $PATH includes **$HP64000/bin**)

```
emul700 em68360
```

After a few seconds, the Graphical User Interface Emulator/Analyzer window should appear on your screen. The window will be similar to the following:
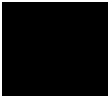
```
┌──────────────────────────────────────────────────────────────────────┐
│ ▤     Hewlett Packard Emulator/Analyzer: em68000 (m68000)     │ ∗ │□│ │
├──────────────────────────────────────────────────────────────────────┤
│  File  Display  Modify  Execution  Breakpoints  Trace  Settings   Help │
│                                                                        │
│  Action keys:  ┌ < Demo > ┐ ┌ Disp Src ( ) ┐ ┌ Trace ( ) ┐ ┌ Run ┐ ┌ Step Source ┐ │
│  ┌ < Your Key > ┐ ┌  Make  ┐ ┌ Disp Src Prev ┐ ┌ Run Xfer to ( ) ┐ ┌ Break ┐ ┌ Step Asm ┐ │
│  ┌────────────────────────────────────────────────────────┬─────────┐ │
│  │ ( ): │main                                              │ Recall  │ │
│  ├──────────────────────────────────────────────────────────────────┤ │
│  │                                                                  │ │
│  │                                                                  │ │
│  │                                                                  │ │
│  │                                                                  │ │
│  │                                                                  │ │
│  │                                                                  │ │
│  │                                                                  │ │
│  │                                                                  │ │
│  │                                                                  │ │
│  │                                                                  │ │
│  │                                                                  │ │
│  ├──────────────────────────────────────────────────────────────────┤ │
│  │ STATUS:    Starting new session                                  │ │
└──────────────────────────────────────────────────────────────────────┘
```
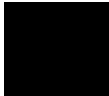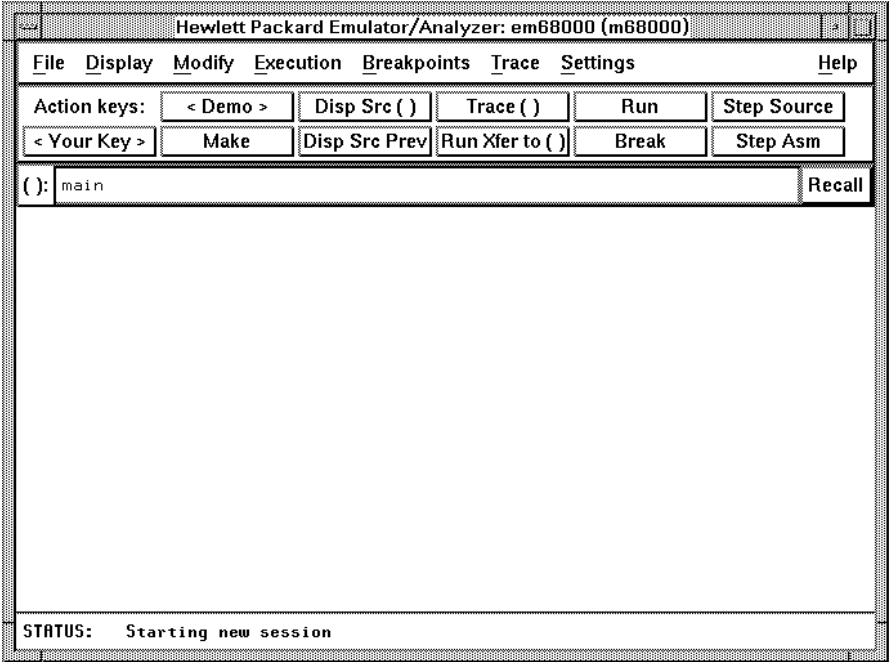
Step 3. Exit the Graphical User Interface

1 Position the mouse pointer over the pulldown 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.

15

Installing/Updating Emulator Firmware
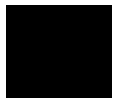
Installing/Updating Emulator Firmware

The 68360 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 68360 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 Chapter 14, "Installation".

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> <product>** 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 <product> option names the product 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 <product> option, it displays the products which have firmware update files on the system and asks you to choose one. You can abort the interactive **progflash** command by pressing <CTRL>c.

progflash will return 0 if it is successful; otherwise, it will return an nonzero (error) and a message will be written on the standard error output.

You can verify the update by displaying the firmware version information.

Examples   To install or update the HP 64780 emulator firmware:

```
$ progflash <RETURN>


    HPB1471-19309 A.04.40 27Sep91
   64700 SERIES EMULATION COMMON FILES

   A Hewlett-Packard Software Product
   Copyright Hewlett-Packard Co. 1991


  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 em68360             m68360

Number of Emulator to Update? (intr (usually cntl C or DEL) to abort)
```

To update firmware in the HP 64700 that contains the 68360, emulator, enter "3".

```
       Product
     1 64700
     2 64703/64704/64706/64740
     3 64744
     4 64780
     5 64760

Number of Product to Update? (intr (usually cntl C or DEL) to abort)
```

To update the HP 64780 68360 emulator firmware, enter "4".

```
Enable progress messages? [y/n] (y)
```

To enable status messages, enter "y".

476

```
Config file path is /usr/hp64000/inst/update/64780.cfg
System firmware revision required = A.03.00
ROM identifier address = 2FFFF0H
Required hardware identifier = 1FF4H
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
Checking System firmware revision...
Rebooting HP64700...
Downloading flash programming code: /usr/hp64000/lib/npf.X
Checking Hardware id code...
Downloading ROM code: /usr/hp64000/inst/update/64780.X
    Code start 280000H (should equal control ROM start)
    Code size 2348CH (must be less than control ROM size)
Finishing up...
Rebooting HP64700...
$
```

You could perform the same update as in the previous example with the following command:

$ **progflash -v** em68360 64780 <RETURN>
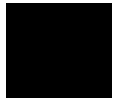
To display current firmware version information

- 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 12Dec93
    Location:  Flash
    System RAM:1 Mbyte

  HP64780 (PPN: 64780) Motorola 68360 Emulator
    Version:   A.03.00
    Speed:     16.7 MHz
    Memory:    512 Kbytes
    CPU:       68360, Mask Set ID = 39

  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.
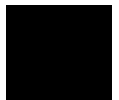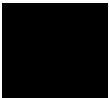
Glossary

Absolute Count

A count in the trace list count column that indicates the total count accumulated between the displayed state and the trigger state.

Absolute File

A file consisting of machine-readable instructions in which absolute addresses are used to store instructions, data, or both. These files are generated by the compiler/assembler/linker and are loaded into the emulator.

Access Breakpoint

A break from execution of your target program to execution of the emulation monitor when the emulator detects a read or write of an address or range of addresses.

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.

Analyzer

An instrument that captures activity of signals synchronously with a clock signal. An emulation-bus analyzer captures emulator bus cycle information. An external analyzer captures activity on signals external to the emulator.

Analyzer Clock Speed

The bus cycle rate of the emulation processor. If the emulation processor is running at 21 MHz and the fastest bus cycle requires three clocks, then the analyzer clock speed (bus cycle rate) is $21/3 = 7$ MHz.

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.

Assembler

A program that translates symbolic instructions into object code.

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. The background monitor can execute when target program execution is temporarily suspended. The background monitor does not occupy any of the address space that is available to your target program.

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. (See also Execution Breakpoint and Access Breakpoint.)

Command File

A file containing a sequence of commands to be executed.

Compatible Mode

The compatible mode of the deep analyzer configures the analyzer to provide the same memory depth as the 1K analyzer: 1024 states deep when the analyzer is not configured to make a count of states or time during a measurement, and 512 states deep when the analyzer is configured to make a count of states or time during a measurement. If the emulator interface you are using along with the deep analyzer requires that you use the compatible mode, the deep analyzer will still be able to provide one of its benefits for your measurement; you will be able to make your counts of states or time at full emulator clock speed.

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.

Counter Overflow

When the counter reaches maximum count and begins a new count from zero. The counter of the deep analyzer simply counts continuously once a trace begins; it increments its count every 20 ns, and reaches maximum count in about 22.9 minutes (22 minutes and 54 seconds). The deep analyzer sets a flag in memory and stores it along with the first state that is captured after the counter overflow occurs (first state captured after the counter begins again at zero).

Configuration File

A file in which configuration information is stored. Typically, configuration files can be modified and re-loaded to configure instruments (such as an emulator) or programs (such as the PC Interface).

Coordinated Measurement

A synchronized measurement made between the emulator and analyzer, between emulation-bus analyzer and external analyzer, or between multiple emulators or analyzers. For example, a coordinated measurement is made when two or more HP 64700 emulators/analyzers start executing together, or break into background monitors at the same time.

Coordinated Measurement Bus (CMB)

The bus that is used for communication between multiple HP 64700 Series emulators/analyzers or between HP 64700 emulators/analyzers and an HP IMB/CMB Interface to allow coordinated measurements.

Cross Trigger

The situation in which the trigger condition of one analyzer is used to trigger another analyzer. Two signals internal to the HP 64700 can be connected through the BNC on the instrumentation card cage to allow cross-triggering between the emulation-bus analyzer and other analyzers.

DCE (Data Communications Equipment)

A specific RS-232C hardware interface configuration. Typically, DCE is a modem.

Deep Analyzer

In this manual, the term "deep analyzer" refers to the HP 64794 Emulation-Bus Analyzer with deep trace memory.

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.

Downloading

The process of transferring absolute files from a host computer into the emulator.

Embedded Microprocessor System

The microprocessor system which the emulator plugs into.

Emulation-Bus Analyzer

The internal analyzer that captures emulator bus cycle information synchronously with the processor's clock signal.

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 Memory

High-speed memory (RAM) in the emulator that can be used in place of target system memory.

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.

Emulator Probe

The assembly that connects the emulator to the target system microprocessor socket.

Escape Sequence (transparent mode)

A keyboard input consisting of a special sequence of characters, beginning with the escape character (1C hexadecimal). This sequence is used to access an emulator while in transparent mode. When using multiple emulators and transparent mode to access the different emulators, each one must be given a unique escape character.

Execution Breakpoint

A BKPT instruction placed in your software in RAM, replacing the normal instruction at the RAM address. Breakpoints for code in ROM are stored in emulation hardware and jammed on the emulation bus during the fetch cycle. When the BKPT is executed, emulation immediately transfers from execution of your target program to execution of the emulation monitor.

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.

Foreground Monitor

A monitor program that executes in the foreground address space. When the monitor exists in foreground, it is directly accessible by, and can interact with, your target program.

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.

Guarded Memory

An address range that is to be inaccessible to the emulation processor. The emulator will generate a break and display an error message if an access to guarded memory occurs.

Handshaking

A process that involves receiving and/or sending control characters which indicate a device is ready to receive data, that data has been sent, and that data has been accepted.

Host Computer

A computer to which an HP 64700 Series emulator can be connected. A host computer may run interface programs which control the emulator. Host computers may also be used to develop programs to be downloaded into the emulator.

Inverse Assembler

A program that translates absolute code into assembly language mnemonics.

Label

A set of one or more analyzer channels. Example, the label "addr" is used to identify the analyzer channels connected to the address bus of the emulation processor.

Linker

A program that combines relocatable object modules into an absolute file which can be loaded into the emulator and executed.

Logical Address Space

The addresses assigned to code during the process of compiling, assembling and linking to generate absolute files.

Macros

Custom made commands that represent a sequence of other commands. Entire sequences of commands defined in macros will be automatically executed when you enter the macro name. Macro nesting is permitted; this allows a macro definition to contain other macros.

Memory Mapper Term

A number assigned to a specific address range in the memory map. Term numbers are consecutive.

Memory Mapping

Defining ranges of the processor address space as emulation RAM or ROM, target RAM or ROM, or guarded memory.

Monitor Program

A program executed by the emulation processor that allows the emulation system 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.

Operating System

Software which controls the execution of computer programs and the flow of data to and from peripheral devices.

Overflow

See counter overflow.

Parity Setting

The configuration of the parity switches. Depending on the configuration of the parity output switch and the parity switch, a parity check bit is added to the end of data to make the sum of the total bits either even or odd. A parity check is performed after data has been transferred, and is accomplished by testing a unit of the data for either odd or even parity to determine whether an error has occurred in reading, writing, or transmitting the data.

Path

Also referred to as a directory (for example \users\projects).

Pass Through Mode

See Transparent Mode.

PC Interface

A program that runs on the HP Vectra and IMB PC/AT compatible computers. This is a friendly interface used to operate an HP 64700 Series emulator.

Performance Verification

A program that tests the emulator to determine whether the emulation and analysis hardware is functioning properly.

Physical Address Space

The address space in hardware memory and hardware I/O that is accessed by the microprocessor during normal program execution.

P/O

An abbreviation for "part of." Used in illustrations to show that a part is shipped with other parts under a certain HP part number.

Prefetch

The ability of a microprocessor to fetch additional opcodes and operands before the current instruction is finished executing.

Prestore

The storage of states captured by the analyzer that precede states which are normally stored. If the normal storage qualifier specifies the entry address of a function or routine, prestore can be used to identify the callers of that function or routine.

Prestore Qualifier

A specification that must be met by a state before it can be saved in the analyzer prestore memory.

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.

Qualifier

A specification that must be met before an action can be taken by the analyzer. For example, a store qualifier is a specification that must be met by an incoming state before it can be stored in the trace memory. The "arm" condition can be used as an additional qualifier. For example, an external analyzer may be set up to supply a true signal to the rear panel BNC connector on the card cage when it detects a true condition in the target system. Then the analyzer can be set up to store qualify a certain kind of state, but only when the arm signal from the BNC is true.

Real-Time Execution

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.)

Relative Count

A count in the trace list count column that shows the count between the present displayed state and the state displayed immediately before it. Relative time count, for example, shows the elapsed time between the previous displayed state and the present state. Note that the count is between displayed states. If your trace list is inverse assembled and/or dequeued, several states may have been captured in memory between the present displayed state and the displayed state immediately before it.

Remote Configuration

The configuration in which an HP 64700 Series emulator is directly connected to a host computer via a single port. Commands are entered (typically from an interface program running on the host computer) and absolute code is downloaded into the emulator through that single port.

RS-232C

A standard serial interface used to connect computers and peripherals.

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.

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.

Single-step

The execution of one microprocessor instruction. Single-stepping the emulator allows you to view program execution one instruction at a time.

Software Breakpoint

Refer to execution breakpoint and access breakpoint in this glossary.

Software Performance Analyzer

An analyzer that measures execution of software modules, interaction between software modules, and usage of data points and I/O ports.

Standalone Configuration

The configuration in which a data terminal is used to control the HP 64700 Series emulator, and the emulator is not connected to a host computer.

stderr

An abbreviation for "standard error output." Standard error can be directed to various output devices connected to the HP 64700 ports.

stdin

An abbreviation for "standard input." Standard input is typically defined as your computer keyboard.

stdout

An abbreviation for "standard output." Standard output can be directed to various output devices connected to the HP 64700 ports.

Step

See Single-step.

Store Qualifier

A specification that must be met by a state before it can be saved in the analyzer trace memory.

Synchronous Execution

The execution of multiple HP 64700 Series emulators/analyzers at the same time (i.e., multiple emulator start/stop).

Syntax

The order in which expressions are structured in command languages. Syntax rules determine which forms of command language syntax are grammatically acceptable.

Target Program

The program you are developing for your product. It is also called user program.

Target System

The circuitry where the emulator probe is connected (typically a microprocessor-based system under development).

Target System Memory

Storage that is present in the target system.

Terminal Interface

The command interface present inside the HP 64700 Series emulators that is used when the emulator is connected to a simple data terminal. This interface provides on-line help, command recall, macros, and other features which provide for easy command entry from a terminal.

Trace

A collection of states captured synchronously by the emulation-bus analyzer and stored in trace memory.

Trigger

The condition that identifies a reference state within an analyzer trace measurement. Trigger also refers to the analyzer signal that becomes active when the trigger condition is found.

Trigger signals called trig1 and trig2 are bidirectional signal lines that can be used to coordinate measurement activity between emulators and analyzers installed in the instrumentation card cage, and between instruments connected to the BNC on the rear panel of the card cage.

Note that there is a delay when you use a trigger for measurement coordination. For example, you may specify that the emulator break to its monitor program when it receives trig1 from the analyzer. Several states may be executed in the emulator between the time the analyzer recognizes its trigger condition, generates trig1, delivers trig1 to the emulator, and the emulator responds to trig1 by breaking to its monitor program.

Uploading

The transfer of emulation or target system memory contents to a host computer.

Unlocked Exit

One of two methods used to leave the high level (Graphical or Softkey) Interface and return to the host computer operating system. An unlocked exit command allows you to exit the high level interface and re-enter later with the default configuration. (See also Locked Exit.) This is not available in the Terminal Interface.

User Program

Another name for your target program (the program you are developing for your product.

Viewport

See Window.

Wait States

Extra microprocessor clock cycles that increase the total time of a bus cycle. Wait states are typically used when slower memory is implemented.

Window

A specified rectangular area of virtual space shown on the display in which data can be observed.

1K Analyzer

The term "1K analyzer" refers to the HP 64704 Emulation-Bus Analyzer with 1K trace memory.

Index

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.

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. 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.

Designed to Meet Requirements of IEC Publication 348

This apparatus has been designed and tested in accordance with IEC Publication 348, safety requirements for electronic measuring apparatus, and has been supplied in a safe condition. The present instruction manual contains some information and warnings which have to be followed by the user to ensure safe operation and to retain the apparatus in safe condition.

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.

Hot Surface. This symbol means the part or surface is hot and should not be touched.

Indicates dangerous voltage (terminals fed from the interior by voltage exceeding 1000 volts must be marked with this symbol).
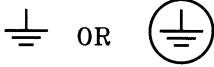
Protective conductor terminal. For protection against electrical shock in case of a fault. Used with field wiring terminals to indicate the terminal which must be connected to ground before operating the equipment.

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.
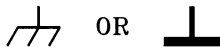
Frame or chassis terminal. A connection to the frame (chassis) of the equipment which normally includes all exposed metal structures.

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.