HEWLETT PACKARD

User's Guide for the Graphical User Interface

MC68040/EC040/LC040 Emulator/Analyzer (HP 64783A/B)

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

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

Printing History

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

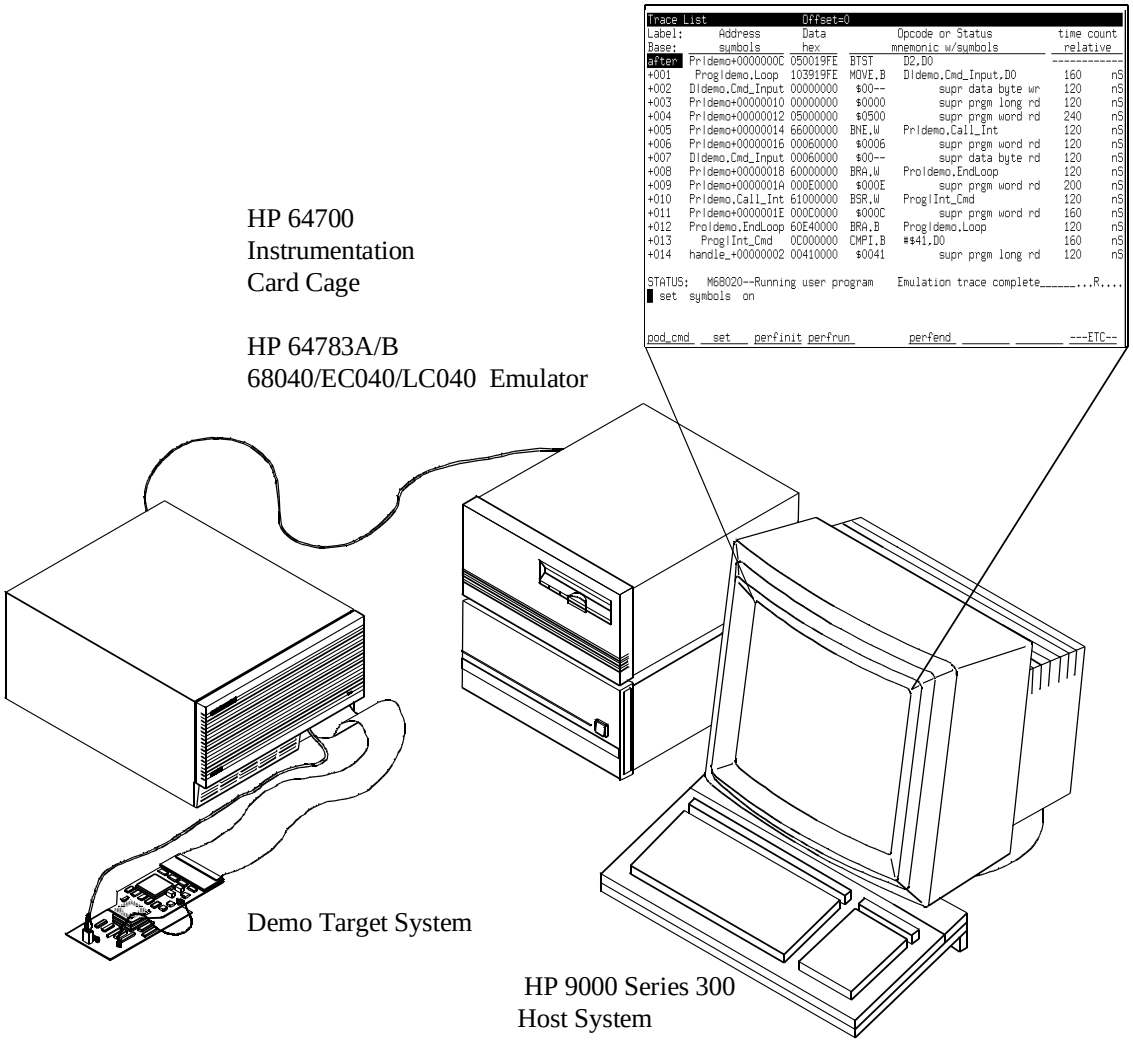
A software code may be printed before the date; this indicates the version level of the software product at the time the manual was issued. Many product updates and fixes do not require manual changes, and manual corrections may be done without accompanying product changes. Therefore, do not expect a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

Edition 1	B3090-97000, March 1993
Edition 2	B3090-97001, October 1993
Edition 3	**B3090-97002, January 1994**

Safety and Certification and Warranty

Safety information, and certification and warranty information can be found on the pages before the back cover.

The HP 64783A/B Emulator



The HP 64783A/B Emulator

Description

The HP 64783A/B emulator supports the Motorola 68040, 68EC040, and 68LC040 microprocessors operating at clock speeds up to 33 MHz (HP 64783A) or 40 MHz (HP 64783B). Differences between the three microprocessors are shown in the table below:

Motorola Processor	Includes MMU	Includes FPU
68040	yes	yes
68EC040	no	no
68LC040	yes	no

The emulator uses an MC68040 microprocessor and is pin-for-pin compatible with the MC68EC040 and MC68LC040 microprocessors. Refer to the end of Chapter 4, "Using the Emulator", for special considerations when using the emulator in target systems designed with the MC68EC040 or MC68LC040.

Throughout this manual, the microprocessor will be referred to as the MC68040, except where the three versions must be discussed separately.

Additionally, this emulator supports development of target systems using the MC68040 together with up to 31 MC68360's in slave mode. Refer to the end of Chapter 4, "Using the Emulator", for an explanation of this emulator's support for the MC68360 slave mode.

The emulators plug into the modular HP 64700 instrumentation card cage and offer 80 channels of processor bus analysis with the HP 64704A or HP 64794A emulation-bus analyzer. Flexible memory configurations are offered from zero through two megabytes of emulation memory. High performance download is achieved through the use of a LAN or RS-422 interface. An RS-232 port 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 HP workstations. This environment includes an ANSI standard

C compiler, assembler/linker, a debugger that uses either a software simulator or the emulator for instruction execution, the HP Software Performance Analyzer that allows you to optimize your product software, and the HP Branch Validator for test suite verification.

If your software development platform is a personal computer, support is 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 64783A/B Emulator

- 16 to 33 MHz active probe emulator (HP 64783A)
- 20 to 40 MHz active probe emulator (HP 64783B)
- Supports MC68040, MC68EC040, and MC68LC040
- Supports burst and synchronous bus modes
- Symbolic support
- Number of breakpoints available:
  - If specified at RAM addresses: unlimited;
  - If specified at ROM addresses: eight.
- 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, which uses the static deMMUer of the MC68040 emulator
- Postprocessed, 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

- 256 Kbyte, 512 Kbyte, 1 Mbyte, 1.25 Mbyte and 2 Mbyte memory configurations available
- 4 Kbytes of dual-ported memory available if you use the background monitor.
- Mapping resolution is 256 bytes
- No wait states required by the emulator for processor speeds up to 25 MHz
- One wait state required in all accesses above 25 MHz

In This Book

This manual covers the HP 64783A/B emulator. All information in the manual applies to all three Microprocessor versions, unless it is marked with the processor name (MC68040, MC68EC040, or MC68LC040).

Part 1, “Quick Start Guide,” tells you how to start using the emulator.

1. Getting Started
2. Solving Quick Start Problems

Part 2, “User’s Guide,” describes how to use the emulator/analyzer interface to perform a variety of tasks.

3. Using the Emulator/Analyzer Interface
4. Using the Emulator
5. Using the Emulation-Bus Analyzer
6. Making Coordinated Measurements
7. Making Software Performance Measurements
8. Configuring the Emulator (to be performed before you run a program in emulation)
9. Solving Problems

Part 3, “Reference Guide,” provides detailed information on emulator functions, commands and environments.

10. Using MC68040 Memory Management
11. Emulator Commands
12. Emulator Messages
13. Setting X Resources
14. The SPARCsystem Interface
15. Microtec Language Tools used with the Emulator
16. Specifications and Characteristics

Part 4. "Concepts Guide," discusses X Resources and the Graphical User Interface.

Part 5, “Installation and Service Guide,” shows you how to install and maintain the emulator.

18. Installation and Service
19. Installing/Updating Emulator Firmware

Contents

9 Solving Problems

Part 1

Quick Start Guide

Quick Start Guide

In This Part

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

1

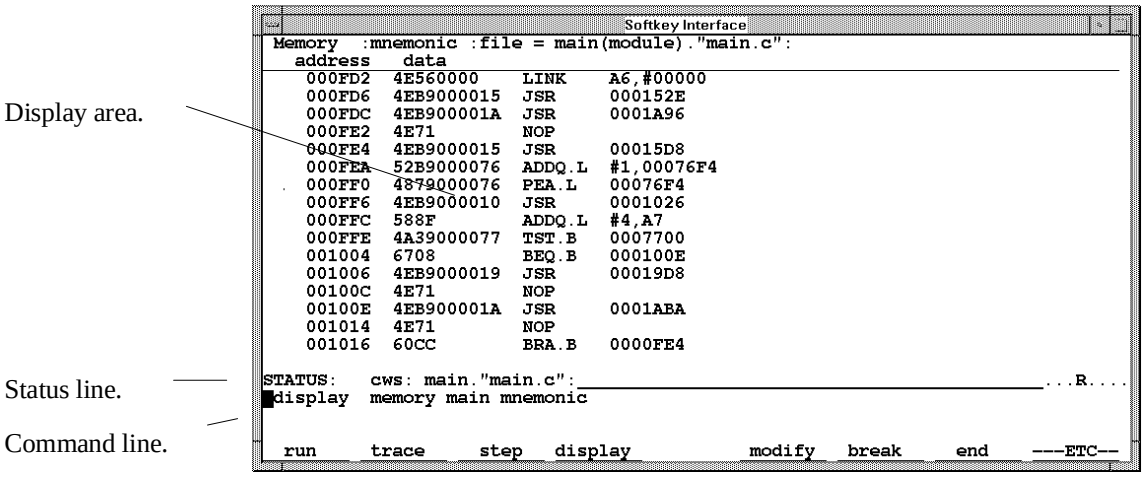
Getting Started

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. Some interface features include 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.

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. 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.

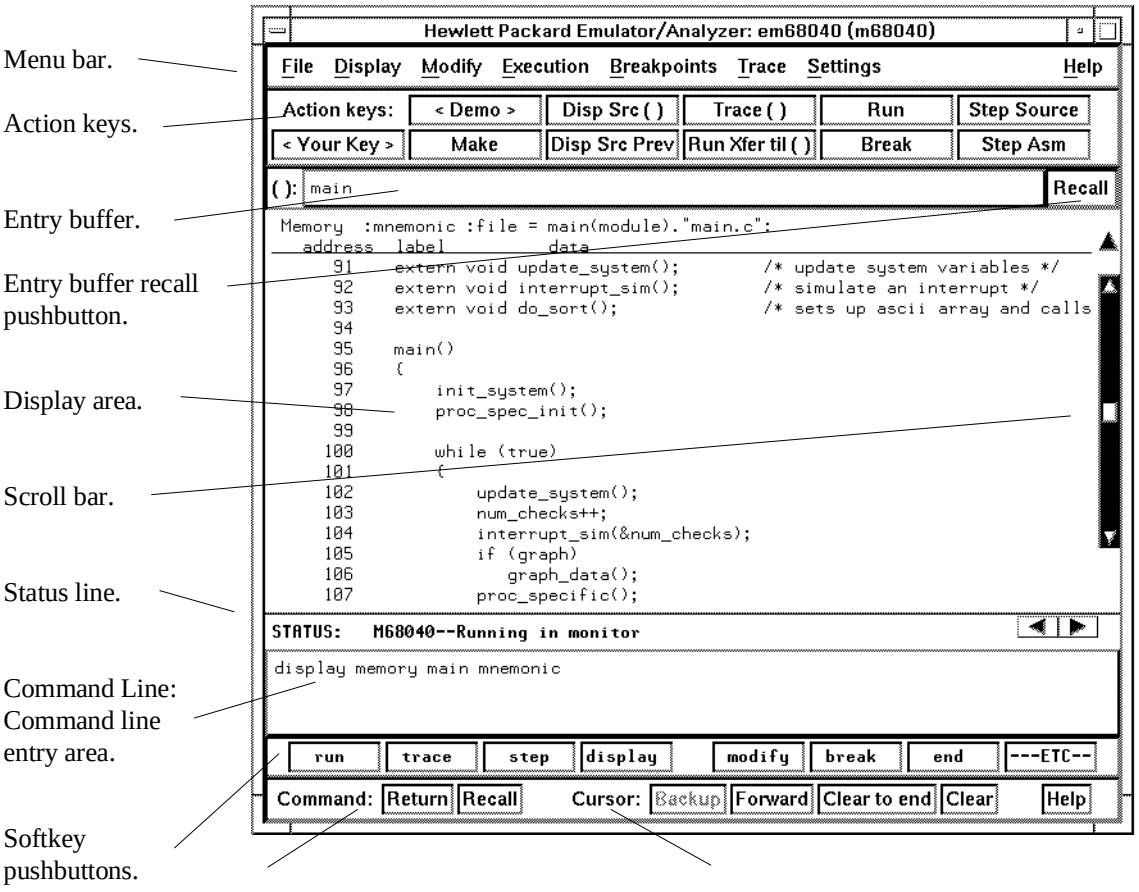
Softkey Interface Conventions

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.

Softkey interface commands are executed by pressing the carriage return key on the keyboard.

The Graphical User Interface



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

Display Area. Can show memory, data values, MMU translation tables, 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 popup 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 popup 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 emulator 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 popup menu.
- **Command pushbuttons** (includes command recall pushbutton). The command **Return** pushbutton 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** pushbutton allows you to recall previous or predefined commands. When you click on the command **Recall** pushbutton, a dialog box appears that allows you to select a command.

- **Cursor pushbuttons for command line area control**. Allows 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, popup 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 Descriptions

Generic Button Name	Bindings: HP 9000	Bindings: 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 popup 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 table shows 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 screen displays shown in this chapter were obtained by running the MC68040 emulator/analyzer.

The tutorial examples presented in this chapter make the following assumptions:

- The HP 64783 emulator and HP 64704 analyzer are installed into the HP 64700 Card Cage.
- The HP 64700 is connected to the host computer.
- The emulator/analyzer interface software has been installed as outlined in Chapter 19, "Installation and Service", and updated as outlined in Chapter 20, "Installing/Updating Emulator Firmware".
- The emulator is operating out-of-circuit; that is, connected to the demo board, not your target system, as outlined in Chapter 19, "Installation and Service".
- The emulator contains at least 60 Kbytes of emulation memory.
- Power is turned on to the instrumentation card cage and the LED on the demo board is lit.

The Demonstration Program

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

Step 1: Start the demo

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

1 Change to the demo directory.

- Type:

```
$ cd /usr/hp64000/demo/debug_env/hp64783
```

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

```
$ more README
```

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

```
$ echo $PATH
```

3 If the Graphical User Interface software is installed on the same computer or same type of computer that you are using to run this "Getting Started" procedure, skip this step and go directly to step 4 of this "Start the demo" procedure.

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 an HP 9000 computer and be displayed on a Sun SPARCsystem computer, change the platform scheme to "SunOS". This can't be done in the demo directory specified above because the Xdefaults.emul file is write-protected. You will need to move it to a new directory and then change its permissions. The best way to do this is to enter the command:

```
$ Startemul <logical_emul_name>
```

In the above command, <logical_emul_name> is the logical name of your emulator, given in the HP 64700 emulator device table file (/usr/hp64000/etc/64700tab.net).

After you give the "Startemul" command, you will be asked if you would like to have the demo files copied to a different directory. Answer yes, and then specify your own demo directory. The files will be copied to your own directory where you can change the permissions on the Xdefaults.emul file so that you can edit it. Type:

```
$ chmod 664 Xdefaults.emul
```

Now edit the Xdefaults.emul file. For this example, you would edit as follows:

```
$ vi Xdefaults.emul
```

```
!*.platformScheme: pc-xview
!*.platformScheme: HP-UX
*.platformScheme: SunOS
!*.platformScheme: HPxterm
```

Finally, save the Xdefaults.emul file with its modifications, and then start the emulation session again from the demo directory where you have your custom Xdefaults.emul file.

4 Start the emulator/analyzer demo with the command:

```
$ Startemul <logical_emul_name>
```

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). For the MC68040 emulator, it is usually m68040.

If you did not perform Step 3 of this "Start the demo" procedure, you will be asked if you would like to have the demo files copied to a directory of your own choosing. It is a good idea to have these files copied to your own demo directory if you have space available in your system because it protects the original demo files from changes you might make during this demo procedure.

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.

Step 2: Display the program in memory

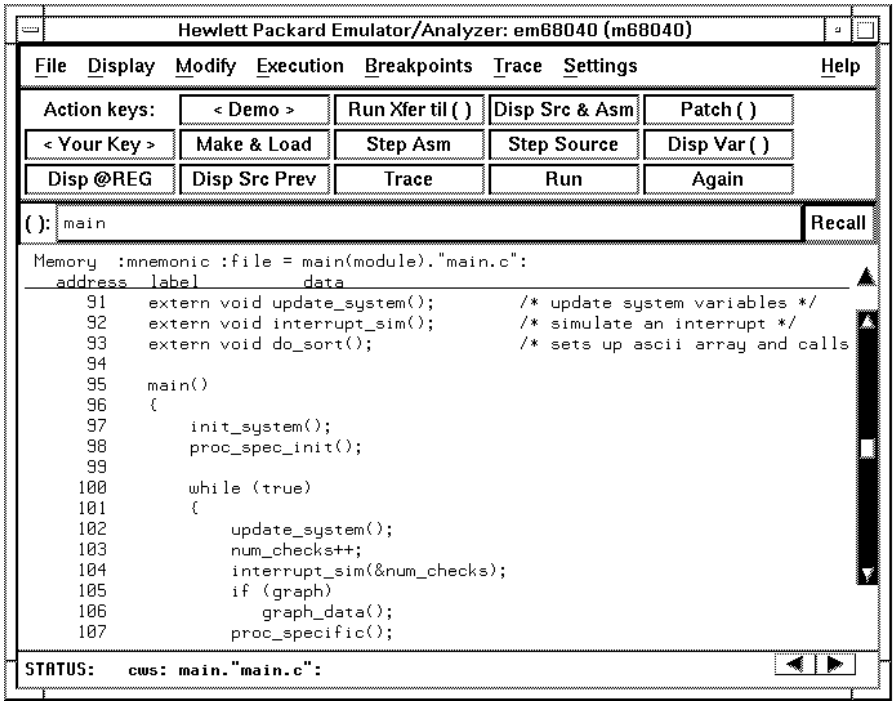
1 If the symbol "main" is not already in the entry buffer, move the mouse pointer to the entry buffer (notice the flashing I-beam cursor) and type in "main".

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

Or, using the command line, enter:

```
display memory main mnemonic
```

The command line can be brought on screen by choosing **Settings→Command Line** in the menu bar or placing the cursor in the display area and typing.



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.

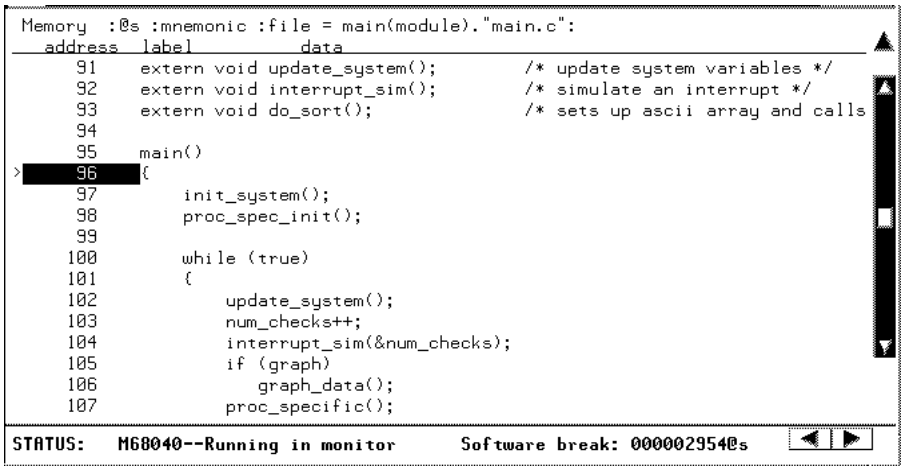
Step 3: Run from the transfer address

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`



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 on the STATUS line to obtain this message. 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 content of 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.

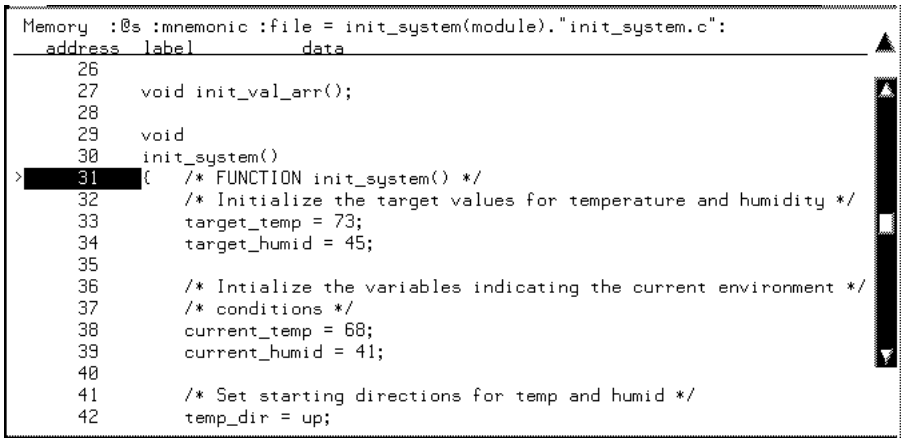
1 To step a source line from the current program counter, click on the **Step Source** action key, or choose **Execution→Step Source→from PC.**

Or, using the command line, enter:

```
step source
```

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

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



Step 5: Display the previous mnemonic display

- Click on the **Disp Src Prev** action key, or choose **Display→Memory→Mnemonic Previous.**

  Or, using the command line, enter:

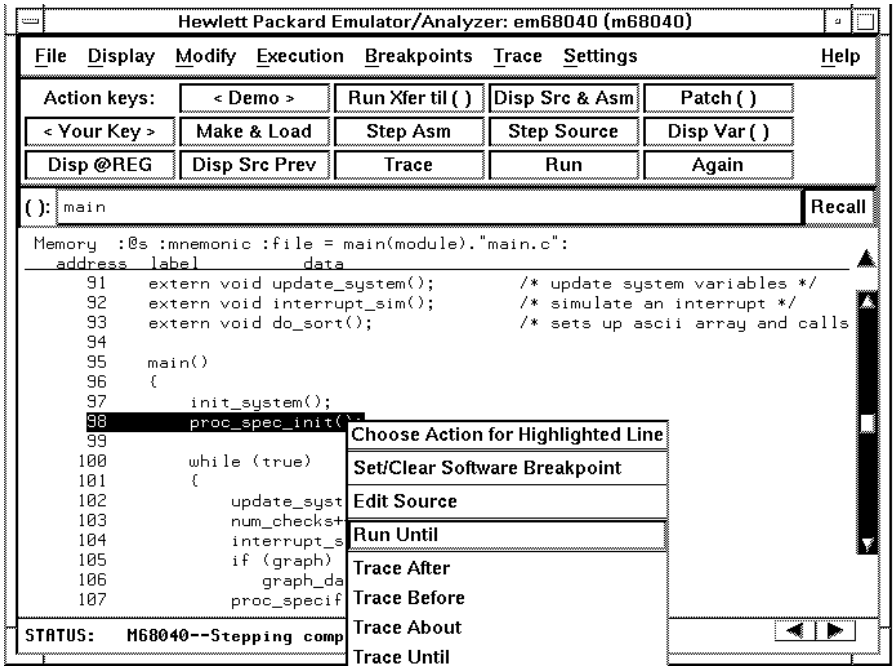
  display memory mnemonic previous_display

  This command is useful, for example, when you have stepped into a function that you do not wish to look at—you can return the previous mnemonic display to the screen and run your program until the source line that follows the function call is reached. The next step in this procedure will show you how to make the emulator run through the function "init_system();" and stop when "proc_spec_init();" is reached.

Step 6: Run until an address

When displaying memory in mnemonic format, a selection in the popup 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 popup menu.



Or, using the command line, enter:

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

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

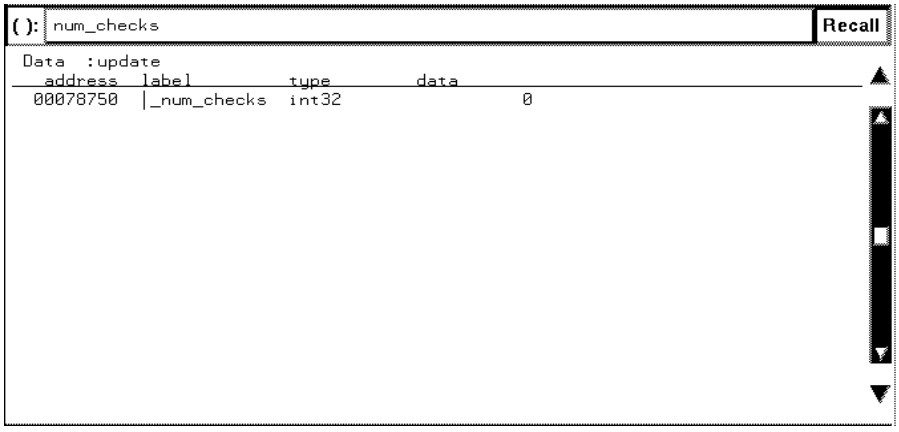
Step 7: Display data values

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

2 Click on the **Disp Var ()** action key, or choose **Display→Data Values→Add()→int32**.

Or, using the command line, enter:

```
display data , num_checks int32
```



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

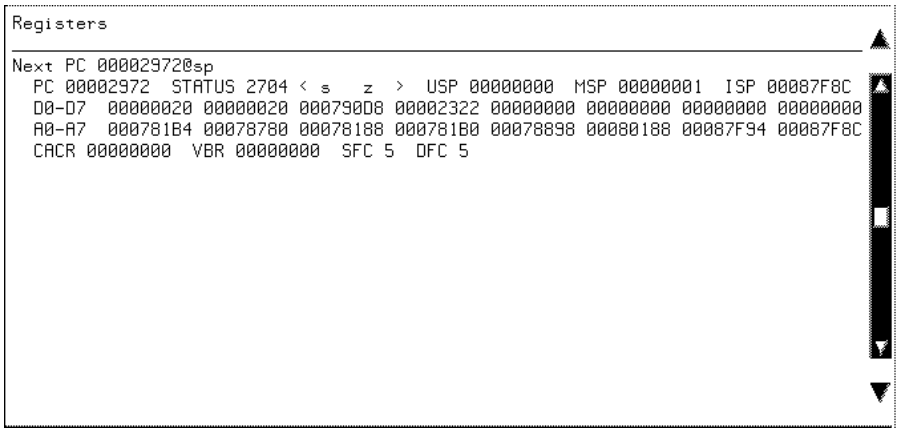
Step 8: Display registers

You can display the contents of the processor registers.

- Choose **Display→Registers→BASIC.**

  Or, using the command line, enter:

  display registers



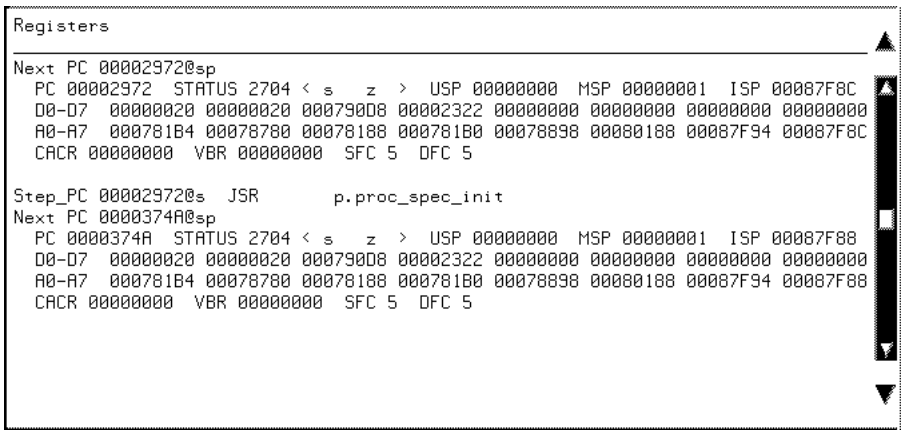
Step 9: Step assembly-level instructions

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 choose **Execution→Step Instruction→from PC**.

  Or, using the command line, enter:

  step



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

Step 10: Trace the program

When the analyzer traces program execution, it looks at the data on the emulation processor's bus and control signals at each bus cycle. The information seen at a particular bus 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** pushbutton 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 long_aligned main
```

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

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

Or, using the command line, enter:

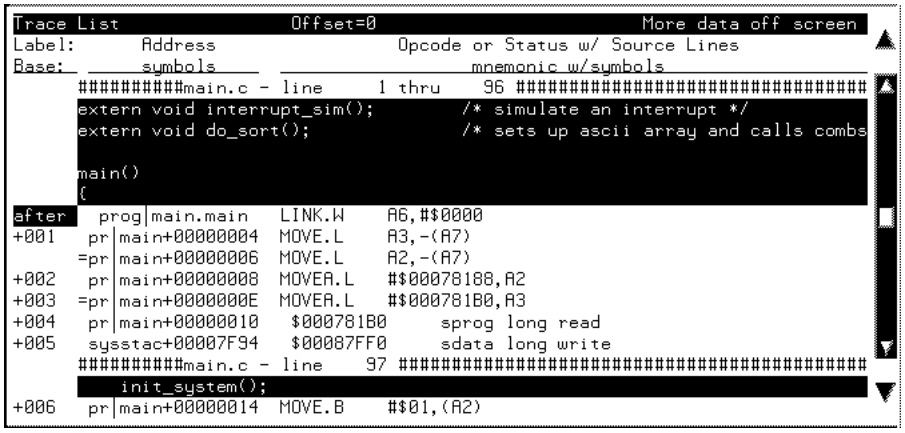
```
run from transfer_address
```

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

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

Or, using the command line, enter:

display trace



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. A "Define command file ..." dialog box appears.

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

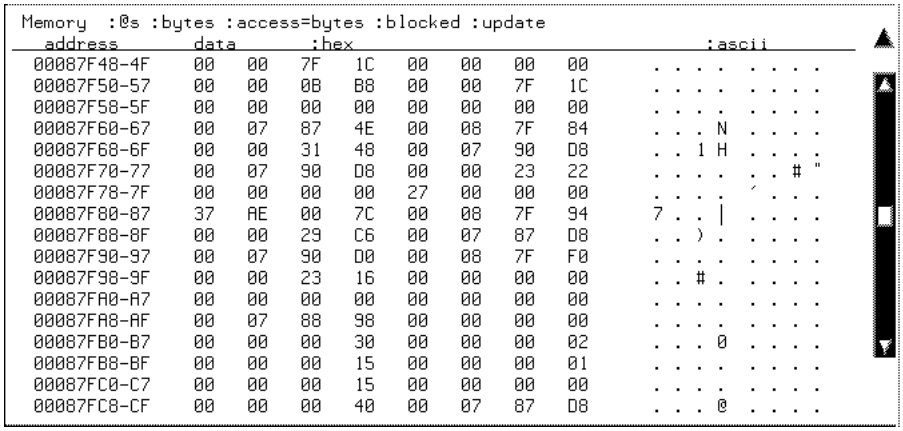
```
mematreg
```

A prompt appears in the command line.

2 Move the mouse pointer to the "Define command file ..." dialog box text entry area, type "A7", and click the "OK" pushbutton.

Or, if the prompt is in the command line, enter:

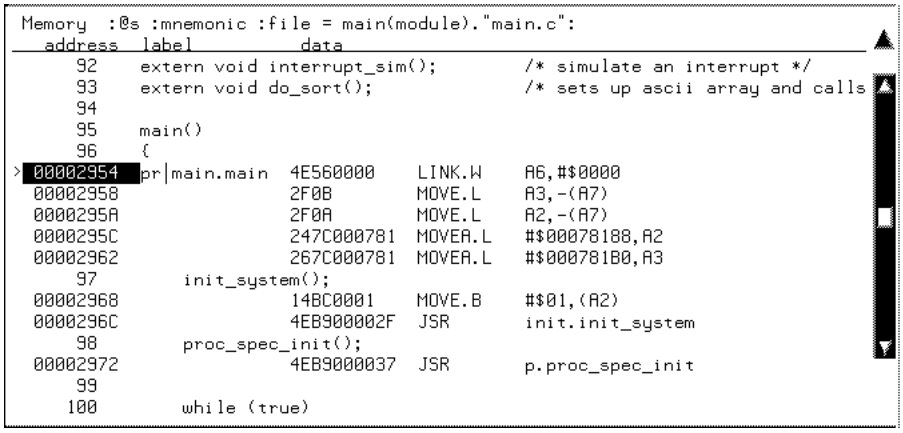
```
A7
```



Step 12: Patch assembly language code

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

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



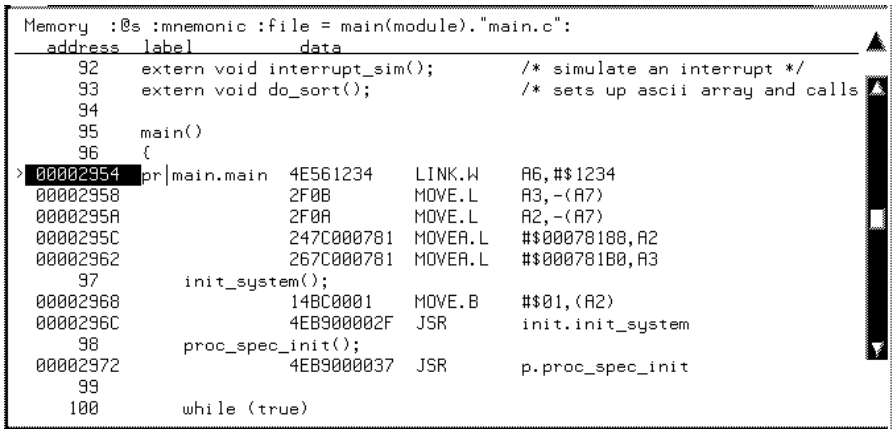
3 Click on the **Patch ()** action key.

A window appears and the **vi** editor is started. Under "ORG main", add the line:

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

Exit out of the editor, saving your changes (using 'wq').

The **Patch ()** action key lets you patch code in your program. The file you just edited is assembled, and the patch main menu appears. Type "a" beside "Enter choice:", and then press your carriage return key to apply the patch.



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

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

A window running the **vi** editor again appears, allowing you to modify the patch code that was just created. Modify the line you added previously to:

```
LINK A6,#0
```

Exit out of the editor, saving your changes.

The file you just edited is assembled, and the patch main menu appears. Type "a" and press your carriage return key to apply the patch.

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

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

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

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

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

A window running the **vi** editor again appears. Suppose you want to patch the demo program so that the proc_spec_init() function is called before the init_system() function. Suppose also that there is memory available at address 7FE0H. Edit the patch template file as shown below.

```
; PCHS700 Assembly Patch File: PCHmain+4.s
;
; Date : Fri Feb 12 14:06:06 MDT 1993
; Dir  : /users/guest/demo/debug_env/hp64783
; Owner: guest
;
        INCLUDE PCHSINC.s
        ORG main+4
        BRA patch1      ;You may want to change this name!
        ORG 7FE0h       ;You MUST set this address!
patch1  NOP
; !!!!!!!!! You may need to modify labels and operands of the     !!!!!!!!!
; !!!!!!!!! following code to match your assembler syntax         !!!!!!!!!
; !!!!!!!!! Patching Range: main+4 thru main+15
; !!!!!!!!! Insert new code here !!!!!!!!!
        JSR _proc_spec_init
        JSR _init_system
        BRA main+16     ;You MUST set this address also!
```

Notice that symbols can be used in the patch file. Exit out of the editor, saving your changes ('wq').

The file you just edited is assembled, and the patch main menu appears. Type "a" and press your carriage return key to apply the patch.

You can step through the program to view execution of the patch. Place "main" in the entry buffer and use the **Step Source** action key, or choose **Execution→Step Source→from ()**.

Or, using the command line, enter:

step source

The MMU Demonstration

The remainder of this demonstration shows how the MC68040 emulator helps you develop and analyze your target program within a memory system that is managed by the MMU of the MC68040 processor.

The MMU demo program attempts to simulate a real target system to display hexadecimal characters on the seven-segment display on the HP 64783A demo board (used with HP 64783A and HP 64783B emulators). A simple operating system uses interrupts to maintain a system clock and configures the MMU to translate addresses and provide memory access protection. The operating system waits for a hexadecimal string to be placed in "sysbuf" and then spawns a user task to display each character on the demo board's seven-segment display. The user task interfaces with the operating system to set up an alarm timer and to output individual characters to the display. The display of characters is interrupt driven.

This demo requires 128 Kbytes of emulation memory. The first 64-Kbyte block of emulation memory is mapped to lower memory and corresponds to system ROM. ROM space is translated 1:1 and is entirely write protected. The first half of ROM contains privileged operating system code and is also protected against user mode access. The second half of ROM is user accessible and contains shared library and operating system interface functions. The second 64-Kbyte block of emulation memory is mapped to upper memory and corresponds to system RAM. RAM space is NOT translated 1:1 and has varying access protections. During bootup, the operating system loads the user program, user data, and operating system data from ROM into RAM after the MMU is enabled. The user program is loaded into the first half of RAM and gets write-protected. The user data and stack is located in the next quarter of RAM and has no access protections. The last quarter of RAM contains operating system data and stack and is protected against user access. A transparent translation register is used to provide a 1:1 address translation for the emulation monitor located at 0xff000000. The following MMU translation display summarizes all address translations:

```
Logical Address     Physical Address      Attributes
00000000..00007fff  00000000..00007fff@a  S W  (32K sprog)
00008000..0000ffff  00008000..0000ffff@a  W    (32K libc)
00010000..00017fff  ffff0000..ffff7fff@a  W    (32K uprog)
00010000..0001bfff  ffff8000..ffffBfff@a       (16K udata)
0001C000..0001ffff  ffffC000..ffffffff@a  S    (16K sdata)
ff000000..ffffffff  ff000000..ffffffff@a  TT   (monitor)

Where:
      S = Supervisor access only.
      W = Write-protected.
      TT = Controlled by a transparent translation register.
```

Read the README file in the mmudemo directory, which you will access next in this demonstration procedure. The README file suggests tests you can make, in addition to those shown in this chapter, with the demo program. These can help you become more comfortable with use of the MMU in this emulator.

Step 13: Obtain the normal interface and MMU demo

1 The MMU demo program is run from the normal interface of the MC68040 emulator/analyzer, not the special interface you used to run the "ecs" demo in the first part of this chapter. If you still have the graphical user interface on screen, choose **File→Exit→Released.**

If using the softkey interface, enter the command:

```
end release_system
```

2 Obtain the MMU demo directory by typing the command:

```
$cd /usr/hp64000/demo/debug_env/hp64783/mmudemo
```

3 Start the normal MC68040 emulator/analyzer with the command:

```
$emul700 <logical_emul_name>
```

The <logical_emul_name> in the command above is the logical emulator name given in the HP 64700 emulator device table file that you used in the first demonstration (Step 1, Substep 4 of this procedure).

4 Load the MMU demo program and start it running, as follows:

Choose **File→Load→ Emulator Config ...** In the file selection dialog box, select /usr/hp64000/demo/debug_env/hp64783/mmudemo/demo.EA, and click OK.

Choose **File→Load→Executable ...** In the file selection dialog box, select /usr/hp64000/demo/debug_env/hp64783/mmudemo/demo.x, and click OK.

Step 13: Obtain the normal interface and MMU demo

Choose **Execution→Run→from Reset**.

Or, using the command line, enter the following commands:

```
load configuration demo.EA
load demo.x
set source memory_only_trace_on
run from reset
```

Step 14: See the setup of the MMU

- Choose **Display→MMU Translations**

- Or, using the command line, enter the following command:

  display mmu_translations

  The above commands let you see the present setup of the MMU. The MMU was set up by the demo program when you first started it.

```
Logical Address          Physical Address          Attributes
 000000000..000007fff     000000000..000007fff@a    S W
 000008000..00000ffff     000008000..00000ffff@a    W
 000010000..000017fff     0ffff0000..0ffff7fff@a    W
 000018000..00001bfff     0ffff8000..0ffffbfff@a
 00001c000..00001ffff     0ffffc000..0ffffffff@a    S
 0ff000000..0ffffffff     0ff000000..0ffffffff@a    TT
```

  Note that the first and second ranges of logical addresses are translated 1:1 to their physical addresses. The third, fourth, and fifth ranges of logical addresses are translated to different physical addresses. The last range of logical addresses is translated 1:1 to its corresponding range of physical addresses.

  The "TT" attribute beside the last range of physical addresses indicates that it is transparently translated by one of the transparent translation registers. The emulation monitor occupies the first part of the last address range.

  The transparent translation registers were used to provide a 1:1 translation for the monitor because they are much easier to use. The demo program could have created an appropriate entry in the MMU tables to provide the required translation for the emulation monitor.

Step 15: Look at the translation table details for a single logical address

- Choose **Display→MMU Translations ...** In the Display MMU Translations dialog box, select MMU Tables, Address 18000h, and Table Level All. Then click OK.

- Or, using the command line, enter the following command:

  display mmu_translations tables 18000h ***level all***

  The following display should appear. It shows how logical address 18000h is translated through the MMU tables to its corresponding physical address ffff8000h.

```
Logical Address (hex)     0    0    0    1    8    0    0    0
Logical Address (bin)  0000 0000 0000 0001 1000 0000 0000 0000
Table Level            AAAA AAAB BBBB BBCC CCCC PPPP PPPP PPPP

LEVEL INDEX LOCATION   CONTENTS   TBL/PAGE G Ux S CM M U W UDT/PDT
SRP                    00000200   00000200                 RESIDENT
A      000  00000200   0000040b   00000400             y n RESIDENT
B      000  00000400   0000060b   00000600             y n RESIDENT
C      024  00000660   ffff801b   ffff8000 n 00 n cw y y n RESIDENT

Physical Address (hex) = ffff8000
```

  When you are developing a virtual memory system, you will need to check the translations of selected addresses. The **MMU tables** option of the **Display→MMU Translations ...** command lets you do this.

- Try displaying the translation for a non-resident page, such as address 54321h. Also, try using the memory command to access a non-resident page. The monitor always recovers from its own exceptions generated during commands, and displays a detailed error message.

Step 16: Look at details of MMU Table C

- Choose **Display→MMU Translations ...** In the Display MMU Translations dialog box, select MMU Tables, Address 18000h, and Table Level C (Page). Then click OK.

- Or, using the command line, enter the following command:

  display mmu_translations tables 18000h ***level C***

  The following display should appear. It shows the portion of MMU Table C that is used to translate logical address 18000h.

```
Logical Address (hex)     0    0    0    1    8    0    0    0
Logical Address (bin)  0000 0000 0000 0001 1000 0000 0000 0000
Table Level            AAAA AAAB BBBB BBCC CCCC PPPP PPPP PPPP

LEVEL INDEX LOCATION   CONTENTS   TBL/PAGE G Ux S CM M U W UDT/PDT
SRP                    00000200   00000200                 RESIDENT
A      000  00000200   0000040b   00000400             y n RESIDENT
B      000  00000400   0000060b   00000600             y n RESIDENT
C      000  00000600   0000009f   00000000 n 00 y cw n y y RESIDENT
C      001  00000604   00001087   00001000 n 00 y cw n n y RESIDENT
C      002  00000608   00002087   00002000 n 00 y cw n n y RESIDENT
C      003  0000060c   00003087   00003000 n 00 y cw n n y RESIDENT
C      004  00000610   00004087   00004000 n 00 y cw n n y RESIDENT
C      005  00000614   00005087   00005000 n 00 y cw n n y RESIDENT
C      006  00000618   00006087   00006000 n 00 y cw n n y RESIDENT
C      007  0000061c   00007087   00007000 n 00 y cw n n y RESIDENT
C      008  00000620   0000800f   00008000 n 00 n cw n y y RESIDENT
C      009  00000624   00009007   00009000 n 00 n cw n n y RESIDENT
C      010  00000628   0000a007   0000a000 n 00 n cw n n y RESIDENT
```

  Occasionally you will need to examine the content of one of the MMU translation tables at the point where it is used to translate a particular logical address. The Table Level selection in the MMU Translation dialog box lets you do this.

Step 17: Output characters on the seven-segment display

1 Choose **Settings→Command Line** to turn on the command line, if you are using the graphical user interface.

2 Using the command line, store the value of a hexadecimal string to be output on the seven-segment display of the demo board. To see the digits ""0123456789AbCdEF" appear once, enter the following:

```
modify memory sysbuf string to "0123456789ABCDEF"
```

As soon as the operating system of the demo program detects that "sysbuf" has been modified, it starts the user task to display each character in the string for 1/2 second. After the last character has been displayed, the user task returns to the operating system and "sysbuf" can be modified again.

3 To display characters repetitively, add an "@" sign to the end of the string. For example:

```
modify memory sysbuf string to "0123456789ABCDEF@"
```

4 To restart the program after an "@" sign is used, choose **Execution→Run→from Reset**, or from the command line:

```
run from reset
```

Step 18: Take a trace of emulation activity

- Choose **Trace→Everything**, and then choose **Trace→Display**. Choose **Settings→Display Modes ...** In the Display Modes dialog box, select Source in Trace Off, Then click OK. Now on the command line, enter the command:

  display trace absolute status mnemonic

- Or, using the command line, enter the following commands:

  trace
  display trace absolute status mnemonic
  set source off symbols off

  A trace list similar to the following should appear on screen.

```
Trace List                Offset=0
Label:  Address   Data                          Absolute Status
Base:     hex     hex                             mnemonic
after  000008F4 65F24A82   $65F24A82 phy sprog long read
+001   000008F8 6FEA7050   $6FEA7050 phy sprog long read
+002   000008FC B08263E4   $B08263E4 phy sprog long read
+003   000008E8 4A322800   $4A322800 phy sprog long read
+004   000008EC 67085282   $67085282 phy sprog long read
+005   000008F0 7050B480   $7050B480 phy sprog long read
+006   FFFFC010 00000000   $00------ phy sdata byte read
+007   000008F4 65F24A82   $65F24A82 phy sprog long read
+008   000008F0 7050B480   $7050B480 phy sprog long read
+009   000008F4 65F24A82   $65F24A82 phy sprog long read
+010   000008F8 6FEA7050   $6FEA7050 phy sprog long read
+011   000008FC B08263E4   $B08263E4 phy sprog long read
+012   00000900 2D7C0000   $2D7C0000 phy sprog long read
+013   00000904 092AFFF8   $092AFFF8 phy sprog long read
+014   000008E0 FF00588F   $FF00588F phy sprog long read
```

Note that all of the addresses displayed are physical addresses (denoted by "phy" in the "Absolute Status Mnemonic" column of the trace list).

When the analyzer receives physical addresses, it can only show hexadecimal values in the "Address" column of the trace list. The analyzer has no way to cross reference the physical addresses on the emulation bus with the logical addresses from which they were translated. Therefore, the analyzer cannot show you any symbol information associated with these addresses. To see logical addresses in the tracelist, you must use the deMMUer.

Step 19: Prepare the deMMUer so you can see symbolic addresses in the trace list

- Choose **Settings→DeMMUer→**, and then make sure the **Verbose** pushbutton is pressed (to see details on screen). Choose **Settings→DeMMUer→Load from Memory**.

- Or, using the command line, enter the following command:

```
load demmuer verbose
```

A display similar to the following should appear.

```
All physical addresses within the following 32-Mbyte range(s) will be
reverse translated into logical addresses for the analyzer:
000000000..001ffffff@a
0fe000000..0ffffffff@a

The lowest logical address from the translation tables is assumed when
multiple translations reference the same physical address.
```

The above command loaded the deMMUer with information to reverse translate two ranges of physical addresses obtained from the MMU. By default, the deMMUer was enabled when it was loaded. The verbose mode of this command was selected so we could see which ranges of physical addresses would be reverse translated by the deMMUer.

Any physical addresses that might have been derived from two or more logical addresses will be reverse translated to the lowest logical address by the deMMUer.

Remember the setup of the MMU. It showed the following:

```
Logical Address         Physical Address         Attributes
000000000..000007fff    000000000..000007fff@a   S W
000008000..00000ffff    000008000..00000ffff@a   W
000010000..000017fff    0ffff0000..0ffff7fff@a   W
000018000..00001bfff    0ffff8000..0ffffbfff@a
00001c000..00001ffff    0ffffc000..0ffffffff@a   S
0ff000000..0ffffffff    0ff000000..0ffffffff@a   TT
```

Physical address ffff0000H, for example, might appear when the MMU translates either logical address 10000H or logical address ffff0000H. The deMMUer will send 10000H to the analyzer because it is the lowest logical address that might have caused physical address ffff0000H to appear on the emulation bus.

When a physical address maps to two or more logical addresses, the deMMUer normally sends the logical address with the lowest value to the analyzer. Exceptions to this rule are discussed in Chapter 10, "Using Memory Management".

Step 20: Take a new trace

- The purpose of this trace is to see if the analyzer is now capturing logical address information for each state on the emulation bus. Choose **Trace→Everything**, and then choose **Trace→Display**. Choose **Settings→Source/Symbol Modes→Symbols**.

- Or, using the command line, enter the following commands:

```
trace
display trace
set symbols on
```

```
after  sys_sta+0000002C 6FEA7050   $6FEA7050 log sprog long read
+001   sys_sta+00000030 B08263E4   $B08263E4 log sprog long read
+002   sys_sta+00000034 2D7C0000   $2D7C0000 log sprog long read
+003   sys_sta+00000038 092AFFF8   $092AFFF8 log sprog long read
+004   sys_sta+00000014 FF00588F   $FF00588F log sprog long read
+005   sys_sta+00000018 74006008   $74006008 log sprog long read
+006   sys_sta+0000001C 4A322800   $4A322800 log sprog long read
+007   sys_sta+00000024 7050B480   $7050B480 log sprog long read
+008   sys_sta+00000028 65F24A82   $65F24A82 log sprog long read
+009   sys_sta+0000002C 6FEA7050   $6FEA7050 log sprog long read
+010   sys_sta+00000030 B08263E4   $B08263E4 log sprog long read
+011   sys_sta+0000001C 4A322800   $4A322800 log sprog long read
+012   sys_sta+00000020 67085282   $67085282 log sprog long read
+013   sys_sta+00000024 7050B480   $7050B480 log sprog long read
+014    sdata|_sysbuf   00000000   $00------ log sdata byte read
```

Note that "log" is now shown in the "Absolute Status Mnemonic" column. Because the deMMUer is supplying logical addresses to the analyzer, the analyzer is able to replace the hexadecimal addresses with the symbols in the trace list (i.e. sdata|_sysbuf).

Step 21: Inverse assemble the trace list

- Now show the trace list inverse assembled into assembly language mnemonics. Choose **Settings→Display Modes ...** In the Display Modes dialog box, select Source Mixed in **Source in Trace**. Then click OK.

- Or, using the command line, enter the following commands:

```
display trace disassemble_from_line_number 0
set source on inverse_video on symbols on
set source memory_only_trace_on
```

```
after  sys_sta+0000002C 6FEA7050  BLE.B     spr|sys_startup+$0018
      =sys_sta+0000002E           MOVEQ     #$00000050,D0
+001   sys_sta+00000030 B08263E4  CMP.L     D2,D0
      =sys_sta+00000032           BLS.B     spr|sys_startup+$0018
      ##########demo.c - line   248 thru   250
#########################
                       {
                                /* invoke user task to display string */
                                argv[0] = "demo";
+002   sys_sta+00000034 2D7C0000  MOVE.L    #$0000092A,($FFF8,A6)
+003   sys_sta+00000038 092AFFF8   $092AFFF8 log sprog long read
+004   sys_sta+00000014 FF00588F  Unimplemented F-Line Opcode: $FF0
      =sys_sta+00000016           ADDQ.L    #4,A7
      ##########demo.c - line   234 thru   238
#########################

             for (;;)
```

The above display is the trace list format established at power up. When you enter a trace command and use your command to change the trace format, your changes become the new default for the **Display→Trace** command.

Note that symbols are shown in the trace list instead of the hexadecimal address values they represent. You requested that symbols be shown in place of hexadecimal address values when you included the "symbols on" option in the commands above.

Step 22: Reset the emulator

- Sometimes you may want to reset the emulation processor. This may be done from the emulator or the target system. To reset the emulation processor from the emulator, choose **Execution→Reset**

- On the comand line, enter:

  reset

  The Status line will show "M68040--Emulation reset".

  When you apply power to the emulator, the initialization process leaves the emulator in the reset state. Changing some configuration items also resets the processor. (Refer to Chapter 8, "Configuring the Emulator", for more information.)

2

Solving Quick Start Problems

Solutions to problems you might face during the Getting Started procedures.

Solving Quick Start Problems

This chapter helps you identify and resolve problems that may arise while using procedures in Chapter 1, "Getting Started".

For more information, refer to Chapter 9, "Solving Problems".

If the desired emulator interface won't start

☐ Check for correct installation of the interface software. Refer to Chapters 18, 19, and 20 in the "Installation and Service" part of this manual, and to the *HP 64700 Series Emulators Installation/Service Guide.*

☐ Verify that the $PATH environment variable includes the directory containing the interface software (/usr/hp64000/bin). The interface files are loaded in the "/usr/hp64000/bin" directory by the installation procedure.

If the text-based Softkey Interface won't start under X-Windows

☐ If the Graphical User Interface is starting when you are trying to start the Softkey Interface, include the **-u skemul** option to the **emul700** command to override the Graphical User Interface and force the start of the Softkey Interface.

If you can't load the demo program

☐ Check to ensure that the emulator probe is plugged into the demo board, with power connected to the demo board from the emulator. (The demo program may not work with target systems other than the demo board.)

☐ Make sure the reset flying lead is connected from the probe to the demo board.

☐ Check to ensure that you changed to the demo directory:

- /usr/hp64000/demo/debug_env/hp64783 for the MC68040.

If you can't display the program

☐ Verify that the program loaded correctly.

☐ Check to see that the status of the emulator is reset or is running in monitor. See the STATUS line on the display. If the emulator is halted, it can't use the monitor to display program memory. In this case, reset the emulator and try to display the program memory again.

☐ Check the event log by choosing **Display→Event Log,** or by using the **display event_log** command on the command line. If the event log shows that the program loaded, try reloading the program again.

☐ If you are displaying memory with **symbols on**, ensure that the symbol data base has been loaded with the program. If this is the cause of the problem, you will be able to obtain the memory display by referring to the address using its hexadecimal value instead of its symbolic value. For example, to obtain a display of the program in memory at symbolic address demo:main:

Move the mouse pointer to the entry buffer and type in "main"; and then select **Display→Memory→Mnemonic ()**.

or use the command line to enter:

```
display memory main mnemonic
```

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 3, "Using the Emulator/Analyzer Interface," tells you how to use the Graphical User Interface and Softkey Interface commands.

Chapter 4, "Using the Emulator," shows you how to use the emulator/analyzer commands to control the emulation processor and make simple emulation measurements.

Chapter 5, "Using the Emulation-Bus Analyzer," explains how to use the emulation-bus analyzer to record program execution for debugging.

Chapter 6, "Making Coordinated Measurements," tells how to couple two or more emulators to coordinate measurements involving more than one processor.

Chapter 7, "Making Software Performance Measurements," shows you how to use the Software Performance Measurement Tool supplied with the emulator.

Chapter 8, "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 9, "Solving Problems," describes some of the problems that you might encounter when you use the emulator, and shows how to solve them.

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.

3

Using the Emulator/Analyzer Interface

How to enter commands in the Graphical User Interface and the Softkey Interface

Using the interface

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 (SPA) 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.

This chapter shows you how to perform the basic tasks associated with each type of emulator/analyzer interface. The information is grouped into the following sections:

- Starting the emulator/analyzer interface.
- Opening other HP 64700 interface windows.
- Entering commands
- Using special features of the Graphical User Interface.
- Using display-control features of the Softkey Interface.
- Copying information to a file or printer.
- Exiting the emulator/analyzer interface.
- Creating and executing command files.
- Forwarding commands to other HP 64700 interfaces.
- Accessing the terminal interface.
- Accessing the operating system.

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 features such as pulldown and popup menus, point and click setting

of breakpoints, cut and paste, on-line help, customizable action keys, and popup recall buffers.

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, there are some commands that must be entered in the command line.

Maximum Number of Windows

Ten is the maximum number of windows you can use to view HP 64700 emulator/analyzer operation. Only one C debugger interface window and one SPA window are allowed, but you can start multiple emulator/analyzer interface windows.

Activities that Occur in the Windows

When using an HP 64700-Series emulator in a window environment (or with multiple terminals), the following activities occur in the windows where the emulator is currently operating.

Commands Complete in Sequence

When you execute commands that access the emulator (in multiple windows) the first command you specify will complete before the second command begins executing.

Status Line is Updated

When you perform an emulation task in one window that updates the status line, status lines are updated in all other windows where the emulator is operating. The event log is also updated in each window.

Status Line	Meaning
Slow clock	No clock source from the emulated system.
Emulation reset	The processor is being reset from the emulator.
Target reset	The processor is being reset from the emulated system.
Bus grant	The processor has not been granted the bus by the external arbiter ($\overline{BG}$ is not asserted).
Halted	The processor has double bus faulted.
No bus cycles	No bus cycles are occurring.
Running user program	The processor is executing a target (user) program.
Running in monitor	The processor is executing the emulation monitor.
No target power	No power from the emulated system.
Awaiting CMB ready	The emulator is waiting for a CMB READY signal. Refer to Chapter 6
CPU in wait state	The processor is waiting for a cycle termination from the target system.
Unknown state	The emulator is in an unknown state. You will probably need to reset the emulation processor, initialize the emulator, or cycle power to reinitialize the emulator.

Ending the Emulation Session

When you are using the emulator in multiple windows, you can choose to either end the emulation session in a single window, or in all the windows. The **end** command by itself just ends the window where the command is executed.

Using Multiple Terminals

If you do not have a window environment installed on your host computer, you can still obtain the benefits of multiple windows by logging onto the same UNIX system from several terminals, and starting the emulator on each terminal, just as described here for several windows.

The rest of this chapter describes how to start and stop interface instances and sessions in multiple windows.

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 19, "Installation and Service".

This section describes how to:

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

To see emulator/analyzer availability before interface startup

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

The **-lv** option of the **emul700** command provides a verbose listing of all emulators defined in the 64700tab and 64700tab.net files, and shows whether they are already in use, locked, or available. If a logical emulator name (<emul_name>) is included in the command, just the status of that emulator is listed. The verbose option also lists all of the interfaces that can be started.

Examples

To list, verbosely, the status of the emulator whose logical name is "em68040":

```
$ emul700 -lv em68040
```

The information may be similar to:

```
em68040 - m68040 available
    description:      M68040 emulation, w/internal analysis, 260Kb emul mem
    user interfaces:  xemul, xperf, skemul, skperf
    device channel:   /dev/emcom23
```

Or, the information may be similar to:

```
em68040 - m68040 running; user = guest@myhost
    description:      M68040 emulation w/internal analysis, 260Kb emul mem
    user interfaces:  xemul, xperf, skemul, skperf
    internet address: 21.17.9.143
```

The "em68040" 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:      em68040    m68040     21.17.9.143
serial:      em68040    m68040     myhost /dev/emcom23 OFF 9600 NONE XON 2 8
```

To start the emulator/analyzer interface

- Use the **emul700 [-u < user interface>]<emul_name>** command.

If **/usr/hp64000/bin** is specified in your PATH environment variable, 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 the X Window System, the graphical user interface for the emulator/analyzer will start by default. Otherwise, the softkey user interface will start. You can force a particular interface to be used by including the "-u" option and the name of the user interface desired.

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, data in a fourth, and results of software performance measurements in the fifth (if a software performance analyzer is installed as part of your system).

Examples

To start the emulator/analyzer interface for the MC68040 emulator, enter:

```
$ emul700 em68040
```

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 softkey user interface for the emulator/analyzer when the X Window System is running, enter:

```
$ emul700 -u skemul em68040
```

To start the interface using the default configuration

- Enter 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 execute a command file at interface startup

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

Starting a command file (**-c <cmd_file>**) at emulator startup allows you to automate some of the setup and configuration of the emulator. For example, you may have a command file that loads a particular configuration file, a program file, and then sets up trace display formats and specifications.

If the HP64KPATH variable is set, the interface will use the search paths specified in this variable to locate a command file passed to it with the **emul700** command.

Refer to the "Creating and Using Command Files" section later in this chapter for information on creating command files.

Examples

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

```
$ emul700 -c startup em68040
```

where “m68040" is the **logical name** for the HP 64783 MC68040 emulator.

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. You can only use this command if there is no current session in progress.

Examples

To unlock the emulator whose logical name is "em68040", enter:

```
$ emul700 -U em68040
```

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 products for those windows 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*
- To open additional Softkey Interface windows, choose **File→Emul700→Emulator/Analyzer** *under Terminal Windows.*
- Enter the **emul700 <emul_name>** command in another terminal emulation window.

  You have a choice of opening up to nine additional windows, whether they be 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 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.

The **File→Emul700** menu may display other choices if the interface finds other HP 64700 products on the computer.

To open the high-level debugger interface window

- Choose **File→Emul700→High-Level Debugger ...** *under Graphic Windows*.

  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.*

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

Entering Commands

The Graphical User Interface and Softkey Interface provide simple, effective mechanisms for entering commands to be processed by the emulator and analyzer. Basic descriptions of both interfaces are given in Chapter 1, "Getting Started".

This section shows you how to:

- Turn the command line on and off.
- Enter commands on the command line.
- Edit commands.
- Recall commands that were used before.
- Execute a completed command.
- Get online help on commands.
- Display the error log and the event log.

To turn the command line on or off in the Graphical User Interface

- 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 popup menu: position the mouse pointer within the status line area, press and hold the *select* mouse button, and choose **Command Line On/Off** from the menu.

- To turn the command line off using the command line entry area popup menu: position the mouse pointer within the entry area, press and hold the *select* mouse button, and choose **Command Line Off** from the menu.

- To turn the command line on, position the mouse pointer in the main display area and start typing.

The above selections turn display of the command line area on or off. When it is on, the command line is displayed; you can use the softkey pushbuttons, the command return and recall pushbuttons, and the cursor pushbuttons for command-line editing. When it is off, the command line is not displayed; 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; it will 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 command-line input that cannot be supplied any other way.

To enter commands on the command line

- In the Graphical User Interface, successively position the mouse pointer on pushbuttons and click the *pushbutton select* mouse button until a complete command is formed.

- Successively press keyboard function keys corresponding to softkey pushbuttons until a complete command is formed.

- Type in the command you want to use. You must type in the full command name as shown in Chapter 11, "Emulator Commands". This may be different from the label on the corresponding softkey.

- Type in the first few characters of a command name, and press **<Tab>**. The interface will complete the command name automatically. Repeat this process until a complete command is formed.

To edit the command line using the command line pushbuttons on the Graphical User Interface

- To position the cursor at a specific character, place the mouse pointer on the character and click the *select* mouse button.
- 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** on your keyboard to change to insertion mode, and then type the characters to be inserted.
- To replace characters at the cursor position, press the **insert key** on your keyboard to change to replacement mode, and then type the replacement characters.
- To delete characters to the left of the cursor position, press the <BACKSPACE> key on your keyboard.

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

  When moving by words to the left or right, the **Forward** pushbutton becomes gray and provides no function when the cursor reaches the end of the command string. The **Backup** pushbutton becomes gray and provides no function when the cursor reaches the beginning of the command.

To edit the command line using the command line popup menu

- To position the cursor at a specific character, place the mouse pointer on the character and click the *select* mouse button.

- 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 popup menu appears, and then choose **Clear Entire Line** from the menu.

- To clear the command line from the cursor position to the end of the line, place the mouse pointer where you want the clear-to-end function to start. Press and hold the *select* mouse button until the Command Line popup menu appears, and then choose **Clear to End of Line** from the menu.

- To insert characters, position the mouse pointer where you wish to locate the text cursor (or over a non-text area to use the current text cursor location). Press and hold the *select* mouse button to display the Command Line popup menu, and then choose **Position Cursor, Insert Mode** from the menu. Type the characters to be inserted.

- To replace characters, position the mouse pointer where you wish to locate the text cursor (or over a non-text area to use the current text cursor location). Press and hold the *select* mouse button to display the Command Line popup menu, and then 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 point in the command.

To edit the command line using the keyboard

- In the Graphical User Interface, place the mouse pointer either in the display area, on the status line, or in the command line area. Then you can use the following keys to select points in the command line: <Left arrow>, <Right arrow>, <Tab>, <Shift><Tab>, <Insert char>, <Back space>, <Delete char>, <Clear line>, and <CTRL>u.

- Move to the next word on the command line by pressing the **<Tab>** key. Move to the previous word on the command line by pressing the **<Shift><Tab>** key combination.

- Enter more than one command on a command line by separating the commands with semicolons (;).

- Recall previous commands by pressing **<Ctrl>r** to cycle backward, or **<Ctrl>b** to cycle forward through the command line buffer.

- Delete the current command line by pressing **<Ctrl>u**.

- Clear the command line from the cursor position to the end of the line by pressing **<Ctrl>e**.

To recall commands

1 In the Graphical User Interface, click the pushbutton labeled **Recall** in the Command Line area to display the dialog box.

2 Choose a command from the dialog box. (You can also enter a command directly into the Selection 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 13, "Setting X Resources").

See "To use dialog boxes" in this chapter for information about using dialog boxes.

To execute a completed command

- In the Graphical User Interface, click the pushbutton labeled **Return** (near the bottom of the command line area).
- In the Graphical User Interface, position the mouse pointer in the command line entry area; press and hold the *select* mouse button until the Command Line popup menu appears; and then choose the **Execute Command** menu item.
- Press the carriage return key on the keyboard.

To get online help on commands

- To get a dialog box that lists an index of helpful information in the Graphical User Interface, select **Help→General Topic...** or **Help→Command Line...**. Then choose a topic of interest from the Help Index.

- To get specific help about the operation of the command line on the Graphical User Interface, click the **Help** pushbutton located near the bottom, right-hand corner of the Command Line area.

- Get specific help about a command to be entered on the command line, type:

  ```
  help <command_name>
  ```

  The <**command_name**> parameter can be entered from the softkeys after you type **help**.

  You can type a question mark (**?**) in place of the keyword **help**. When you use the **help** command, information about the command you selected (including syntax and sample usage) scrolls onto the screen.

  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.

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

  The last 100 error messages that have occurred during the emulation session are displayed.

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

  The last 100 events that have occurred during the emulation session are displayed.

  The status of the emulator and analyzer are recorded in the event log, as well as the conditions that cause the status to change (for example, breakpoints and trace commands).

Using Special Features of the Graphical User Interface

The following paragraphs show you how to use pulldown and popup menus, the entry buffer, action keys, and dialog boxes to compose commands and control emulator and analyzer operation. These features are only available in the Graphical User Interface.

This section shows you how to:

- Choose a pulldown menu item.
- Choose a popup menu item.
- Place values into the entry buffer.
- Copy and paste from the entry buffer to the command line.
- Use action keys.
- Use dialog boxes.

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.

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 pushbutton), 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.

3 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 pushbutton), 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 an HP 9000 keyboard), and then type the underlined character in the menu label on the menu bar. (For example, "f" for "File". Type the character in 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 carriage 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 13, "Setting X Resources", and the "Softkey.Input" scheme file for more information about setting the X resources that control defining keyboard accelerators.

To choose popup menu items

1. Move the mouse pointer to the area whose popup menu you wish to access. (If a popup menu is available, the mouse pointer changes from an arrow to a hand.)
2. Press and hold the *select* mouse button.
3. After the popup 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 popup 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 several displays of the entry buffer, there is only one entry buffer; it is common to all windows. That means you can copy and paste a symbol or an address seen in 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.

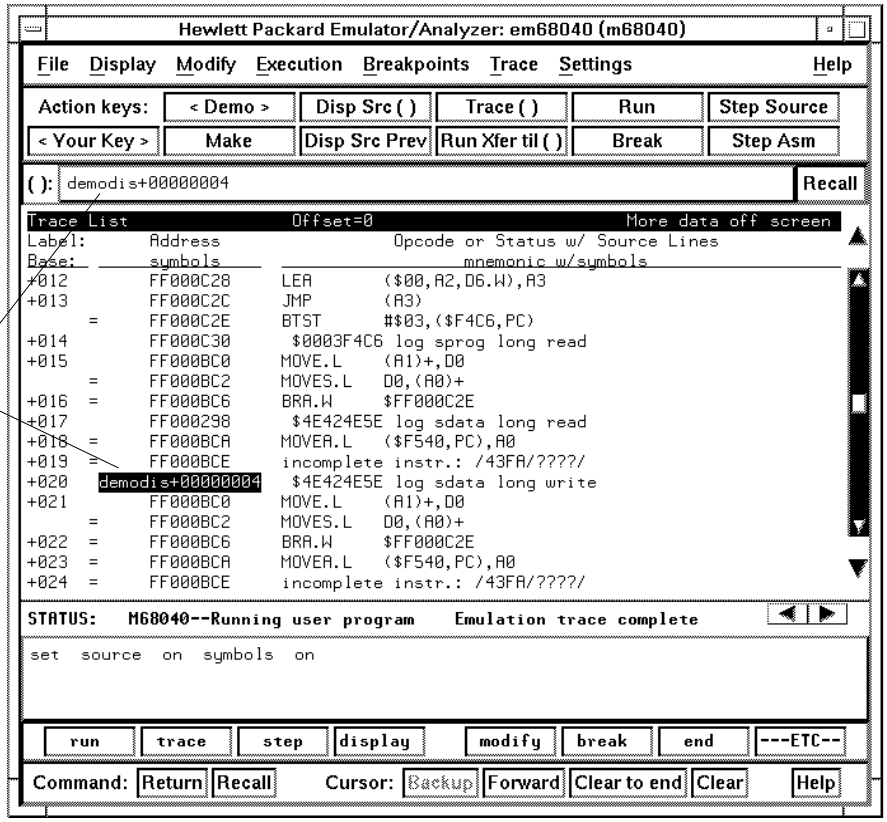
Text pasted into the entry buffer replaces that which is currently there. You cannot use paste to append text to existing text already in the entry buffer.

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 (plus offset) "demodis+00000004" 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 mmutest+0000000C and paste the symbol into the entry buffer.



To recall entry buffer values

- Position the mouse pointer over the **Recall** pushbutton 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 13, "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 (by pressing the <insert> key on the keyboard).

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 required 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. Some of these perform tasks and others show you how to define and use action keys. 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 13, "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 of the dialog box.

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

3 Finally:

- 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.
- Click on the "Cancel" pushbutton to cancel the selection and close the dialog box.

The Graphical User 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.
Settings Display Modes	You can set the display mode and customize the display presentation for memory and trace list displays.
Modify Register	You can view and modify values of any selected register, as well as recalling previous values of the registers.
Symbol Selection	Selects the current working symbol (cws). You can change to a previously accessed cws, a predefined cws, or specify a new cws.

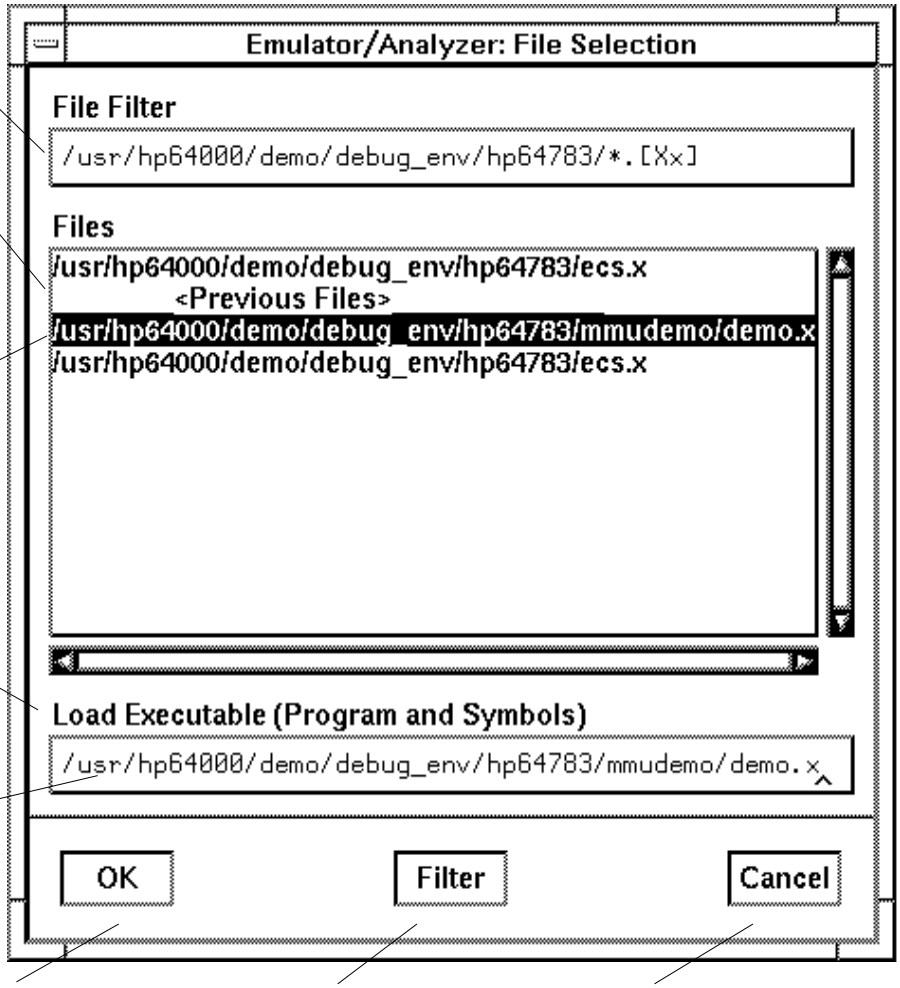
The dialog boxes share some common properties:

- 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 User 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 13, "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.

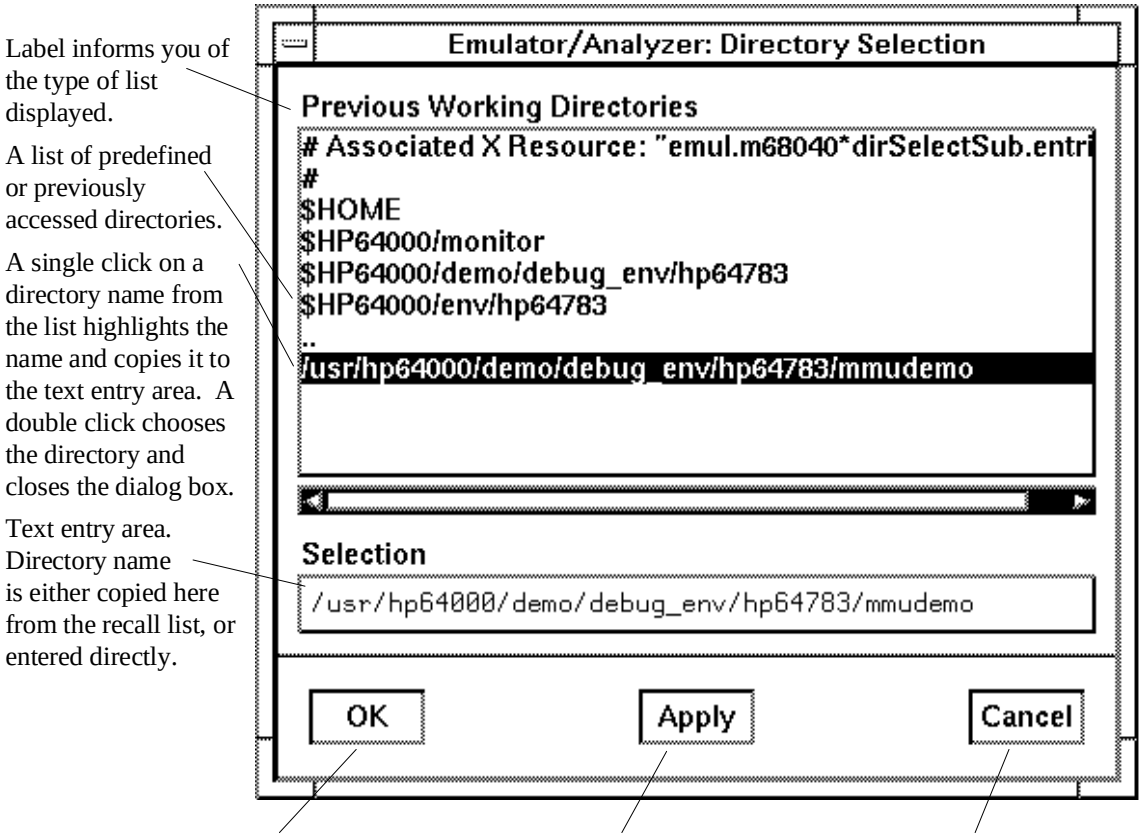
Text entry area. Text is either copied here from the recall list, or entered directly.

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

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

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

To use the Directory Selection dialog box:



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

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

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

Using display-control features of the Softkey Interface

- Use the following control-key combinations to redraw, reposition, and update the display of the Softkey Interface:

Input	Result
<Ctrl>l	To redraw the current display
<Ctrl>f	To roll the display left
<Ctrl>g	To roll the display right
<Ctrl>s	To stop screen updates
<Ctrl>q	To resume screen updates

You can roll the display left and right only if there is more information than will fit into 80 columns.

Copying information to a file or printer

- Choose **File→Copy**. Select the type of information from the cascade menu (see copy options below), and use the dialog box to select the file or printer.

- Using the command line, enter a command such as:

  copy <copy option> ***to*** <destination>

  In the above command, <copy option> has a name similar to those listed below (available through softkey selection), and <destination> is a printer or the name of a file.

  ASCII characters are copied to the file or printer. If you copy information to an existing file, it will be appended to the file. Details of the copy options are discussed in the following paragraphs.

  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.

  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 the event log display.

See the **copy** command syntax in Chapter 11, "Emulator Commands", for more information.

Exiting the Emulator/Analyzer Interface

The following paragraphs show you how to end single instances of the interface in selected windows, and how to exit from the interface and end your session.

This section shows you how to:

- End a single window in the interface.
- End the emulation session in all windows.

To end a single window in the interface

- In the interface window you wish to close, select **File→Exit→Window**.
- If using the command line, in the interface window you wish to close, enter:

  end

  This ends the interface instance in the window where the command is executed. None of the other windows are affected.

  If the window is the only window into the emulation session, the above command ends the emulation session and leaves the emulator in a locked state. Emulators restarted from a locked state will reload the last valid configuration and absolute file.

To end the emulation session in all windows

- To exit all windows, save your configuration to a temporary file, and lock the emulator so it cannot be accessed by others, select **File→Exit→Locked**.

  If using the command line, enter:

  ```
  end locked
  ```

- To exit all windows and release the emulator for use by others, select **File→Exit→Released**.

  If using the command line, enter:

  ```
  end release_system
  ```

  If you exit 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 you return to the configuration you were using when you quit.

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

  In contrast, if you end released, all changes you made to your configuration are lost. You may want to save your current configuration to a configuration file before you end released.

Creating and Executing Command Files

A command file is an ASCII file containing command-line commands. The interface can read a command file and execute the commands found there as if the commands were entered one-by-one on the command line. Command files can, in turn, call other command files. The interface will execute the called file like a subroutine of the calling file.

You can create command files from within the interface by logging commands to a command file as you execute commands. You can also create a command file outside the interface with an ASCII text editor. Logging commands from the command line has the advantage of making sure the commands are syntactically correct when they reach the command file. Syntactically incorrect commands found by the interface will cause it to halt execution of a command file.

With a single command file, you can automate a complete test procedure. For example, you could start the interface and then execute a command file that would perform the following steps:

1 Load a configuration file.
2 Load an absolute file.
3 Modify registers or memory locations.
4 Set up a trace specification.
5 Start the program running.
6 Capture a trace.
7 Save the trace listing to a file.

Command files are also useful for saving very complex trace specifications so that they can be used again during another emulation session, or by other people.

Passing Parameters to Command Files

Command files can accept parameters. Parameters are like variables in the command file and are usually used in place of explicit arguments to interface commands. A command file that accepts parameters can be made more general than a command file containing explicit argument values and can apply to a wider range of uses.

Parameters can be passed in either of two ways. You can pass the parameters on the command line when you execute the command file, or you can execute the command file without the parameters and let the interface prompt you for the parameters.

Parameters must be declared at the beginning of a command file using the **PARMS** keyword. Parameters are preceded by an ampersand (&) and consist of a combination of one or more letters or underscores. Letters may be upper-case or lower-case.

Using &ArG_lEfT in Command Files

A command file may contain a special argument named **&ArG_lEfT**. This special argument does not have to be declared using the **PARMS** keyword. It can be used in command files containing other parameters, or in command files that do not contain any parameters. This special argument can accept the union of zero or more command line arguments as a single argument. See “To increase flexibility of command files by using &ArG_lEfT” for more information.

Using UNIX Commands and Scripts with Command Files

Command files may include UNIX commands and may call shell scripts. Some commands are recognized directly by the interface (**pwd**, for example) while others require a preceding exclamation point (!) to identify them as shell commands.

Using Shell Variables with Command Files

Command files may contain shell variables. Command files only support shell variables beginning with “$”, followed by an identifier. An identifier is composed of an underscore or a letter followed by zero or more letters, digits, or underscores. Identifiers may follow the “$” symbol directly, or follow the “$” enclosed in braces “{}”. An identifier *must* be enclosed in braces if any letter, digit, or underscore that is not part of the identifier immediately follows the identifier. Otherwise, the following text will be interpreted as part of the identifier. You can examine any shell variables defined for your environment by using the UNIX **env** command. Positional shell variables, such as $1, $2, and so on, are not supported. Neither are special shell variables, such as $@, $*, and so on.

To illustrate how shell variables work, consider the shell variable “S”, defined to be the string “soft”. Suppose you wanted to use the shell variable to reference the directory “/users/softkey”. The reference “/users/${S}key” would produce the desired directory name. However, the reference “/users/$Skey” would cause the shell variable “Skey” to be searched for.

Restrictions on Commands

There are certain commands that you cannot execute from a command file. These are commands that require a response from you. For example, you cannot place modify configuration commands in a command file because the command file cannot “respond” to configuration questions.

Another restriction has to do with calling a command file from an executing command file when the called command file requires parameters. You must supply the parameters with the call to the command file, or the calling command file will abort. That is because the calling command file cannot respond to the called file’s parameter prompts.

Status Line Updates

The emulator status line is not always immediately updated with new status information when the interface executes commands from a command file. You may have to explicitly display the emulator status after a command file has executed by issuing a **display status** command.

Nesting Command Files

You can call other command files from an executing command file. Called files can, in turn, call other command files. This nesting of calls can continue to a maximum of eight levels. Command files called from an executing command file are executed like subroutines of the calling file. Control returns to the calling file after the called file has executed.

Pausing Command Files

You can use the **wait** command in command files. This allows you to pause execution of the command file between commands.

A variation of the wait command, the **wait measurement_complete** command, should be used after starting a trace. Use this command so that a copy or display command following a trace command will not execute until states from the new trace are available for copy or display.

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.

Placing Comments in Command Files

As with any source file, comments in command files can help to explain the operation of the command file, and record creation and modification information. You can place comments in a command file either by using a text editor or by entering the comment as a "command" in the interface command line while logging commands. A special character, the pound sign (#), causes the interface to ignore comments in command files, and also allows you to log comments to a command file from the command line. A comment may appear on a line by itself, or it may *follow* a command on a line. Commands cannot appear on a line after the comment character because they will be interpreted as part of the comment.

Continuing Command File Lines

You can continue command file lines across several physical text file lines. This is done by using a continuation character.

The continuation character is the backslash (\) character. Placing a backslash at the end of the line just before the line feed causes the following line to be concatenated with the current line. Multiple lines can be concatenated by ending all but the last line with a backslash. The concatenated lines will be treated as a single command line. Note that if you end the last line of a command file with a backslash, the command will appear in the interface command line, but will not be executed.

Specifying a Search of Several Command File Directories

HP64KPATH is a special shell variable you can set to specify alternative search paths for command files. HP64KPATH works much like the UNIX PATH in that you can specify several directories, separated by colons (:), to be searched.

The remainder of this section lists the tasks associated with creating and using command files.

To create a command file by logging commands

1 Select **File→Log→Record...** and use the dialog box to select a command file name. If using the command line, enter the command:

```
log_commands to <filename>
```

2 Enter and execute commands to complete the desired task.

3 Stop logging commands by selecting **File→Log→Stop**. If using the command line, enter the command:

```
log_commands off
```

The above commands provide a mechanism that logs commands, entered and executed at the command line, to a file. Later, the command file can be executed by the interface. The **log_commands** command does not appear on the softkeys. Type it on the command line, or type the first few letters of the command and then press <TAB>.

All commands entered on the command line after you type **log_commands to <filename>** are logged to the <filename> until either the **log_commands off** command is used or the interface is exited.

<filename> is any valid UNIX file name. File names may include path information. If <filename> already exists, commands are appended to the current contents of the file, unless the **noappend** option is used. If <filename> does not exist, a new file is created.

File creation errors can sometimes be caused by write permission violations of either files or directories. If you are having trouble creating a command file, make sure you have the correct permissions.

Example

To save a set of commands in the file STARTEMUL by logging commands while executing them during an emulation session, enter the following commands in the command line:

```
log_commands to STARTEMUL
# You can add a comment to a file while logging if you
# precede the comment with a pound sign.
# The Softkey Interface will
# ignore the rest of the line up to the line feed.
load configuration bigproject/config
load bigproject/program
trace after START
run from 2000h # Comments can follow on the same line
```

```
wait measurement_complete
# The preceding wait command variation ensures that
# new trace states will be available in the trace buffer
# before the "display trace" command is executed.
display trace
log_commands off
```

To create a command file by using a text editor

- Use a text editor to create the command file.

  A command file is a text file containing commands in the form that appear on the command line. You can create command files with an ASCII text editor, such as **vi**.

  Make sure that the commands you create in your command file are syntactically correct. Syntactically incorrect command lines will halt command file execution.

To execute (or playback) a command file

- To execute a command file at interface startup, use the **-c <command file name>** option with your **emul700** command.

- To execute a command file from within the Graphical User Interface, select **File→Log→Playback** and use the dialog box to select the name of the command file you wish to execute.

- To execute a command file using the command line, enter the name of the command file and press the carriage return key.

  Any name entered on the command line that is not recognized as a member of the emulator/analyzer command set will be treated as the name of a command file. Command file names may be preceded by directory paths.

  If the command file name does not have a directory path prefixed to it, the interface will search for it as follows:

  - If the environment variable **HP64KPATH** is set, the interface will first search in all directories listed in the HP64KPATH variable. If the interface does not find the command file in those directories, it then searches the current working directory for the command file.
  - If the environment variable **HP64KPATH** is *not* set, the interface searches only in the current working directory for the command file.

  If the command file name has a path name prefixed to it, the interface will only look in the specified path for the command file.

  To interrupt execution 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, <CTRL>c will terminate the "wait" command, but will not terminate command file execution; in this case, you must press <CTRL>c again.

  See “To specify the order of searching several command file directories (HP64KPATH)” for more information about the **HP64KPATH** variable.

Examples

Suppose you have a command file named STARTEMUL, it is located in your current working directory, and it contains the following commands:

```
log_commands to STARTEMUL
load configuration bigproject/config
load bigproject/program
trace after START
run from 2000h
wait measurement_complete
# The preceding wait command variation ensures that
# new trace states will be available in the trace buffer
# before the "display trace" command is executed.
display trace
log_commands off
```

If you start the emulation session and enter STARTEMUL (the command file name) in the command line, all commands from **load configuration...** to **display trace** will be sequentially executed on the command line.

To nest command files

- Call a command file from an executing command file by including the command file name in the executing command file.

The emulation/analyzer interface executes commands found in a command file just as if they were entered into the command line. That means if the interface encounters a command that is not part of its own command set, it will attempt to execute it as a command file. (See "To execute a command file" in this section for an explanation of command file execution.)

Command files called from other command files may be nested to a maximum of eight levels. Control returns to the calling command file after the called command file is executed. A called command file is like a subroutine of the calling command file.

Command files requiring parameters must have those parameters supplied by the calling command file as part of the call. Failure to supply the parameters causes an error and a halt of the calling command file.

Example

The first command file (named "cmdfile1") calls the second (named "cmdfile2") and then executes a single instruction after control returns.

cmdfile1:

```
cmdfile2
display memory
```

cmdfile2:

```
load configuration democfg
load demo
```

To pause command file execution

- To pause execution of a command file until the SIGINT (<CTRL>c) signal is received, use the **wait** command.
- To pause execution of a command file for a specific amount of time, use the **wait <time>** command, where **<time>** is in seconds.
- To pause execution of a command file until a trace trigger has been found and the trace buffer is filled, use the **wait measurement_complete** command.

  You may want to add a delay to a command file under certain conditions. For example, you may want to execute a command file up to a certain point, have it display a screen, and then pause while you examine the output on the screen.

  Place **wait measurement_complete** in your command file following a trace command to ensure that the trace completes before command file execution continues. This ensures that subsequent trace display or trace copy commands use the new trace states, not states from a previous trace.

A **wait** command without parameters will cause execution to pause until the <CTRL>c key combination is entered. If you have a command file that is hanging on a **wait** command, check to make sure that the wait has a **<time>** or a **measurement_complete** argument.

Examples

Pause a command file for 5 seconds by placing the following command in the command file:

```
wait 5
```

Pause a command file until a trace trigger has been satified and the trace buffer has filled by placing the following command after a trace command in a command file:

```
wait measurement_complete
```

To add a comment to a command file

- Use a pound sign (#) to precede the comment string.

You can use this technique either while logging commands to a file during an emulation session, or when you are creating a command file with a text editor. Any text that follows the comment character, up to the next new line, is ignored by the interface. Comments may appear on lines by themselves, or comments may *follow* commands on the same line.

Example

Two variations of comments are shown in the following command file fragment:

```
# The next command is the default trace command
trace
wait measurement_complete # make sure the trace buffer
# has new states
display trace
```

To pass parameters to a command file

1 Define formal parameters on the first line of the command file following the **PARMS** keyword.

2 Pass actual parameters to the command file when it is executed.

A *formal parameter* is composed of an ampersand (&) followed by one or more letters or underscores. Formal parameters are like variables in the command file. Formal parameters are replaced by actual parameters when the command file is executed.

An *actual parameter* is an ASCII string that represents a symbol or value. Actual parameters containing blanks must be enclosed in single or double quotes.

Actual parameters are supplied to the command file in two ways.

- As arguments to the command file entered on the command line along with the command file name. Values are *positional*. Enter a value for the first parameter that follows the PARMS keyword in the command file immediately following the command file name on the command line. Enter a value for the second parameter second after the command file name. And so on.
- In response to prompts from the interface. If a formal parameter exists in the command file and no actual parameter was passed to it on the command line, the interface will prompt you for a value for the formal parameter. If you enter a command file name without supplying *any* actual parameters, the interface will prompt you for values for *all* the formal parameters.

You may use either method to supply parameters, or a combination of the two. Being prompted for the parameters relieves you from having to remember the parameters.

If, from another command file, you call a command file that requires parameters, you must supply all the parameters with the call. The calling command file cannot respond to parameter prompts; an error will occur and the calling command file will halt.

Examples

The following command file, called “loadany,” is a general command file for loading a configuration file and then an executable file.

```
PARMS &cfgname &binfile
load configuration &cfgname
load &binfile
```

The following command, entered on the command line, calls the command file "loadany" and passes the actual parameters needed by the command file:

```
loadany democfg demo
```

You could start the command file "loadany" without parameters and allow the interface to prompt you for the actual parameters. Issue the command:

```
loadany
```

and then respond to the parameter prompts. A prompt for the "cfgname" parameter for this command file will look like the following:

```
STATUS: M68040--Running in monitor------------....
Define command file parameter [&cfgname]
```

You might also start the command file, supply just the first parameter, and have the interface prompt you for the second parameter. Issue the command:

```
loadany democfg
```

to cause the interface to prompt you for the second parameter (&binfile).

To increase flexibility of command files by using &ArG_lEfT

1 Use the special parameter **&ArG_lEfT** anywhere in the command file.

2 Pass zero or more arguments to the command file on the command line.

You can create highly flexible command files using the special parameter **&ArG_lEfT**. It must be entered with the preceding ampersand (&) and exactly in the combination of upper and lower case letters shown here. It is not a parameter in the sense of command file formal and actual parameters. (See "To pass parameters to a command file" for more information.) Instead, it is a special parameter that may be included in either a command file with formal parameters or in a command file without formal parameters.

When the interface finds **&ArG_lEfT** in a command file, it replaces it with the union of all arguments remaining in the string of arguments passed to the command file. Arguments for this special parameter must be passed on the command line and can be zero or more in number. The interface *will not* prompt for a value for **&ArG_lEfT**. If you do not pass any values, the interface removes the special parameter and executes the command associated with the special parameter without any arguments.

Example

The following three commands are all variations of the **display memory** command:

```
display memory
display memory 1000h
display memory 1000h, 2000h thru +20h, 3000h
```

The first command displays memory in the format specified by the last memory display command. The second command displays memory at address 1000h in blocked word format. The third command displays memory at two specific memory locations and also from a range of locations all in a single blocked word display.

The following command file (consisting of one line), called "dm," can be used to implement all three commands:

```
display memory &ArG_lEfT
```

The following three command-file invocations replicate the three separate commands:

```
dm
dm 1000h
dm 1000h, 2000h thru +20h, 3000h
```

To specify the order of searching several command file directories (HP64KPATH)

- Set the environment variable **HP64KPATH** to one or more alternative directory paths. Separate each path from the others with a colon (:).

You can set the environment variable HP64KPATH to specify alternative directories for command files. If this variable is set, the interface searches each path listed in the variable successively until the command file it is searching for is found or no more paths exist. If the command file has not been found after this search, then the interface looks in the current working directory for the command file. If this variable is not set, the interface only searches the current working directory.

This variable is typically set to point to a common directory of command files that might be used by several people. You could also use this variable so that you would not have to store command files in the same working directory as, say, source files for a project.

The directories listed in the HP64KPATH variable are *not* searched if the command file has an explicit path name prefixed to it.

Use **set** to specify or change this variable if you are using the command line. Use **export** to set this variable from your HP-UX *.profile* file.

Examples

Set this variable, from within the interface, to cause the interface to search first the "/users/common/cmdfiles" directory, then the "/users/myid/cmdfiles" directory, and then the current working directory, by issuing either of the two following **set** commands:

```
set HP64KPATH=/users/cmdfiles:/users/myid/cmdfiles

or

set HP64KPATH=/users/cmdfiles:/users/myid/cmdfiles:.
```

Force the current working directory to be the first directory searched instead of the last directory searched by including the dot symbol as the first directory in the HP64KPATH, as in

```
set HP64KPATH=.:/users/cmdfiles:/users/myid/cmdfiles
```

By making the current directory the first in the path, you speed up command file access for command files in the current working directory because the interface would otherwise search the current working directory only after searching all of the other directories listed in HP64KPATH.

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. All interfaces having software versions above 5.00 may forward commands; only Graphical User Interfaces can receive forwarded commands. Commands are forwarded using the **forward** command available in the command line. The general syntax is:

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

This section shows you how to:

- Forward commands to the 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"
```

Or, since only the capitalized key is required:

```
forward debug "P R"
```

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"
```

Accessing the Terminal Interface

The Terminal Interface is the name given to a primitive command set that resides in the emulator firmware. The Terminal Interface is described in the *MC68040/EC040/LC040 Emulator/Analyzer Terminal Interface User's Guide.* You may sometimes need to use Terminal Interface commands during an emulation session. For example, you must use a Terminal Interface command to run the emulator's internal performance verification (**pv**) test.

You can access the Terminal Interface of the emulator directly through *pod commands* in your high-level interface (Graphical User Interface or Softkey Interface). The high-level interface provides a screen to display Terminal Interface output and two ways to use the keyboard to input Terminal Interface commands.

Terminal Interface commands bypass the high-level interface and are executed directly by the emulator firmware. For that reason, the high-level interface can become out-of-sync with the emulator if you use certain Terminal Interface commands. Changing configuration items, for example, will cause the actual state of the emulator to be different from the internal record of the state of the emulator that is kept by the high-level interface. Changing communications parameters can prevent the high-level interface from communicating further with the emulator, and cause abnormal termination of the interface. Be careful when using Terminal Interface commands to avoid creating problems for the high-level interface. The following table lists some Terminal Interface commands to avoid, and why:

Commands	Reasons to Avoid
stty, po, xp	Do not use. Will change the channel operation and hang emulator.
echo, mac	Usage may confuse the channel protocol.
wait	Do not use, will block access to emulator.
init, pv *	Will reset emulator and force **end release_system**.
t	Do not use. Will confuse trace status polling and unload.

**Performance verification (pv)* is an internal self-test of the emulator hardware. If you suspect any problems with your emulation system hardware, use the Terminal Interface command "pv" to run the internal self-test. pv is on this list of pod commands to avoid because running it will reset the emulator and end the emulation session. That does not mean you should not run pv if you suspect

hardware trouble. Just be aware that it will terminate the emulation session if you do run it.

See the *MC68040/EC040/LC040 Emulator/Analyzer Terminal Interface User's Guide* for more information about the Terminal Interface.

The remainder of this section explains how to display the Terminal Interface screen, copy the Terminal Interface screen to a file, and enter Terminal Interface commands.

To display the Terminal Interface screen

- Select **Display→Pod Commands**. If you are using the command line, enter:

  display pod_command

  The interface will accept Terminal Interface commands, but will not show the results (output) of those commands unless the Terminal Interface (pod command) screen is displayed. Generally, you display this screen before entering one or more pod commands.

To copy the Terminal Interface screen contents to a file

- To append the contents of the Terminal Interface screen to the contents of a file, select **File→Copy→Pod Commands ...**, or if using the command line, enter:

  copy pod_cmd to <filename>

- To replace the contents of a file with the contents of the Terminal Interface screen, on the command line enter:

  copy pod_command to <filename> ***noappend***

  You can save the current contents of the Terminal Interface screen to a file by using the copy command. Additionally, you can copy the Terminal Interface screen to a printer or to a UNIX command by using other copy command options. Refer to Chapter 11, "Emulator Commands", for more information about copy command options.

  <filename> is any valid UNIX file name. The file name may include path information. If the file does not exist, the interface creates it. File creation errors can sometimes be caused by UNIX permission violations on files or subdirectories. Make sure you have write permission on the file and on the directory where you intend to create the file.

To enter Terminal Interface commands

1 To execute just one or two Terminal Interface commands, enter the Terminal Interface command, enclosed in double quotes, as an argument to **pod_command** on the command line.

or

2 If you expect to enter several Terminal Interface commands, enable Terminal Interface command pass-through and disable high-level interface command processing by selecting **Settings→Pod Command→Keyboard**, or on the command line, enter **pod_command keyboard**.

3 Enter the desired Terminal Interface commands.

4 End Terminal Interface command pass-through and re-enable high-level interface command processing by pressing the **suspend** softkey.

Before you enter a Terminal Interface command, you should use the **display pod_command** command to display the Terminal Interface screen. If you do not display the Terminal Interface screen, you cannot see the output from the Terminal Interface commands you enter.

If you are entering a single Terminal Interface command, the **pod_command "<command>"** variation is useful. However, entering a series of pod commands is easier if done from the **keyboard**. While **keyboard** entry is in effect, the interface passes all keyboard input through to the Terminal Interface. The Terminal Interface validates and executes all commands directly and displays the results on the Terminal Interface screen.

Examples

Access the Terminal Interface and display memory locations 0 through 20 in long word format:

```
display pod_command
pod_command "m -dl 0..20"
```

Access the Terminal Interface from the command line, check the emulator status, then the trace configuration, and finally return keyboard control to the command line:

```
display pod_command
pod_command keyboard
es
tcf
suspend
```

To get help on Terminal Interface commands

1 Select **Display→Pod Commands**. If you are using the command line, enter:

```
display pod_command
```

2 On the command line, enter:

```
pod_command "help <cmd_name>"
```

<cmd_name> is the Terminal Interface command on which you want to receive help.

You can access the emulator's low-level Terminal Interface using the **pod_command** keyword. If you need help on any Terminal Interface command, you can use its **help** command.

See the *MC68040/EC040/LC040 Emulator/Analyzer Terminal Interface User's Guide* for more information regarding the Terminal Interface.

Examples

Get help on the Terminal Interface **cf** command:

```
display pod_command

pod_command "help cf"
```

Get help on all Terminal Interface command groups:

```
display pod_command

pod_command "help *"
```

Accessing the Operating System

Through the command line, you can access the operating system to use services available there. You can set environment variables and enter UNIX commands.

This section shows you how to:

- Set environment variables.
- Enter UNIX commands.
- Display the name of the emulation module.

To set environment variables

- Type **set <ENVIRONMENT VARIABLE>=<VALUE>**.

You can set UNIX environment variables with the **set** command. The **<ENVIRONMENT VARIABLE>** can be any UNIX environment identifier name. The **<VALUE>** can be any string value. If the value has embedded spaces, use double quotes around the string.

Example

To set the PRINTER environment variable to **lp -s**, enter:

```
set PRINTER = "lp -s"
```

To enter UNIX commands

- Type **!<UNIX_COMMAND>**
- Type **!<UNIX_COMMAND>! <options>**

You can execute any UNIX command by preceding it on the command line with an exclamation mark (!). The system creates a shell process and executes the command line string following the exclamation mark.

If you enter only the exclamation mark (!), a command process is created and the command shell is started. Exiting the command shell by typing **exit** returns you to the emulator/analyzer interface.

You can precede and follow your UNIX command with exclamation marks. This allows you to include options with your command, such as:

in_browser	executes your UNIX command and provides results in a scrollable window instead of a terminal window (default display).
wait_for_exit	waits for your UNIX command to finish its execution before allowing the next command to begin. This is useful for commands that require extra time to complete, such as "make".
no_prompt_ before_exit	used to speed operation when you don't need the results display. For example, this will complete a command without prompting for the press of the RETURN key in commands that would normally prompt for the RETURN key.

Examples

Show the values of the current environment variables:

```
!set|more
```

Edit a command file previously created with the command:
log_commands to <FILENAME>

```
!vi <FILENAME>
```

See a directory listing in a browser window:

```
!ls! in_browser
```

Make files in a directory and hold off loading the executable file until the "make" has finished:

```
!make! wait_for_exit; load <file>
```

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
```

The name of the emulation module is displayed on the status line.

4

Using the Emulator

How to control the processor and view system resources

The Emulator And Its Applications

The HP 64783 emulator helps you test and debug applications in real time. The emulator is a functional replacement for MC68040 microprocessor. It provides access to processor registers and memory, as well as complete execution run control.

The emulator provides debugging capability for:

- embedded system hardware startup and test
- hardware/software integration

The emulator may also be used out-of-circuit as a code execution environment. However, software development and testing will probably be handled best in a host execution environment.

The emulation system is connected to a host computer by a LAN connection, or by a serial (RS-232C) data communication link.

The emulation interface can display data and symbolic assembly code in windows, or can additionally show the C-language source code intermixed with the assembly code. You can start and stop execution of application code using the **run** and **step** commands. Breakpoints can be placed at strategic locations to stop application execution when a specific address is reached.

An application can communicate directly with the emulation interface by using the simulated I/O library. This provides standard input and output, messaging and access to the UNIX file system. During initial stages of development, an application can print status and debugging messages to the emulation display using simulated I/O. Files can be created, opened, read from, and written to on the host system. These routines can be converted as target system hardware becomes available.

The demo Application

A demo program, and an associated configuration file, are provided with the emulator. The demo application allows you to learn about the emulator without the bother of writing and loading your own program.

The standard program was written in MC68000 assembly language. When the emulator loads the program, it also defines a symbol table containing symbols from the program. You can use these symbols when you're making measurements using the program.

The demo program emulates a hypothetical environmental control system for a computer room. The name of the demo program is ecs.x.

For detailed information about the files used in the demo program, and methods and requirements for starting the demo program, read the READMEDEMO and README files in the directory named /usr/hp64000/demo/debug_env/hp64783.

To load and run the complete demo program, your emulation system must have at least 256K of emulation memory (obtained by installing at least one SRAM on the emulation probe. Refer to Chapter 19, "Installation and Service", at the end of this manual for instructions on how to install SRAM memory modules.

To build programs

1 Create source files in "C" or MC68040 assembly language using a text editor.

2 Translate the "C" source files to relocatable object code using a compatible C cross compiler.

3 Translate the assembly source files to relocatable object files using a compatible MC68040 cross assembler.

4 Link all relocatable object files with the linker/loader to produce an absolute object file in the IEEE-695 format. (The loaders for the HP language tools produce a file with the extension .x for IEEE-695 format.) If you want to produce an absolute file in the HP64000 format, specify the appropriate loader options. (The IEEE-695 format is better for emulation tasks.)

5 (Optional) Build an SRU symbol database before entering emulation by entering the **srubuild <absfilename>** command.

If you're planning to load programs into emulation or target system memory, you need to have your files in a format acceptable to the MC68040 emulator. Usually, this means that you'll want your files in IEEE-695 absolute format. The HP language tools for the HP 9000 produce this format.

Processor	C Compiler	Assembler
MC68040	HP B1463	HP B1465

You may use other language tools if they produce either IEEE-695 or HP64000 absolute file formats.

Other file formats, such as Motorola S-records and Tektronix hex format can be converted to HP64000 format by using the HP 64888 utility software.

To configure the emulator

- Configure the emulator to meet the resource needs of your target system and application program by following the instructions in Chapter 8, "Configuring the Emulator".

- To configure the emulator, choose **Modify→Emulator Config ...**. Then answer the questions that appear in the Emulator Configuration dialog box.

- Using the command line, enter:

  modify configuration

  This starts a series of questions whose answers define the emulator configuration.

  You must configure the emulator to allocate system resources such as memory, and to set handling of interrupts. You must do this before you load and execute programs and make emulator measurements. Refer to Chapter 8, "Configuring the Emulator".

  If you want to use the examples in this manual, you must load a special configuration file and load the demo program. See "To load the demo program" in this chapter for more information.

Loading and Storing Programs

The emulator provides commands that allow you to move files into emulation or target memory from a host computer through the LAN or serial ports of the HP 64700 Card Cage. You can also save a range of memory in an absolute file for later reuse. (You might do this if you patch a section of code and need to do further testing.)

Many different absolute file formats are supported. The primary ones used with the emulator interface are the IEEE-695 and HP64000 absolute formats.

The **load** command has other options that allow you to control the load process. Refer to the **load** command syntax in Chapter 11, "Emulator Commands".

To load a program

- Choose **File→Load→Executable...** In the dialog box, click on the name of the executable file to load, and then click OK.

- Using the command line, load a program absolute file into emulation or target memory by entering **load [<memory_type>] <filename> [fcode <fcode>]** .

  <memory_type> is optional. **emul_mem** is emulation memory and **user_mem** is target system memory. The default is to load all memory.

  <filename> is the name (including paths if needed) of an HP64000 or IEEE-695 format absolute file. You do not need to specify the extension if it is **.x** or .**X**.

  <fcode> is an optional function code from the following list.

<fcode>	Meaning
none	emulator load, defaults to supervisor space
super	supervisor address space
user	user address space

The emulator can load HP64000 or IEEE-695 format absolute files into emulation or target system memory. So, you can develop programs on your UNIX workstation; then build the programs and load them into the emulator for debugging.

Use the memory type parameter if you want to load only the parts of the program that have addresses corresponding to those types of memory in the map.

Example

To load the executable part of your absolute file into memory and any symbolic information found in the absolute file, choose **File→Load→Executable...**

To load the executable part of your absolute file into memory but not load symbolic information found in the absolute file, choose **File→Load→Program Only...**

To load only the symbolic information found in the absolute file (without loading the executable part of your absolute file), choose **File→Load→Symbols Only...**

Suppose you are using two MMU mappings, one of which is user space from 1000 through 1fff hex. The other is supervisor space from 1000 through 1fff hex. You have absolute files called userprog.x and supprog.x. To load these programs using the command line, enter:

```
load userprog fcode user
load supprog fcode super
```

The programs are loaded into the correct address space.

To load the demo program

1 With your emulator interface *not* on screen, enter the following commands in a terminal window:

cd /usr/hp64000/demo/debug_env/hp64783
Startemul <logical name>

Where <logical name> is the name assigned to your emulator. The default logical name for the MC68040 emulator is **m68040**. For a detailed discussion of how to find a logical name, refer to Chapter 1, "Getting Started".

2 The terminal window will ask you if you wish to copy the demo files to a different directory. It is best to answer "y" to this question, and then supply the full path name of your own demo directory.

The demo files will be copied and modified, as required, into the directory you specify, and then the emulator interface will appear on screen. It will be ready for you to run the demo procedure.

3 Press the Action keys from left to right and top to bottom to see the demo.

The demo program supplied with the MC68040 acts as a hypothetical environmental control system for a computer room. You can use this program to learn more about the emulator. Refer to the information on the demo program in the reference part of this manual.

The examples in this manual use the demo program. To make the examples work correctly, you must load the demo emulator configuration file and demo program as described above.

To store a program

- Using the command line, transfer a range of memory locations from the emulator to an HP 9000 file by entering the command:

```
store memory [fcode <fcode>] <expression> [thru
<expression>] [offset_by <offset>] to <filename>
```

<fcode> is an optional function code as follows:

<fcode>	**Meaning**
none	emulator store, defaults to supervisor space.
super	supervisor address space
user	user address space

<expression> specifies the starting (and ending, in **thru** **<expression>**) addresses of the memory range to be stored.

<offset> optional value to be subtracted from **<expression>**.

<filename> is the name (including paths if needed) of a file to store the data.

If you patched a program or data structure by modifying memory, you may want to save the memory image for comparison with other changes or for future testing. The **store** command allows you to do this.

The **store** command creates absolute files in HP64000 format. The **.X** extension is added automatically.

Example

To save the memory locations of the init_system routine in an absolute file named new, use the command line to enter:

```
store memory init_system thru init_system end to new
```

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 the address reference (either absolute or symbolic) in the entry buffer; and 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 popup 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 popup 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 message if it cannot find a source file for the address.

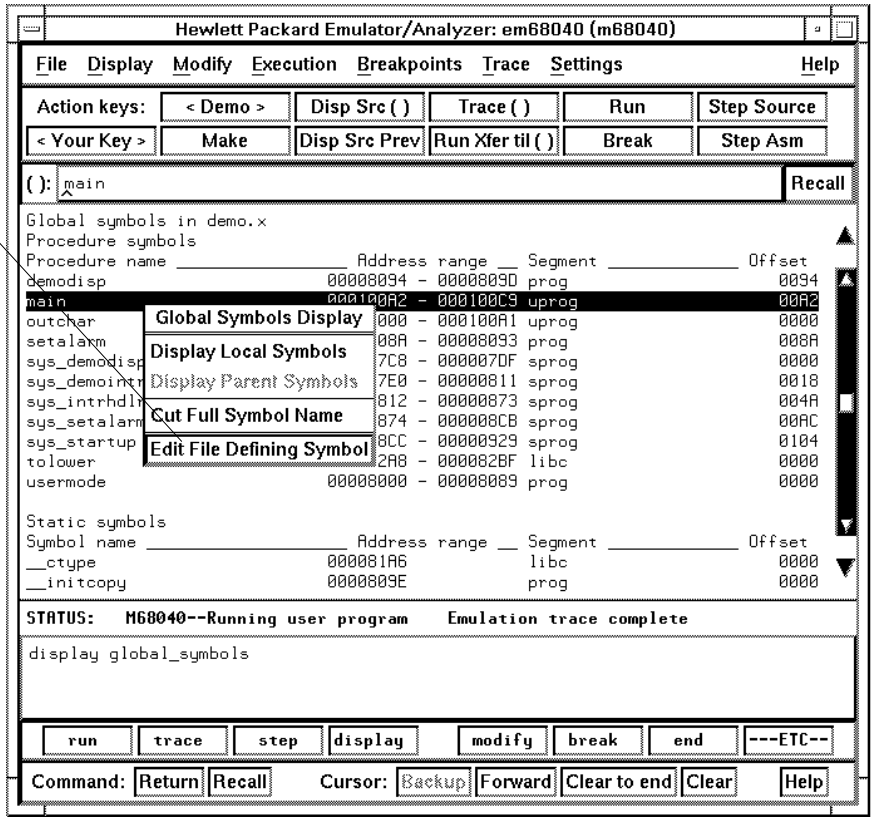
  The interface will choose the "vi" editor as its default editor, unless you specify another editor by setting an X resource. Refer to Chapter 13, "Setting X Resources", for more information about setting this resource.

  You must load symbols before most edit commands are available because symbol information is needed to be able to locate the files.

Examples

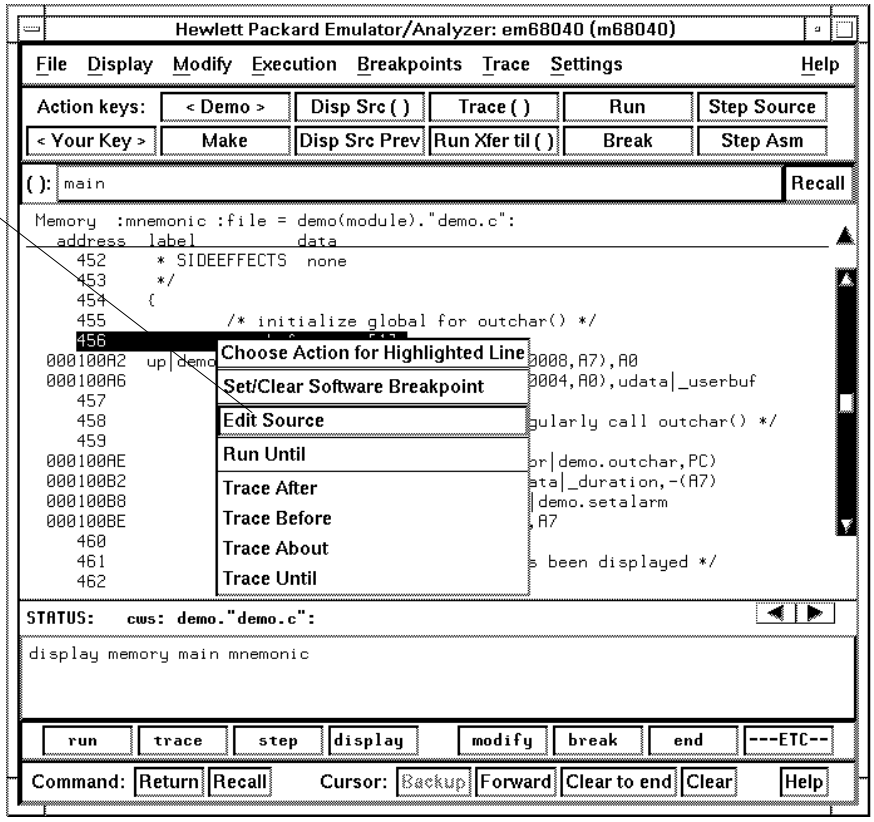
To edit a file that defines a symbol:

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 the location of 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.



Using Symbols

When you load a program for the first time, the emulator uses the Symbolic Retrieval Utilities (SRU) to build a symbol database for each module. This database associates symbol names and symbol type information (not data types) with logical addresses. You will see a message on screen showing the module for which the database is being built.

Once a symbol database is created for a particular module, it does not need to be rebuilt unless the module is changed. You can rebuild modules using the **srubuild** utility (refer to the *Symbol Retrieval Utilities, SRU, User's Guide*). If you reenter emulation without building symbols, the emulator software automatically rebuilds portions of the symbol database as you reference symbols in modified modules. Usually, you should use **srubuild** after you rebuild your absolute files to save time during emulation.

Global symbol information is immediately available for the file that you loaded. To obtain local symbol information, you need to reference the module that contains the symbols.

You can use the symbol names instead of addresses when entering expressions as part of an emulation command. Therefore, you don't have to remember address information to make a measurement. Also, the emulator can display symbols within the results of a measurement, using the **set symbols on** command. This helps you relate the measurement to your original program.

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 13, "Setting X Resources").

The MC68040 emulator interface can read absolute files in HP-OMF or IEEE-695 format. For more information on SRU, refer to the *Symbol Retrieval Utilities, SRU, User's Guide*. Also refer to the information on symbol entry syntax in the **—SYMB—** section of Chapter 11, "Emulator Commands".

When you load an absolute file into memory (unless you specify a load without symbols), symbol information is also loaded. Both global symbols and symbols that are local to a source file can be displayed.

To load a symbol database

- Choose **File→Load→Symbols Only...** In the dialog box, click on the name of the desired symbols file, and then click OK.

- Using the command line, load a new symbol database by entering the command:

```
load symbols <filename>
```

<filename> is the name of the absolute file in HP64000 or IEEE-695 format for which you want to load symbols.

The **load symbols** command is useful when your system uses several different absolute files or when the target program resides in target ROM and is not loaded through the emulator. The symbol database for the most recently loaded absolute file is the current symbol database. If you want to use a symbol database from a different absolute file without reloading the file, use the **load symbols** command to load only the symbol database for that file.

Example

Suppose you have a system that uses two absolute files, one called system.x and another called task.x. You load these as follows:

```
load system.x
load task.x
```

The symbol database for task.x will be available because it was loaded last. To reference symbols from system.x, use the command:

```
load symbols system.x
```

Now the symbol database for task.x will not be available.

To display global symbols

- Choose **Display→Global Symbols.**
- Using the command line, display global symbols by entering the command:

```
display global_symbols
```

The **display global_symbols** command displays a list of global (externally defined) symbols in the program modules you have loaded into emulation or target memory. The symbols list includes the address range associated with a symbol, the name of the associated segment, and the offset of the symbol within the segment.

You can use the **UP** and **DOWN** cursor keys and the **NEXT** and **PREV** keys to scroll or page through the global symbols listing.

Example

Display the global symbols for the demo program:

```
display global_symbols
```

To display local symbols

- If you are using the Graphical User Interface:
  - First place the name of the symbol whose local symbols should be displayed into the entry buffer, and then in the menu bar, choose **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 popup menu.
- Using the command line, display the symbols defined within a given symbol by entering the command:

  display local_symbols_in <symbol_name>

  This command displays address information associated with each symbol. The symbols defined within a given symbol are local to that symbol. That is, they are defined as children of that symbol. See "To enter a symbol" for more information on the <symbol_name>. If no local symbols are associated with your selection, the interface displays the parent symbol.

  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.

Example

Display the local symbols for the update_sys module in the demo program:

display local_symbols_in update_sys(module)

Suppose that you had an IEEE-695 absolute file with a module named system and a procedure within that module also named system. You could display the local symbols for the procedure named system by entering:

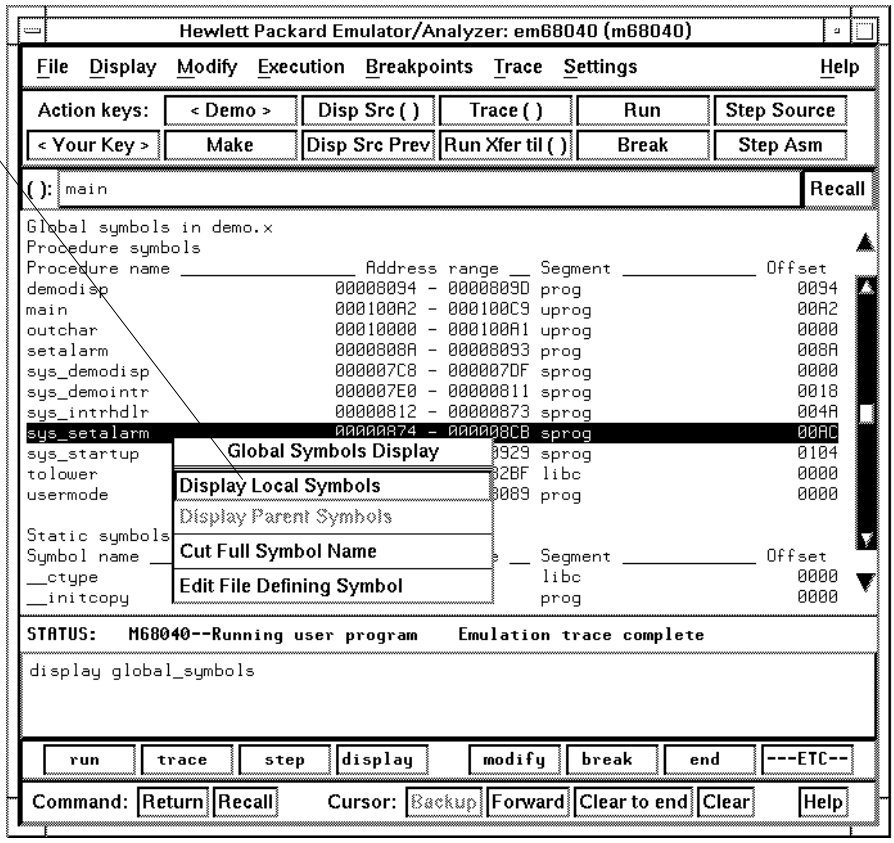
display local_symbols_in system.system

To display the source reference address ranges:

display local_symbols_in `system.c:`

To display local symbols using the symbols display popup menu:

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



If local symbols exist within the scope of the symbol you chose, then the display changes to show those symbols. Otherwise, the interface issues an error.

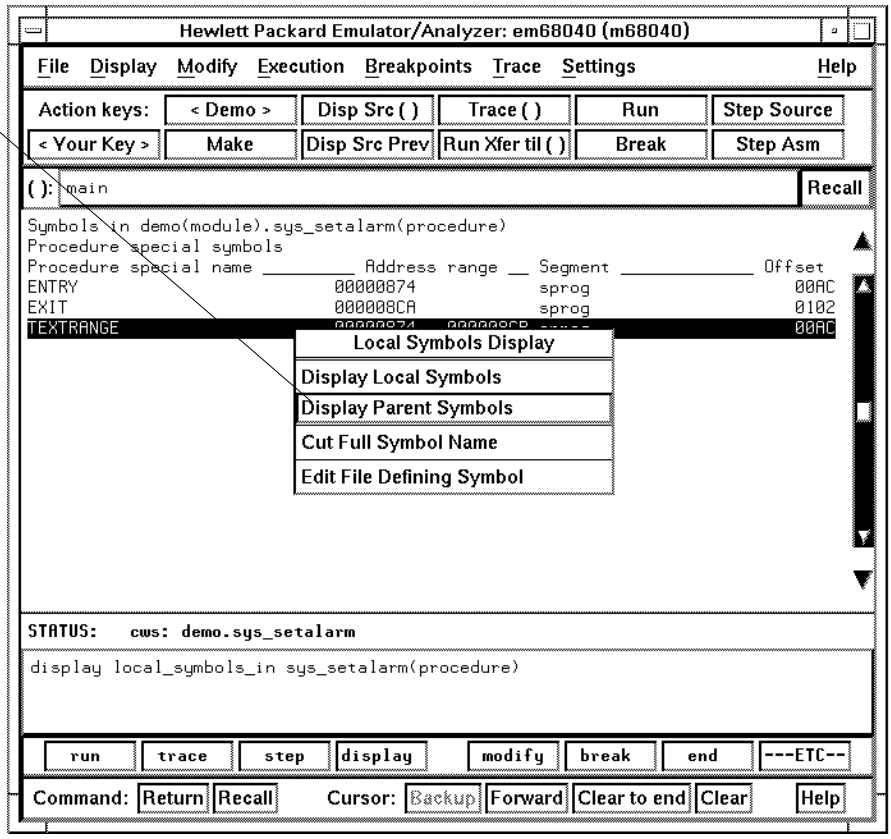
To display the parent symbol of a symbol

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

  If a parent symbol does not exist for the highlighted symbol, this menu item will be grayed-out and unresponsive to mouse clicks.

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

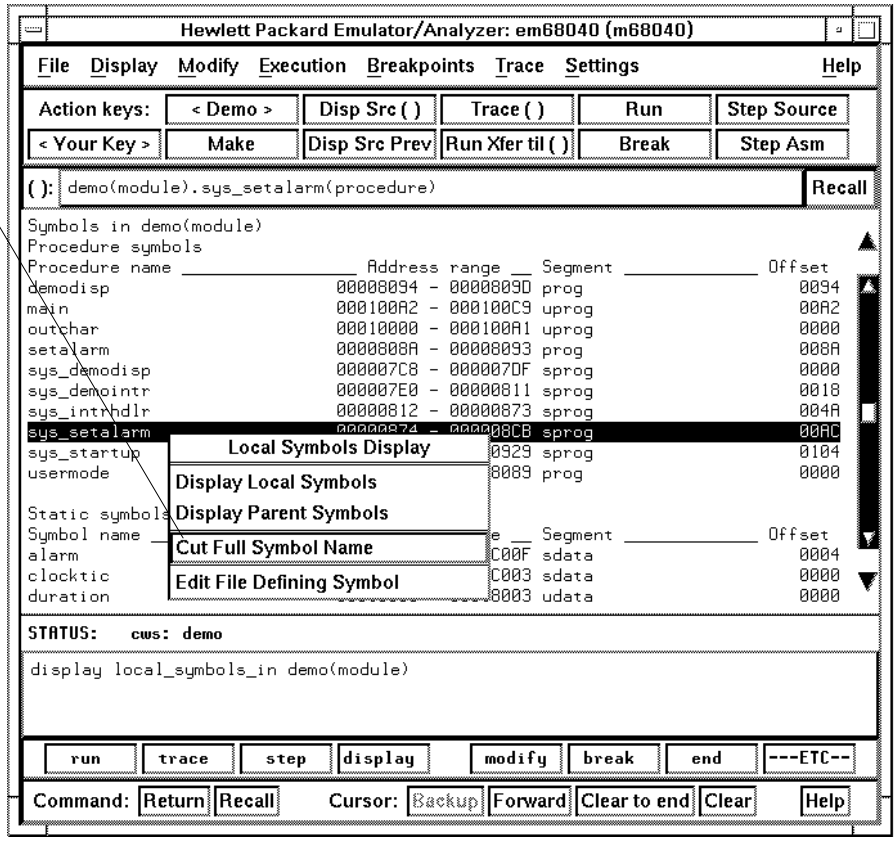
- When displaying symbols, position the mouse pointer over the symbol, press and hold the *select* mouse button, and choose **Cut Full Symbol Name** from the popup menu.

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 can be sure that you specified the complete scope of the symbol, including all names of symbols that were truncated.

Examples

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



To enter a symbol

- Enter symbols according to the syntax shown in the —SYMB— syntax pages in Chapter 11, "Emulator Commands".

Examples

These are examples of some valid symbol entries:

```
Int_Cmd
demo.Main(procedure)
demo.EndLoop
handle_msg.Fill_Dest
handle_msg.Cmd_A
system.c:line10
```

To display the current directory and current working symbol

- Choose **Display→Context...** A dialog box will open and show the name of the current directory and current working symbol.

- Using the command line, display the name of the current directory by typing **pwd**, and the name of the current working symbol by typing **pws**.

  If you're entering symbol names from several different modules, you may be unsure which symbol is the current working symbol. The **Display→Context...** or **pws** commands allow you to check this.

  The **pws** and **pwd** commands aren't available on the softkeys. You must type them at the keyboard.

  The directory context, included in the dialog box seen in the Graphical User Interface is the directory accessed by all system references for files (primary 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.

  The current working symbol context is supported by the emulator/analyzer and the Symbol Retrieval Utilities (SRU) working together. 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.

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 13, "Setting X Resources").

To change the current working symbol context

- Choose **File→Context→Symbols** and use the dialog box to select a 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 13, "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.

Example

The update_sys module of the demo program defines several symbols, including get_targets, graph_data, and write_hdwr. You refer to these in a group of memory display commands as follows:

```
display memory update_sys.get_targets blocked bytes
display memory update_sys.graph_data blocked bytes
display memory update_sys.write_hdwr blocked bytes
```

To save repeated typing of update_sys, enter:

```
cws update_sys
```

Then enter the memory display commands as:

```
display memory get_targets blocked bytes
display memory graph_data blocked bytes
display memory write_hdwr blocked bytes
```

Accessing Processor Memory Resources

While you are debugging your system, you may want to examine memory resources. For example, you may need to verify that the correct data is loaded, or check the results of a data write. Also, you may need to modify memory locations to test different data sets for a program. The emulator has flexible memory commands that allow you to view and modify memory as needed.

To display program data structures

- Place an absolute or symbolic address or file name containing the desired data structures in the entry buffer. Then choose **Display→Data Values→New ()** and select the data type from the cascade menu. This clears the data values display and adds a new item.

- Place the absolute or symbolic address of the desired data in the entry buffer. Then choose **Display→Data Values→Add ()** and select the data type from the cascade menu. This adds data items to the data values display.

- Choose **Display→Data Values** if you have a display of data values on screen and you want to update that display.

- Using the command line, display a program data structure by entering:

  display data <lower> **[*thru*** <upper>**]** <type> **{,**<lower> **[*thru*** <upper>**]** <type>**}**.

  <lower> and <upper> are address expressions representing the lower and upper boundaries of the memory range to be displayed.

<type> is a data type for display formatting as follows:

Type Designator	Description
byte	Hex display of one 8-bit location
word	Hex display of one 16-bit location
long	Hex display of one 32-bit location
int8	Display one 8-bit location as a signed integer (two's complement)
int16	Display one 16-bit location as a signed integer (two's complement)
int32	Display one 32-bit location as a signed integer (two's complement)
u_int8	Display one 8-bit location as an unsigned positive integer
u_int16	Display one 16-bit location as an unsigned positive integer
u_int32	Display one 32-bit location as an unsigned positive integer
char	ASCII characters

You can use the **display data** command to display simple data types in your program. This can make the display of simple variables easier to read because you don't have to visually sort a display (such as a memory display) to find the locations of interest.

You can use symbols in the address expression.

Example

To clear the data values display and add the target_temp static symbol from the demo program:

```
display data target_temp byte
```

To add display of the aver_temp array from the demo program:

```
display data , aver_temp thru aver_temp end word
```

To display only source lines

- Choose **Settings→Source/Symbol Modes→Source Only**.
- Using the command line, enter:

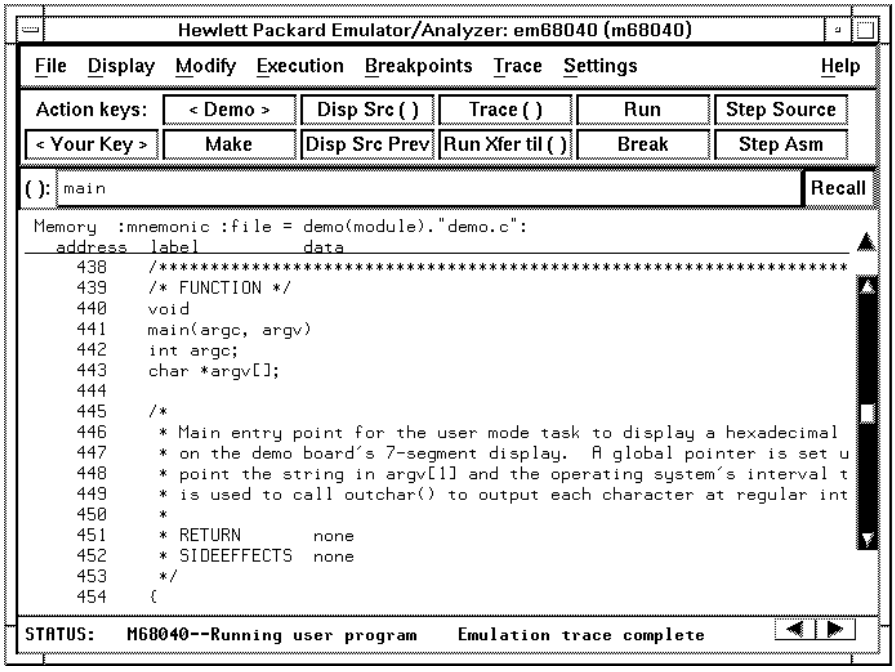
```
set source only symbols on
```

Only high-level source lines are displayed in mnemonic memory and trace displays.

Examples

To turn ON source lines in displays, and display memory in mnemonic format:

```
set source only symbols on
display memory main mnemonic
```



To display intermixed source lines

- Choose **Settings→Source/Symbol Modes→Source Mixed**.
- Using the command line, enter:

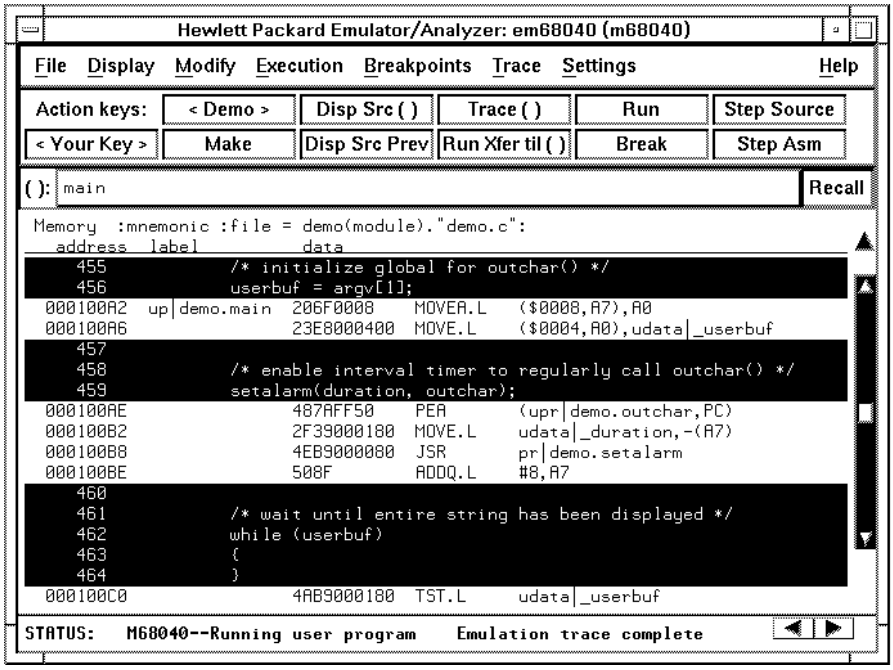
```
set source on symbols on
```

High-level source lines are intermixed with assembly language instructions in mnemonic memory and trace displays.

Examples

To turn ON source lines in displays, and display memory in mnemonic format:

```
set source on symbols on
display memory main mnemonic
```



To display symbols without source lines

- Choose **Settings→Source/Symbol Modes→Symbols**.
- Using the command line, enter:

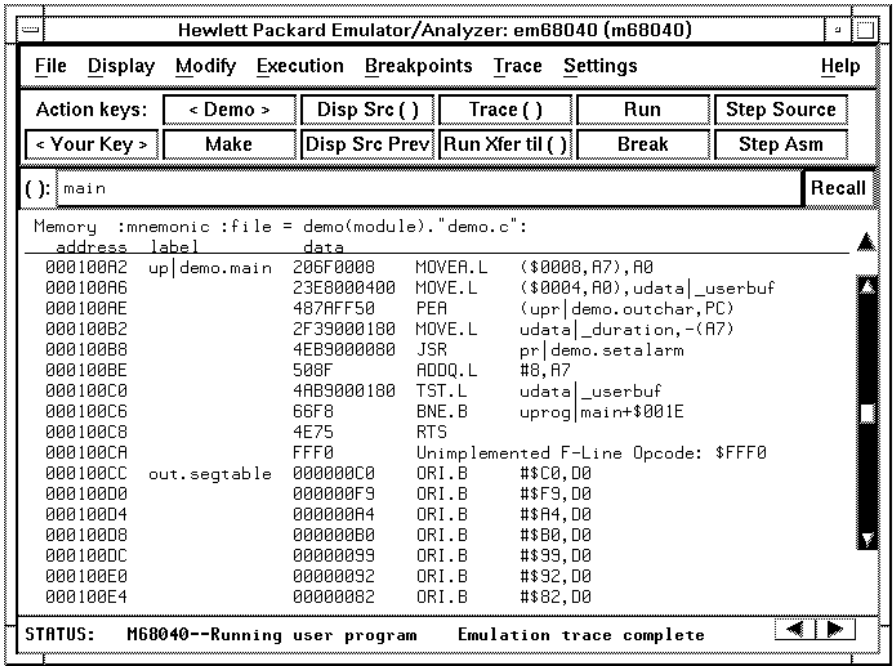
```
set source off symbols on
```

Symbols are included in memory mnemonic, trace, breakpoints, and register step displays.

Examples

To turn ON symbols in displays, and display memory in mnemonic format:

```
set source off symbols on
display memory main mnemonic
```



To display absolute addresses

- Choose **Settings→Source/Symbol Modes→Absolute**.
- Using the command line, enter:

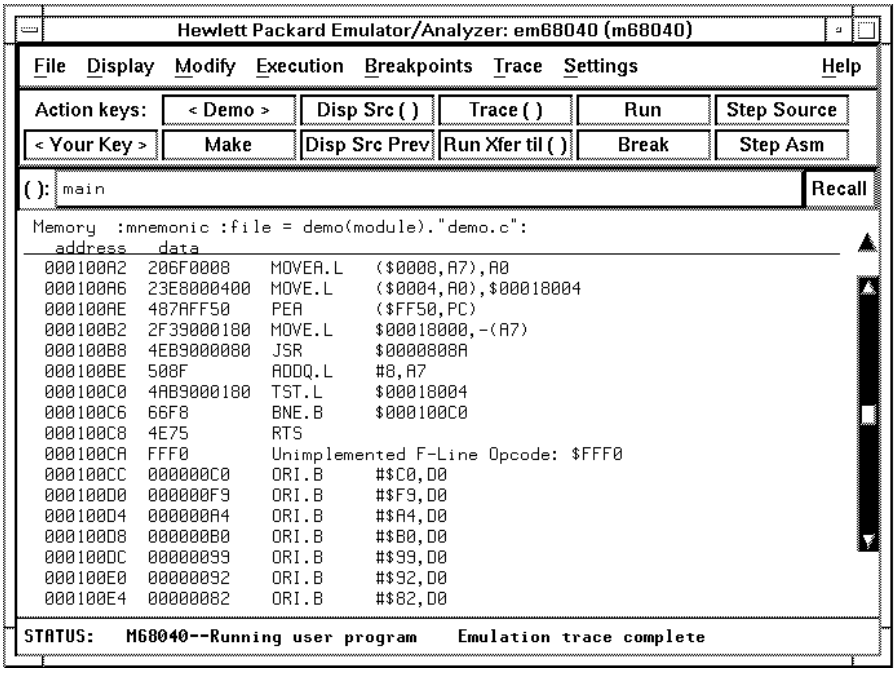
```
set source off symbols off
```

No symbols or source lines are included in mnemonic memory or trace displays.

Examples

To turn OFF symbols and source lines in displays, and display memory in mnemonic format:

```
set source off symbols off
display memory main mnemonic
```



To display memory in byte format

- Choose **Display→Memory→Hex()→bytes.** If you want to include a line range or starting point for your memory display in your command, enter it into the entry buffer before you execute this command.

- Using the command line, display a range of memory in byte format by entering:

  display memory <lower> **[*thru*** <upper>**]** ***bytes***

  To format the memory listing as a single column, add the keyword **absolute** before the data type in the **display memory** command. To format the memory listing as multiple columns, add the keyword **blocked** before the data type in the **display memory** command.

Example

Display the demo program's average temperature array:

```
display memory aver_temp thru aver_temp end bytes
```

To display memory in word format

- Choose **Display→Memory→Hex()→words.** If you want to include a line range or starting point for your memory display in your command, enter it into the entry buffer before you execute this command.

- Using the command line, display a range of memory in word format by entering:

  display memory <lower> **[*thru*** <upper>**]** ***words***

  To format the memory listing as a single column, add the keyword **absolute** before the data type in the **display memory** command. To format the memory listing as multiple columns, add the keyword **blocked** before the data type in the **display memory** command.

Example

Display the demo program's average temperature array:

```
display memory aver_temp thru aver_temp end blocked
words
```

To display memory in long word format

- Choose **Display→Memory→Hex()→long.** If you want to include a line range or starting point for your memory display in your command, enter it into the entry buffer before you execute this command.

- Using the command line, display a range of memory in long word format by entering:

  display memory <lower> **[*thru*** <upper>**]** ***long***

  To format the memory listing as a single column, add the keyword **absolute** before the data type in the **display memory** command. To format the memory listing as multiple columns, add the keyword **blocked** before the data type in the **display memory** command.

Example

Display the processor's interrupt vector table:

display memory `0` ***thru*** `3ffh` ***absolute long***

To display memory in mnemonic format

- Choose **Display→Memory→Mmemonic()** or **Mnemonic at PC.** If you want to include a line range or starting point in your command, enter it into the entry buffer before you choose the **Mnemonic()** command.

- Using the command line, display memory in mnemonic format by entering:

  display memory <lower> **[*thru*** <upper>**]** ***mnemonic***

  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.

  When you **display memory mnemonic**, the emulator disassembles the memory locations beginning with the first address you specify. If this address is not the starting address of an instruction, the display will be incorrect.

  To offset the addresses in the memory mnemonic display, add the parameter **offset_by <expression>** to the end of the display memory command line. **<expression>** is an address expression that is subtracted from each address in the memory display. If code gets relocated, and therefore makes symbolic information obsolete, you can use the **offset_by** option to change the address information so that it again agrees with the symbolic information. You can also use **offset_by** to change listed addresses so that they match addresses in compiler or assembler listings.

  Whether source lines, assembly language instructions, or symbols are included in the display depends on what you choose with the **Settings→Source/Symbols Modes** or **Settings→Display Modes** pulldown menu items.

  Use the **set symbols on** command to display symbol information for addresses in the memory mnemonic display.

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

Examples

To display memory for the main part of the demo program, enter main in the entry buffer and choose **Display→Memory→Mmemonic()**, or enter on the command line:

```
display memory main mnemonic
```

Display the write_hdwr routine for the update_sys program in mnemonic format, with symbols in the address column:

```
set symbols on
display memory update_sys.write_hdwr thru write_hdwr end
mnemonic
```

To return to the previous mnemonic display

- Choose **Display→Memory→Mmemonic previous.**

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 the line following the function call, and run the program from the current program counter.

To display memory in real number form

- Choose **Display→Memory→Real()→<real type>.** If you want to include a line range or starting point in your command, enter it into the entry buffer before you execute this command. **<real type>** may be short, long, extended, or packed.

- Using the command line, enter commands as follows:

  1 Display memory values as 32-bit (IEEE-754 single precision) real numbers by selecting:

  display memory <lower> **[*thru <upper>*] *real short***

  2 Display memory values as 64-bit (IEEE-754 double precision) real numbers by selecting:

  display memory <lower> **[*thru*** <upper>**] *real long***

  3 Display memory values as 96-bit (IEEE-754 double extended precision) real numbers by selecting

  display memory <lower> **[*thru*** <upper>**] *real extended***

  4 Display memory values as 96-bit Motorola Packed real numbers by selecting:

  display memory <lower> **[*thru*** <upper>**] *real packed***

  Real numbers use the formats defined by the *IEEE Standard for Binary Floating-Point Arithmetic.* They can be short (32 bits), long (64 bits), or extended (96 bits).

Example

To display a set of data values in real numbers, beginning with the floating humidity in the demo program, place the global symbol float_humid in the entry buffer and choose **Display→Memory→Real()→long.** If using the command line, enter:

display memory float_humid ***real long***

To redisplay memory locations

- Choose **Display→Memory.**
- Using the command line, redisplay memory with the same address range and format as the previous memory display by selecting:

```
display memory
```

The last range and format options are maintained in the interface. When you display memory without specifying the location or format for the display, the previous options are used.

To display memory repetitively

- Choose **Display→Memory→Repetitively.**
- Using the command line, continuously display memory with:

```
display memory repetitively
```

This command continuously updates the memory display. Use this only to monitor memory while running your target code; it requires a lot of CPU time. To allow the current memory display to be updated whenever the emulator detects a modification to memory content (such as loading a file, or setting a software breakpoint) use the **set update** command, or **Settings→Display Modes...**.

To modify memory

- Choose **Modify→Memory**, or enter the desired memory location and new value in the entry buffer and click on **Modify→Memory at ()**. The equivalent command will be shown on the command line. Complete the command by entering appropriate information on the command line.

- Using the command line, enter commands as follows:

  - To modify a single memory location to a single value, select:

```
modify memory <address> to <value>
```

  - To modify a range of memory locations to a single value, select:

```
modify memory <lower> thru <upper> to <value>
```

  - To modify a range of memory locations with a list of values, select:

```
modify memory <lower> thru <upper> to
<value1>,<value2>, ....
```

  - To change whether memory is modified by bytes, words, or long words, add the <mode> parameter before the **to** keyword.

  - To modify memory as real numbers, select:

```
modify memory <lower> [thru <upper>] real [short|long]
to <real1>[,<real2>, ...]
```

  - To modify a sequence of bytes to an ASCII string literal, select:

```
modify memory <lower> thru <upper> string to "<string>"
```

  The <address> parameter is an expression representing a single address location. The **<lower>** and **<upper>** values are address expressions representing the lower and upper boundaries of the memory area to be modified. **<value>** represents the data value to which the contents of memory are to be modified. The **<string>** is a character string.

The **<mode>** parameter can be either **bytes**, **words**, or **long**. Otherwise, the mode specified by the last **display memory** command determines how data is displayed. If you selected "Any" when you specified "Memory Access Size" as part of the emulation configuration, the size you specify here will be used to access memory for the modification you specify.

Examples

Modify the byte at e1f hex to 43 hex:

Choose **Modify→Memory**, and on the command line, type 0e1fh **to** 43h

```
modify memory 0e1fh to 43h
```

The above example assumes that byte mode was in effect. If not, add the mode parameter:

```
modify memory 0e1fh bytes to 43h
```

Modify the memory using a symbol:

```
modify memory erno bytes to 43h
```

Modify the range of locations from e00 through e38 to zero:

```
modify memory 0e00h thru 0e38h to 0
```

Modify the range of locations from 0e00 through 0e38 to “ABC”:

```
modify memory 0e00h thru 0e38h bytes to 41h,42h,43h
```

Modify the memory at e00 hex to the string “This is a string”:

```
modify memory 0e00h string to "This is a string\n\0"
```

Remember that the memory modification is affected by the display mode. Suppose that locations f00 and f01 each contain 01. If you enter the command:

```
modify memory 0f01h bytes to 3h
```

Then location f00 contains 01 and location f01 contains 03. But, if you entered:

```
modify memory 0f00h words to 3h
```

Then location f00 will contain 00, and location f01 will contain 03. Notice that you refer to a word by an even address, which is the address of its most significant byte (this is defined by the MC68040 processor architecture).

Using Processor Run Controls

When you don't use an emulator, run control can be difficult. Usually, you're limited to starting the processor from reset, and then entering data values that vector program execution to the routines you want to test. Reaching those routines may be difficult or impossible if the data values are boundary conditions or if the program logic is faulty.

By using the emulator, you can run the processor from the current program counter or any desired address. If you want to examine the system after each program instruction, you can use the **step** command to step through the program. You can break to the monitor program to examine on-chip resources such as RAM and registers, and you can reset the processor from the emulator.

To run a program

- Choose **Execution→Run** and select the desired starting point from the submenu, or select **until()** to specify the ending point. Enter the starting or ending address in the entry buffer before you choose a command that contains **from()** or **until()**.

- Using the command line, enter commands as follows:

  - To run a program from the current program counter (PC) value, enter:

  ```
  run
  ```

  - To run a program from a specific address, enter:

  ```
  run from <–EXPR–>
  ```

  Where <—EXPR—> is a valid address expression that may include symbols.

  - To run a program from the reset vector, enter:

  ```
  run from reset
  ```

  - To run a program from its transfer address, enter:

run from transfer_address

When you're ready to start a program run, either to test target system operation or make an analyzer measurement, you use the **run** command.

<—EXPR—> is a 32-bit address expression. You can include supervisor or user function codes to specify the privilege level for the run command.

The **run from reset** command pulses the processor reset line. The processor fetches the values at offsets 0 and 4 from the vector table, loads these values into the interrupt stack pointer and program counter registers, and then begins running from the program counter address value.

A **run** command causes the emulator processor to begin running from the current program counter, provided that the emulator is not in the reset state. If the emulator is in the reset state, the **run** command (with no parameters) is equivalent to a **run from reset** command, unless the **run** command is preceded by a **break** command.

If you reset the emulator, break to the monitor, and then run the emulator, the stack pointer and program counter values are taken from the values supplied to configuration items instead of from the reset vector locations. Refer to Chapter 8, "Configuring the Emulator", for more information about setting the initial stack pointer and program counter values.

Examples

To run from the demo program's starting location:

Enter main in the entry buffer, and choose **Execution→Run→from()**.

Or, on the command line, enter:

```
run from main
```

To run programs from the current program counter value:

Choose **Execution→Run→from PC**, or on the command line, enter:

```
run
```

To run programs from the transfer address

- Choose **Execution→Run→from Transfer Address**.
- Using the command line, enter:

```
run from transfer_address
```

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

Before you can run from the transfer address, it must exist in the absolute file, and you must load symbols along with the program code from the absolute file. If the interface does not detect a transfer address, this menu item is grayed-out and unresponsive to mouse clicks.

To run programs from reset

- Choose **Execution→Run→from Reset**.
- Using the command line, enter:

```
run from reset
```

This command resets the emulation processor and begins executing your target program at either the start address for the processor, or at the address fetched from the reset vector for the processor. It may be necessary to supply a reset signal from your target system as well. See your processor-specific documentation for information about the exact mechanism involved.

To run programs until a selected address occurs

- 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 popup menu.

- Place the address you want to run until in the entry buffer; then choose **Execution→Run→until()**.

- Using the command line, enter:

  run until <address>

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

  This command is useful for bypassing large areas of code. For example, you may want to run your program through the program startup code until the "main" function begins so that you can begin testing your code at "main".

  When using the command line, you can combine the various types of run-from commands with the run-until command; for example, you can run from the transfer address until the start of a routine you wish to test.

  You may need to enable breakpoints before "run until" will work. See "To enable or disable the breakpoint feature" later in this chapter.

To break to the monitor

- Choose **Execution→Break**.

- Using the command line, cause the emulation processor to break from execution of your target program and start execution in the monitor by entering:

  break

  The emulation monitor is a program that provides various emulation functions, including register access and target system memory manipulation. During a run that is restricted to real-time execution, you must break execution to the monitor before executing any emulation commands that access registers, emulation memory that is not dual-port, or target system memory. You also can use the **break** command to pause your target program execution.

  Execution breakpoints and **run until <address>** commands can be used to break to the monitor at selected points in your target program.

  The status line changes to "Running in monitor."

  If you enter a **break** command while the processor is in a wait state (hung bus cycle), the emulator may terminate hung target bus cycles in an attempt to transition into the monitor. A bus cycle is considered hung when the target system has not provided the required termination within 300 ms. The emulator never attempts to terminate hung bus cycles in program space. The emulator will generate a status message for each address where it forcefully terminates a bus cycle. You can determine emulator status (**Display→Status**) to get information about a hung bus cycle before initiating a break (and accept the termination side effect) or use the **reset** command.

To step the processor

- Choose **Execution→Step Source** or **Execution→Step Instruction**. Select the starting point for processor stepping from the associated submenu. If you will enter a command that requires a starting address, enter that address in the entry buffer before entering the command.

- Using the command line, enter commands as follows:

  - To step the processor one instruction from the current program counter value, enter:

  step

  - To step one line of high level source, enter:

  step source

  - To step the processor <count> number of times from the current program counter value, enter:

  step <count>

  - To step the processor one instruction from an address given by <address>, type:

  step from <address>

  - To step the processor <count> number of times from an address given by <address>, type:

  step <count> ***from*** <address>

  - To suppress display of registers for intermediate steps of a multi-step execution, add the **silently** parameter after the **step <count>** command. (**<count>** must be greater than one.) This is only effective when stepping is done in the same interface displaying registers.

The **step** command lets you single-step the processor through program code. **Step Source** executes one line in your high-level source program; **Step Instruction** executes one line of your assembly language program.

When displaying memory mnemonic, a highlighted bar shows the current program counter address. After each step, the highlighted bar moves to the new PC address. When displaying registers, the registers are updated after every step.

You can open multiple windows to show memory mnemonic and registers at the same time. Both are updated with each step.

If you omit the **<address>**, the current program counter value is used. You can use **transfer_address** to step from the entry point of the program.

When stepping through instructions associated with source lines, execution may take a long time 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.

The emulator uses the built-in tracing capability of the MC68040 processor to single step assembly instructions. The emulator needs the trace exception vector (located at offset 0x24 in the vector table) to be set properly in order to single step instructions. When a step command is given to the emulator, the emulator reads the trace exception vector and attempts to change one or more vector table entries if the trace exception vector is not set correctly. As long as the vector table is located in emulation memory or target RAM, stepping should always succeed. Upon completion of single stepping, the emulator restores modified vector table entries and issues a status message the first time the vector table is modified.

If the trace exception vector does not contain the correct value and the vector table is located in target ROM, the emulator will issue an error message and not perform the single step. There are two ways to deal with this situation. Either alter the ROM-based code so the trace vector contains the correct value, or copy/relocate the vector table into emulation memory or target RAM.

The correct value of the trace exception vector differs, depending on whether you are using a background or foreground monitor. The foreground monitor requires that the trace exception vector point to the TRACE_ENTRY address in the monitor (located at offset 0x680 from the start of the monitor). If the trace exception vector already contains the correct value, the emulator performs the single step without modifying the vector table. Otherwise, the emulator attempts to change the trace a-line and f-line exception vectors to the TRACE_ENTRY address in the foreground monitor.

The background monitor only requires that the trace exception vector be an even value and point to readable memory. This allows the processor to complete trace exception processing, including initial prefetches from the trace exception handler, during transition into the background monitor. After reading the trace exception

vector, the emulator attempts to read from the address it points to. If the read succeeds, the emulator single steps without modifying the vector table. Otherwise, the emulator attempts to write the current value of VBR into the trace exception vector (because the vector table is readable).

There are some limitations when single stepping. A step may fail when single stepping an instruction that changes the address of the vector table (modifies the VBR register). With the background monitor, instructions that can be interrupted (ie: floating-point operations) may not complete because the emulator generates an interrupt after a finite amount of time after the single step is initiated.

Examples

To step the processor one instruction from its present location, choose **Execution→Step Instruction→from PC**, or on the command line, enter:

```
step
```

To step the processor three instructions from the current program counter:

```
step 3
```

To step the processor five source-level instructions from the init_system symbol in the demo program:

```
step 5 source from init_system
```

To step once from the program entry point, choose **Execution→Step Instruction→from Transfer Address**, or on the command line, enter:

```
step from transfer_address
```

To reset the processor

- Choose **Execution→Reset**.
- Using the command line, enter commands as follows:
  - To reset the emulation processor from the emulator, enter:

    ```
    reset
    ```

  - To reset the emulator from the target system, assert the RESET signal in your target system.

When you apply power to the emulator, the initialization process leaves the emulator in the reset state. Changing some configuration items also resets the processor. (Refer to Chapter 8, "Configuring the Emulator", for more information.)

Sometimes you may want to reset the emulation processor prior to a program run. The **reset** command allows you to do this. Or, you can reset the emulation processor from the target system.

The MC68040 emulator will respond to a target system reset. A target system reset does not reset the entire emulator. It resets only the emulation processor.

If the emulator is running a user program when the target system reset occurs, it will behave as if a **run from reset** command were issued.

If the MC68040 emulator is in the monitor when the target reset occurs, it will reenter the monitor when the reset is released.

The reset command holds the processor in the reset state until a break, run, or step command. A CMB command can cause the emulator to run from reset. Also, a request to access memory or registers may cause a break into the monitor.

Viewing and Modifying Registers

The emulator allows you to display registers to determine the results of program execution. You can display a single register, or you can display groups of related registers.

Sometimes you may want to modify a register, and then run a segment of program code to test the results.

To display registers

- Choose **Display→Registers→BASIC**, or **FPU**, or **MMU** to display the desired register class.

- Using the command line, enter commands as follows:

  - To display an individual register, enter:

    display registers <register_name>

    where <register_name> is one of the names shown in the table on the next page.

  - To display the basic processor register set, enter:

    display registers or ***display registers BASIC***

  - To display the floating-point registers, enter:

    display registers FPU

The available registers and register classes are in the table on the following page.

Register Class	Register Names
BASIC	**PC, STATUS, USP, ISP, MSP, CACR, D0..D7, A0..A7, VBR, DFC, SFC**
FPU	**FPCR, FPSR, FPIAR, FP0..FP7**
MMU	**ITT0, DTT0, ITT1, DTT1, MMUSR, TC, URP, SRP**

The processor must be running to allow register displays. If it's running in the monitor, the emulator does the display directly. If the emulator is reset, it will try to break to the monitor. If it's running the target system program, the emulator forces a break to the monitor, gets the register data, and then returns to the user program. (If you restrict the emulator to real-time runs, the display registers command isn't allowed while you're running your target program. Refer to Chapter 8, "Configuring the Emulator.")

The MMU register class of the MC68EC040 is different from the MMU register class of the MC68040 and MC68LC040. The MC68EC040 uses registers DACR0/IACR0 and DACR1/IACR1, which are nearly identical to DTT0/ITT0 and DTT1/ITT1. These MC68EC040 registers are displayed in the DTT0/ITT0, DTT1/ITT1 registers, respectively.

The TTRs are still usable when the MMU is disabled and correspond with ACRs.

Examples

Display the processor's A0 register:

```
display registers A0
```

To modify registers

- Choose **Modify→Register...**, and in the dialog box, type in the register name and new value.

- Using the command line, modify a register to a new value by typing:

  modify register <regname> ***to*** <value>

  Where <regname> is the name of a processor register, and <value> is an expression matching the data type of the register (byte or word).

  You can enter values into the three FPU control registers using numbers in the following bases: hexadecimal, decimal, octal, and binary. (You can't use symbols for the floating-point registers.)

  You can enter values into the eight floating-point registers using either floating-point or hexadecimal notation. Special values, such as denormals, infinity, and **NaN** (Not a Number) can be entered by using hexadecimal notation. The following are examples of acceptable entries for the floating-point registers:

  +12.34e+56
  -1.E23
  .1e-23
  1.2
  .7
  7654321
  0000.000001
  7fff0000ffffffffffffffffH

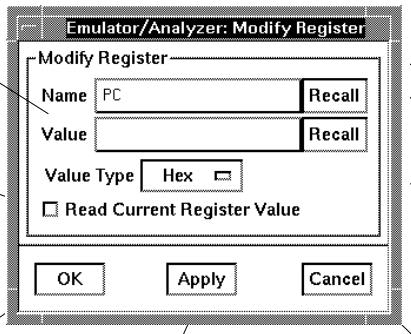
  Modifying a register's contents can help you test the effects of different program values without the trouble of rebuilding your program code. For example, you might stop the processor at a certain point (use a software breakpoint), and then modify a register and run from that point to test the result.

  The register is displayed after modification to confirm the change.

  The processor must be running to allow modifying registers. See "To display registers" above for more information.

Examples

To modify a register, choose **Modify→Register...**, and fill in the dialog box.



Place the mouse pointer in the text entry area and type in the name of the register and the new value.

Click this pushbutton to read the present value of the selected register.

Click Recall to select register names and values from predefined or previously specified entries.

Click this pushbutton and select the desired type from the submenu.

Click OK to modify the register to the new value and close the dialog box.

Click Apply to modify the register to the value specified and leave the dialog box open.

Click this pushbutton to cancel the modification and close the dialog box.

To use the command line to modify the PC register to an address:

```
modify register PC to init_system
```

To use the command line to modify the D3 register to 0:

```
modify register D3 to 0
```

Using Execution Breakpoints

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

Execution breakpoints are implemented using the BKPT instruction of the MC68040. You can enable, disable, set, or clear execution breakpoints.

Set execution breakpoints at the first word of program instructions. Otherwise, your BKPT may be interpreted as data and no breakpoint cycle will occur. When the BKPT instruction is executed, target program execution stops immediately (unlike using the analyzer to cause a break into the monitor, which may allow several additional bus cycles to execute before the break finally occurs).

Setting execution breakpoints in RAM

When you set an execution breakpoint in RAM, the emulator will place a breakpoint instruction (BKPT) at the address you specified, and then read that address to ensure that the BKPT instruction is there. The program instruction that was replaced by BKPT is saved by the emulator.

When the breakpoint instruction is executed, the BKPT acknowledge cycle is detected by the emulator, and the emulator causes a break to the monitor. At this point, the emulator replaces the BKPT instruction with the original instruction it saved. It also replaces the BKPT instruction with the original instruction whenever you disable or remove the breakpoint.

The emulator allows an unlimited number of breakpoints to be set in RAM.

Setting execution breakpoints in ROM

If you try to set an execution breakpoint at a location in ROM, the emulator will attempt to set the breakpoint as it does in RAM, but it will fail because the instruction in ROM will not change. Then the emulator will set up a hardware resource to "jam" the BKPT instruction onto the data bus when the processor attempts to fetch the normal instruction from the breakpoint address.

There are only enough resources in hardware to specify eight ROM breakpoints at one time.

To determine if an active breakpoint uses one of the eight hardware resources, display the address in memory. Breakpoints implemented in software will show a BKPT instruction at the breakpoint address. Breakpoints implemented using one of the eight hardware resources will show the original instruction at the breakpoint address.

Execution breakpoints in ROM when the MMU manages memory

If the MMU is enabled when setting an execution breakpoint in ROM, the emulator translates the logical breakpoint address and uses the physical address to set up the emulation hardware resource.

In the unlikely event that multiple logical addresses translate to the same physical address in ROM, or that ROM address translations change while the breakpoint is set, it is possible for the breakpoint to be jammed onto the data bus for the wrong logical address.

Using temporary and permanent breakpoints

When you set a temporary execution breakpoint, the emulator creates the breakpoint as described in the preceding paragraphs. When the breakpoint instruction is executed, the emulator breaks to the monitor and removes the breakpoint. Now you can execute that portion of program code as often as you like and the breakpoint will not occur again, unless you enable it again.

When you set a permanent breakpoint, the emulator will process it the same as a temporary breakpoint, but when the breakpoint instruction is executed, the original instruction will only replace the breakpoint instruction during its next execution. This allows you to step through the original instruction one time. After your first step, the BKPT instruction will replace the original instruction again so that the breakpoint will occur the next time the breakpoint address is hit.

Permanent breakpoints remain in effect until you explicitly disable or remove them.

Permanent breakpoints are available when using version A.04.00 or greater of the emulation system firmware.

To enable execution breakpoints

- Choose **Breakpoints→Enable.**
- Inside the breakpoints list display, press and hold the *select* mouse button and then choose **Enable/Disable Software Breakpoints** from the popup menu.
- Using the command line, enable breakpoints with:

  modify software_breakpoints enable

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

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

To disable an execution breakpoint

- Choose **Breakpoints→Enable** again. The **Breakpoints→Enable** selection is a switch.
- Inside the breakpoints list display, press and hold the *select* mouse button and then choose **Enable/Disable Software Breakpoints** from the popup menu.
- Using the command line, disable breakpoints with:

  modify software_breakpoints disable

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

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

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

To set a permanent breakpoint

When displaying memory in mnemonic format, position the mouse pointer over the program line where you wish to set the breakpoint and click the *select* mouse button. Or, press and hold the *select* mouse button and choose **Set/Clear Software Breakpoint** from the popup menu.

Place an absolute or symbolic address in the entry buffer; then, choose **Breakpoints→Permanent()**

Using the command line, enter the command:

modify software_breakpoints set <address> ***permanent***

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

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

To set a temporary breakpoint

- Type in the absolute or symbolic address of the breakpoint you want to set in the entry buffer. Then choose **Breakpoints→Temporary()**, (or choose **Breakpoints→Set()** if your version of HP 64700 system firmware is less than A.04.00).

- Choose **Breakpoints→Set All** to set all existing breakpoints in the breakpoint table.

- Inside the breakpoints list display, press and hold the *select* mouse button and then choose **Set All Breakpoints** from the popup menu.

- Using the command line, enter comands as follows:

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

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

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

  ```
  modify software_breakpoints set
  ```

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

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

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

Examples

Set a new breakpoint at get_targets:

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

Reenable all existing breakpoints:

```
modify software_breakpoints set
```

To set a ROM breakpoint in RAM

- Type in the name of the breakpoint you want to set in the entry buffer. Then choose **Breakpoints→Force HW→Permanent()** or **Temporary().**

- Using the command line, enter:

  ```
  modify software breakpoints set <ADDRESS> permanent (or temporary) force_hw
  ```

  There may be times when you want to have the emulator use one of its eight hardware resources to ensure an emulation break at a RAM address. For example, you may know that the program in ROM will overwrite the RAM address before the breakpoint is executed. Normally, this will eliminate the breakpoint instruction. The above commands ensure that the breakpoint will be executed at the specified address, regardless of how the software at that address may change during execution.

To clear an execution breakpoint

- Type in the name of the breakpoint you want to clear in the entry buffer. Then choose **Breakpoints→Clear().**

- Choose **Breakpoints→Clear All** to clear all existing breakpoints in the breakpoint table.

- Inside the breakpoints list display, press and hold the *select* mouse button and then choose **Clear (delete) Breakpoint** from the popup menu to clear the selected breakpoint.

- Using the command line, enter comands as follows:

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

  modify software_breakpoints clear <address>

  - To remove all existing breakpoints, enter:

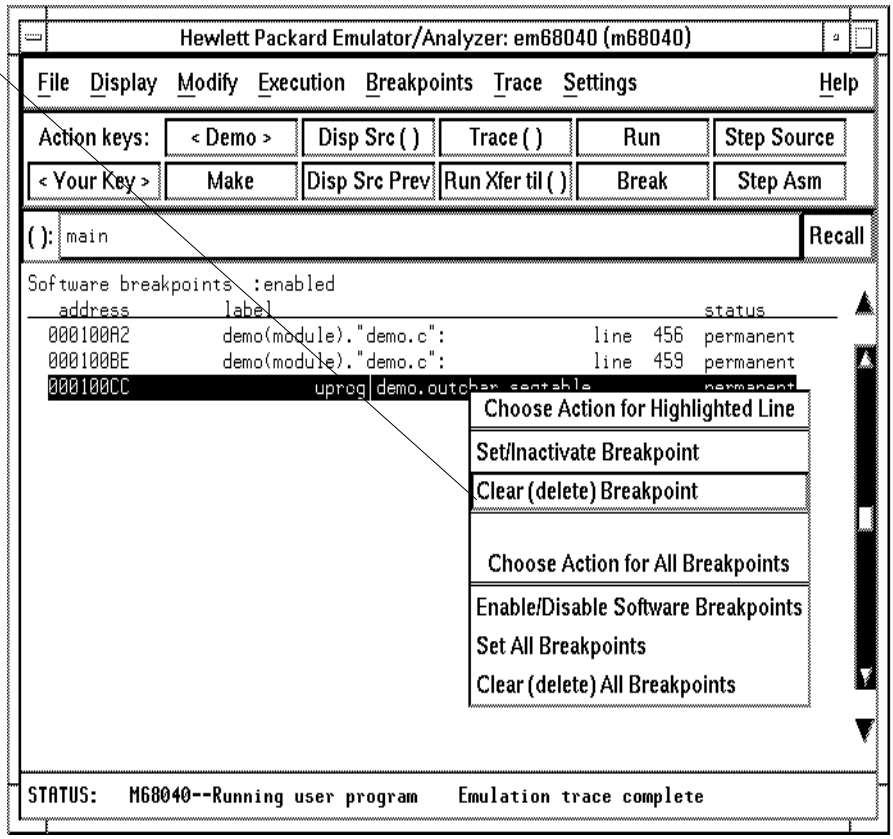
  modify software_breakpoints clear

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

Examples

To clear a breakpoint using the breakpoints display popup menu:

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



To clear an existing breakpoint at get_targets:

```
modify software_breakpoints clear update_sys.get_targets
```

To clear all existing breakpoints:

```
modify software_breakpoints clear
```

To clear all execution breakpoints

- When displaying breakpoints, position the mouse pointer within the breakpoints display screen, press and hold the *select* mouse button, and choose **Clear (delete) All Breakpoints** from the popup menu.
- Choose **Breakpoints→Clear All**.
- Using the command line, enter:

```
modify software_breakpoints clear
```

To display the status of all execution breakpoints

- Choose **Breakpoints→Display** or **Display→Breakpoints.**
- Using the command line, display the status of all breakpoints by selecting:

```
display software_breakpoints
```

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

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

The status of a breakpoint can be:

temporary	Which means the temporary breakpoint has been set but not encountered during program execution. These

	breakpoints are removed from the breakpoint table when the breakpoint is encountered.
pending	Which means the temporary breakpoint has been set but not encountered during program execution. These breakpoints are inactivated when the breakpoint is encountered.
permanent	Which means the permanent breakpoint is active.
inactivated	Which means the breakpoint has been inactivated. Temporary breakpoints are inactivated when they are encountered during program execution. Both temporary and permanent breakpoints may be inactivated using the breakpoints display popup menu.

In the breakpoints display, a popup menu is available, obtained by pressing the *select* mouse button. You can set, inactivate, or clear breakpoints as well as enable or disable the breakpoints feature from the popup menu.

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**.
- To display assembly language mnemonics with absolute addresses replaced by global and local syumbols where possible, choose **Settings→Source/Symbol Modes→Symbols**.
- To display assembly language mnemonics intermixed with high-level source lines, choose **Settings→Source/Symbol Modes→Source Mixed**.
- To display only high-level source lines, choose **Settings→Source/Symbol Modes→Source Only**.

  Using the command line, enter commands as follows:

  - To display mixed source and assembly language, enter:

  ```
  set source on
  ```

  - To display only source language statements, enter

  ```
  set source only
  ```

  - To display only assembly language, enter:

  ```
  set source off
  ```

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".

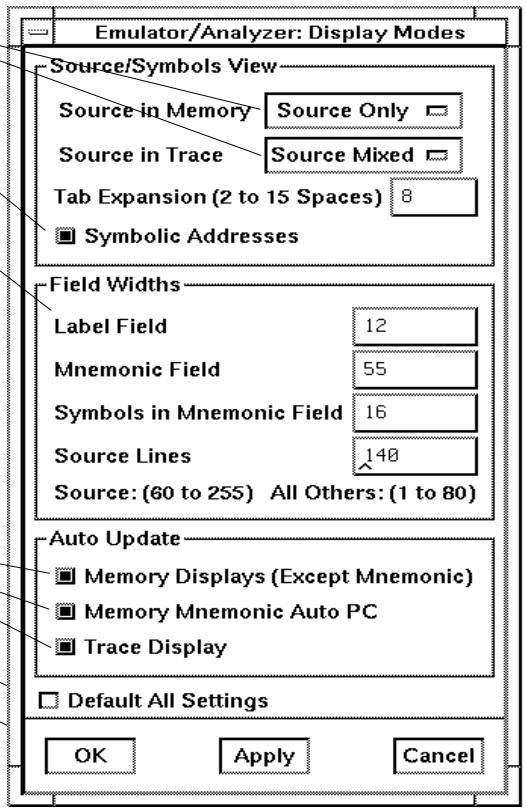
Clicking toggles whether symbolic information is displayed.

Move the mouse pointer to the text entry area and type in the desired field widths.
Label Field sets the width of the Label:/Address field.
Mnemonic Field sets the width of the Opcode or Status field.
Symbols in Mnemonic Field sets the widths of symbols shown in the Opcode or Status field.
Source Lines field sets the width of lines that show source-file lines.

Clicking toggles auto update settings.

Clicking this checkbox changes all display mode settings to their defaults.

Clicking:
OK saves changes and closes dialog box.
Apply saves changes and leaves dialog box open.
Cancel closes dialog box and ignores changes.



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.

Field Widths

Label Field sets the width (in characters) of the address field in the trace list or label (symbols) field in any of the other displays.

Mnemonic Field sets the width (in characters) of the mnemonic field in memory mnemonic, trace list, and register step mnemonic displays. It also changes the width of the status field in the trace list.

Symbols in Mnemonic Field sets the maximum width of symbols in the mnemonic field of the trace list, memory mnemonic, and register step mnemonic displays.

Source Lines sets the width (in characters) of the source lines in the memory mnemonic display.

Auto Update

Memory Displays (Except Mnemonic) toggles whether memory displays are automatically updated after commands that change memory contents or whether you must enter memory display commands to update the display. You may wish to turn off memory display updates, for example, when displaying memory mapped I/O.

Memory Mnemonic Auto PC toggles whether the mnemonic memory display is automatically updated to follow the PC or remain unchanged.

Trace Displays 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 the Emulator In-Circuit

As your target system design progresses, you will want to test features of your program that will interact with your target system hardware instead of emulation memory hardware.

You must connect the emulator probe to your target system to do in-circuit emulation. Then you can make analyzer measurements and have the memory display and other capabilities of the emulator to debug target system problems.

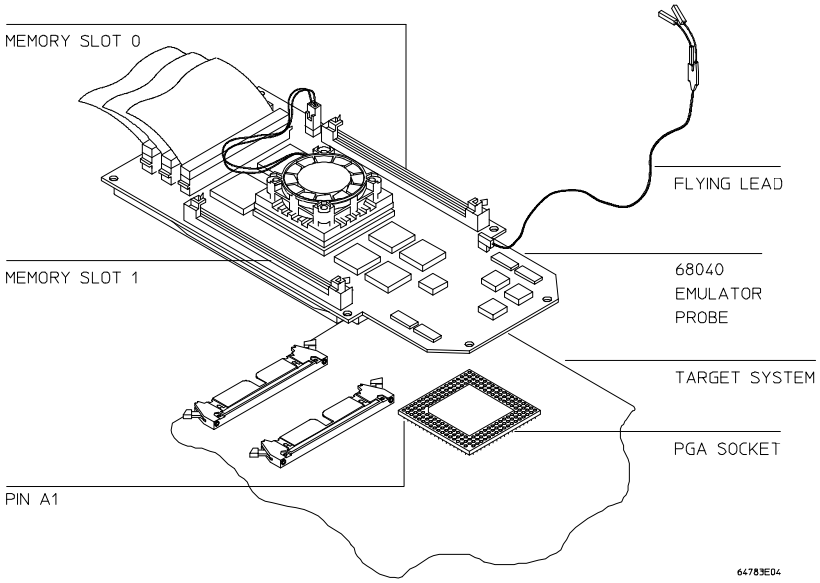
CAUTION When you use the emulator in-circuit, you need to carefully consider the relationship of the emulator to your target system design. Refer to Chapter 18, "Connecting the Emulator to a Target System", later in this manual. It discusses things you need to know to successfully connect the emulator to a target system and overcome problems you may encounter. Refer to Chapter 8, "Configuring the Emulator", for details of the emulation configuration.

To install the emulation probe

CAUTION Possible damage to the emulator probe. The emulation probe contains devices that are susceptible to damage by static discharge. Take precautions before handling the probe to avoid damaging the internal components of the probe with static electricity.

CAUTION Possible damage to the emulator. Make sure both your target system and emulator power are OFF before installing the emulator probe into the target system. The emulator may be damaged if the power is on when installing the probe.

CAUTION The emulator probe will be damaged if incorrectly installed. Make sure to align pin A1 of the probe connector with pin A1 of the socket.



1 Remove the processor from your target system socket. Note the location of pin A1 on the processor and on the target system socket. Store the processor in a protected environment (such as antistatic foam).

2 Insert the emulator probe into your target system socket. Make sure to align pin A1 of the emulator probe and the target system socket.

To power-on the emulator and your target system

CAUTION You must turn on power to the emulator before you turn on power to your target system. Otherwise, the emulator may be damaged. Turn off power to the target system before turning off power to the emulator.

1 Turn on power to the emulator.

2 Turn on power to your target system.

3 Before you turn off power to the emulator, be sure to turn off power to your target system.

To probe target system sockets

- A flexible adapter is available from Hewlett-Packard for special target system probing needs. It is listed in the following table:

Probe type	HP part number
68040 PGA to PGA flexible adapter	E3429A

Using The Emulator With MMU Enabled

When you enable memory management in the MC68040 emulator, many capabilities and features become available that are not otherwise offered. Also, some of the features of the emulator behave differently. The remaining pages in this chapter will help you when you are using the MC68040 emulator with the MMU enabled. Chapter 10, "Using Memory Management", provides detailed information to help you use the MC68040 MMU most efficiently.

Disable the MMU unless you are using it for address translation. You will still be able to use the transparent translation registers for such tasks as defining cache modes.

To enable the processor memory management unit

In order to use the MC68040 MMU to provide logical-to-physical address translation, the MMU must be enabled within the emulator configuration and the target system must deassert the MDIS signal (MMU Disable). If the MMU is not enabled within the emulator configuration, the emulator asserts the MDIS signal and ignores the MDIS signal from the target system, thus preventing the target system from using the MMU. If you are using the background monitor, you will need to select a foreground monitor before the MMU can be enabled within the emulator configuration. Refer to the chapter titled "Configuring the Emulator" for details of setting up the emulator configuration.

Once the $\overline{\text{MDIS}}$ signal is driven properly, the target system software is responsible for setting up address translation tables in memory and initializing the processor's MMU registers at run time. This task is typically managed by the target system's boot code or operating system. Refer to your *Motorola 68040 User's Manual* for information on how to use the MMU.

If the emulator is being used in an MC68EC040 target system, or if the MMU is not needed for translating page addresses from address translation tables in memory, then you should disable the MMU within the emulator configuration. This causes the emulator to assert the MDIS signal. However, the assertion of this

signal does not affect the operation of the transparent translation or access control registers.

To view the present logical-to-physical mappings

- Choose **Display→MMU Translations.** If you want to specify a logical address range for your mappings display, choose **Display→MMU Translations...** Then in the dialog box, click on **MMU Mappings**, and enter the desired logical address range.

- Using the command line, enter the command:

```
display mmu_translations
```

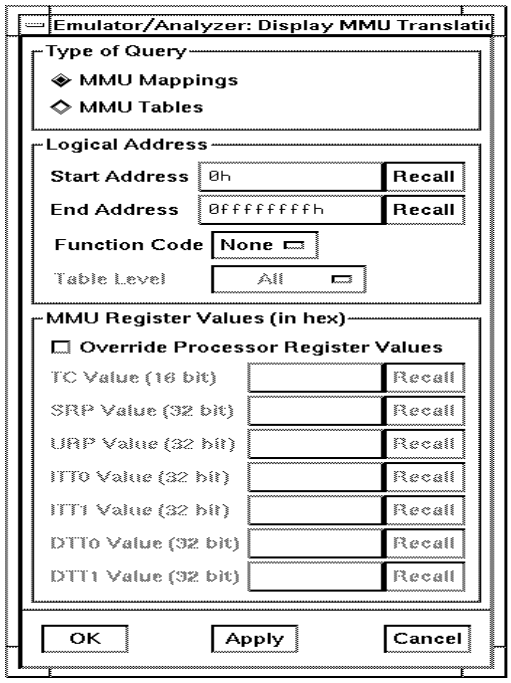
The display will show the logical-to-physical address translations defined by the current MMU registers and translation tables.

Examples

To see the logical-to-physical mappings using the default range of logical addresses (initially 0 through 0ffffffffh), choose **Display→MMU Translations**, or on the command line, enter:

```
display mmu_translations
```

To see all of the logical-to-physical mappings for logical addresses from 0 through 0ffffh (when only the URP root pointer is enabled), choose **Display→MMU Translations....** Then in the dialog box, click on **MMU Mappings**, and enter Start Address 0 and End Address 0ffffh, and click ok.



Using the command line, enter:

```
display mmu_translations 0 thru 0ffffh
```

To see the logical-to-physical mappings for the pages that contain logical address 40f0h, enter the command:

```
display mmu_translations 40f0h
```

To see only the mappings in supervisor space in the address range from 0 through 0ffffh, enter the command:

```
display mmu_translations fcode super 0 thru 0ffffh
```

To see only the mappings in user space in the address range from 0 through 0ffffh, enter the command:

```
display mmu_translations fcode user 0 thru 0ffffh
```

To show all of the valid mappings using the register overload capability of the command, enter a command to disable the MMU, and then enable it in your command, such as:

```
modify register MMU TC to 0
display mmu_translations use_value TC 8000h
```

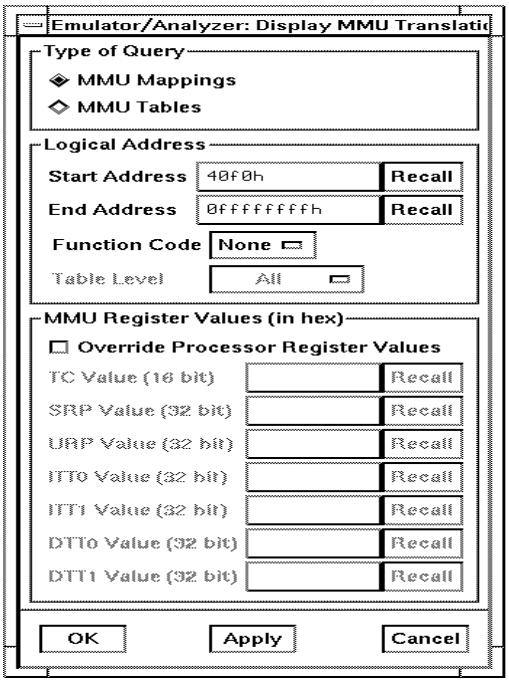
To see translation details for a single logical address

- Choose **Display→MMU Translations...** Then in the dialog box, click on **MMU Tables**, and enter the Logical Address whose table details you want to see in the Address box, and click ok.
- Using the command line, enter the command:

```
display mmu_translations tables <address>
```

Examples

To see how logical address 40f0h is mapped through the translation tables to its corresponding physical address, choose **Display→MMU Translations...** Then in the dialog box, click on **MMU Tables**, enter 40f0h in the Address box, and click ok.



Using the command line, enter:

display mmu_translations tables `40f0h`

To see how logical address 1000h in user space is mapped through the translation tables to its corresponding physical address, choose **Display→MMU Translations...** Then in the dialog box, click on **MMU Tables**, enter 1000h in the Address box, click on the pushbutton beside Function Code and select **user** from the submenu, and click ok.

Using the command line, enter:

display mmu_translations tables fcode user `1000h`

To see details of a translation table used to map a selected logical address

- Choose **Display→MMU Translations...** Then in the dialog box, click on **MMU Tables**, and enter the Logical Address whose translation table you want to see in the Address box. Finally, beside **Table Level**, click on the pushbutton to identify the table you want to see, and then click ok.

- Using the command line, enter the command:

  display mmu_translations tables <address> ***level*** <table_level>

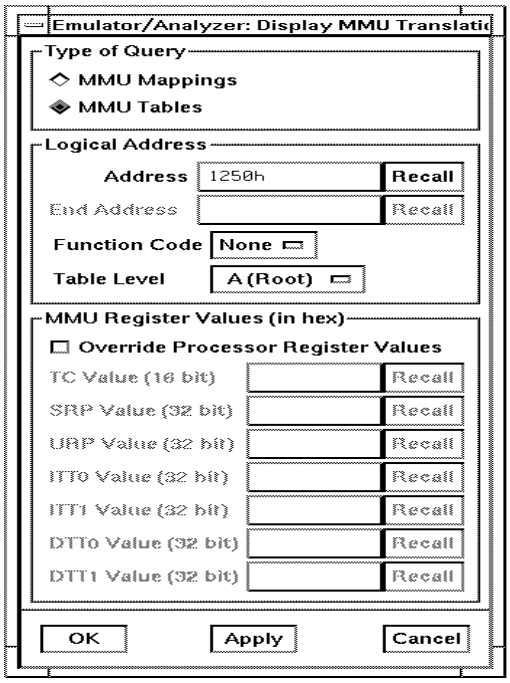
  Where <table_level> is the table level you want to see (either **A**, **B**, or **C**), and <address> is the logical address that uses the table at the point to be shown.

  Note that table level **all** is also offered. If you select **all**, you will see the translation details for your logical address through the tables. This is the same as if you had not selected the **level** <table_level> option.

  Table A may be accessed at several different base addresses, depending on which logical address is to be translated. This command ensures you see Table A where you want to see it.

Examples

To see the details of Table A used to map logical address 1250h, choose **Display→MMU Translations...** Then in the dialog box, click on **MMU Tables**, and enter 1250h in the Address box. Finally, beside **Table Level**, click on the pushbutton to select A, and then click ok.



Using the command line, enter:

display mmu_translations tables 1250h **level A**

Using an FPU with an MC68EC040 or MC68LC040 Target System

The MC68EC040 and MC68LC040 processors do not have an on-chip FPU. When floating-point functionality is required, all floating-point operations must be implemented in software using integer instructions. Language systems usually provide a floating-point software library for this purpose.

The HP 64783A/B emulator uses an MC68040 processor with an on-chip FPU. Because there is no way to disable the FPU, floating-point operations may execute differently, depending on the language system used. If your language system generates calls directly to the floating-point software library and does not emit any opcodes for floating-point instructions, then there should be no difference in floating-point operations whether you are using the emulator or the MC68EC040/LC040 processor plugged into your target system.

If your language system emits opcodes for floating-point instructions and relies on an F-Line exception handler to call the floating-point software library when the instruction is executed, then your target system will operate differently when the emulator is plugged in. When using the emulator, most floating-point instructions will be executed on the FPU in hardware instead of generating an F-Line exception and allowing the floating-point operations to be implemented in software. For this scenario, the following three points should be taken into consideration:

- Floating-point software libraries cannot be tested while the emulator is plugged in. Floating-point instructions are always executed on-chip, not by your floating-point libraries. This will definitely cause a problem for anyone trying to develop floating-point software libraries.
- Target programs containing FPU instructions will run faster when the emulator is plugged into the target system because they are executed in the hardware of the MC68040 instead of by the floating-point software libraries, as they will be when the MC68EC040/LC040 processor is plugged in. This will cause performance measurements to show much better results when using the emulator than you will actually obtain when you use the MC68EC040/LC040 processor.
- If you are unaware that your language tools use floating-point instructions (and you do not actively provide floating-point libraries and F-Line exception handling), you may find that your target system does not work when you unplug the emulator and plug in your MC68EC040/LC040 target processor.

Using M68040 support for the M68360 Companion Mode

Many designers need development tools for Motorola's M68360 processor. However, designers of higher performance systems will need to achieve a greater level of throughput than the 5 MIPS CPU32+ processor on board the M68360 can provide. These designers will consider using the M68360 Companion Mode together with a 22 MIPS M68040 "master" CPU.

This section shows you how to:

- set up a custom arrangement of action keys on the Graphical User Interface to operate the M68040/M68360 in the Companion Mode.
- use the custom action keys to develop products that use the M68040/M68360 Companion Mode. Through the action keys, you can perform such actions as viewing registers, configuring registers, developing boot code, and running programs.

To set up custom M68040 Action Keys to support the M68360 Companion Mode

The following paragraphs show you how to set up a custom ararngement of M68040 Action Keys in the Graphical User Interface to support the M68360 Companion Mode. Refer to Chapter 13 "Setting X Resources", and to the discussion in online file **$HP64000/lib/X11/app-defaults/HP64_Softkey** for details of how action keys are configured, formatted, and used within the Graphical User Interface.

1 Find the Motorola 68040 family-specific Application Resources in your **HP64_Softkey** file in your **$HP64000/lib/X11/app-defaults** directory. This portion of the file sets up the standard arrangement of action keys in the 68040 Graphical User Interface. It will be similar to the following example.

```
!-------------------------------------------------------------------------------! Action
! Action Key Definitions (See also XcHotkey discussion above)
*m68040*actionKeys.packing:    PACK_COLUMN
*m68040*actionKeys.numColumns: 2
*m68040*actionKeysSub.keyDefs: \
        " Demo "          "!telldemoHP 64783DEMO! in_browser" \
        "Disp Src ( )"      "display memory () mnemonic" \
        "Trace ( )"         "trace about (); display trace" \
        "Run"               "run" \
        "Step Source"       "step source" \
        " Your Key "      "!tellkeysHP! in_browser" \
        "Make"              "!make! in_browser" \
        "Disp Src Prev"     "display memory mnemonic previous_display" \
        "Run Xfer til ( )" "run from transfer_address until ()" \
        "Break"             "break" \
        "Step Asm"          "step"
!-------------------------------------------------------------------------------
```

One way to access the **HP64_Softkey** file is to choose the **File→Edit→File** pulldown in the Graphical User Interface, and in the file selection dialog box, select **$HP64000/lib/X11/app-defaults/HP64_Softkey** .

2 Copy the **$HP64000/lib/X11/app-defaults/HP64_Softkey** file to a temporary filename within your home directory. Name it "MyActKeys.tmp". Then edit the temporary file to delete all lines except those that define the present setup of action keys in your interface. See above.

3 Now change directories to **$HP64000/inst/emul/64783A/compmode**, and view the file **ACTION040360** on your screen.

```
$ cat ACTION040360
emul.m68040*browseSub.enableEnhancements: True
emul.m68040*browse_popup.title:Browser Window
emul.m68040*actionKeys.packing:       PACK_COLUMN
emul.m68040*actionKeys.numColumns:    3
emul.m68040*actionKeysSub.keyDefs: \
    "COMPANION"          "help68360register2" \
    "MODE KEYS"          "help68360register2" \
    "Gen Boot Code"      "boot68360sim" \
    "Help Reg ( )"       "help68360register1 ()" \
    "Help 360"           "help68360register" \
    "Pick Util"          "utils68360chip" \
    "Pick Reg 360"       "display68360reglist" \
    "Pick Chip 360"      "select68360chip" \
    "Reg 360 All"        "display68360registers" \
    "Mod 360 ( )"        "modify68360register ()" \
    "Reg 360 ( )"        "display68360register ()" \
    "Run Util ( )"       "()" \
    "Disp Mod 1/0"       "display68360aftmod" \
    "Set Chip ( )"       "set68360chip ()" \
    "PRBD 360 All"       "display68360prbds" \
    "Mod Memory"         "modify68360memory" \
    "PRBD 360 ( )"       "display68360prbd ()"
```

The file **ACTION040360** contains the special action keys that support the M68360 Companion Mode in the M68040 emulator. It is set up to arrange the action keys in three rows across the M68040 interface (note .numColumns: 3 in the file listing).

4 Add the Companion Mode action keys from file **$HP64000/inst/emul/64783A/compmode/ACTION040360** to the normal action keys contained in your file "MyActKeys.tmp". Save this file.

5 Edit your "MyActKeys.tmp" file to add a blank action key with a nondestructive action string before the first Companion Mode action key ("COMPANION") in order to obtain a balanced arrangement of action keys across the interface, and to change the number of action key rows to five (indicated by .numColumns: 5). When the edit is complete, your file should appear as follows:

Note

All three of the action strings that call the "help68360register2" command file are visual placeholders. You can customize this file by replacing these two action strings with any other action strings desired.

```
!-------------------------------------------------------------------------------

! Action Key Definitions (See also XcHotkey discussion above)
emul.m68040*browseSub.enableEnhancements: True
emul.m68040*browse_popup.title:Browser Window
emul.m68040*actionKeys.packing:       PACK_COLUMN
emul.m68040*actionKeys.numColumns:    5
emul.m68040*actionKeysSub.keyDefs: \
        " Demo "          "!telldemoHP 64783DEMO! in_browser" \
        "Disp Src ( )"      "display memory () mnemonic" \
        "Trace ( )"         "trace about (); display trace" \
        "Run"               "run" \
        "Step Source"       "step source" \
        " Your Key "      "!tellkeysHP! in_browser" \
        "Make"              "!make! in_browser" \
        "Disp Src Prev"     "display memory mnemonic previous_display" \
        "Run Xfer til ( )" "run from transfer_address until ()" \
        "Break"             "break" \
        "Step Asm"          "step" \
    "         "           "help68360register2" \
    "COMPANION"           "help68360register2" \
    "MODE KEYS"           "help68360register2" \
    "Gen Boot Code"       "boot68360sim" \
    "Help Reg ( )"        "help68360register1 ()" \
    "Help 360"            "help68360register" \
    "Pick Util"           "utils68360chip" \
    "Pick Reg 360"        "display68360reglist" \
    "Pick Chip 360"       "select68360chip" \
    "Reg 360 All"         "display68360registers" \
    "Mod 360 ( )"         "modify68360register ()" \
    "Reg 360 ( )"         "display68360register ()" \
    "Run Util ( )"        "()" \
    "Disp Mod 1/0"        "display68360aftmod" \
    "Set Chip ( )"        "set68360chip ()" \
    "PRBD 360 All"        "display68360prbds" \
    "Mod Memory"          "modify68360memory" \
    "PRBD 360 ( )"        "display68360prbd ()"
```

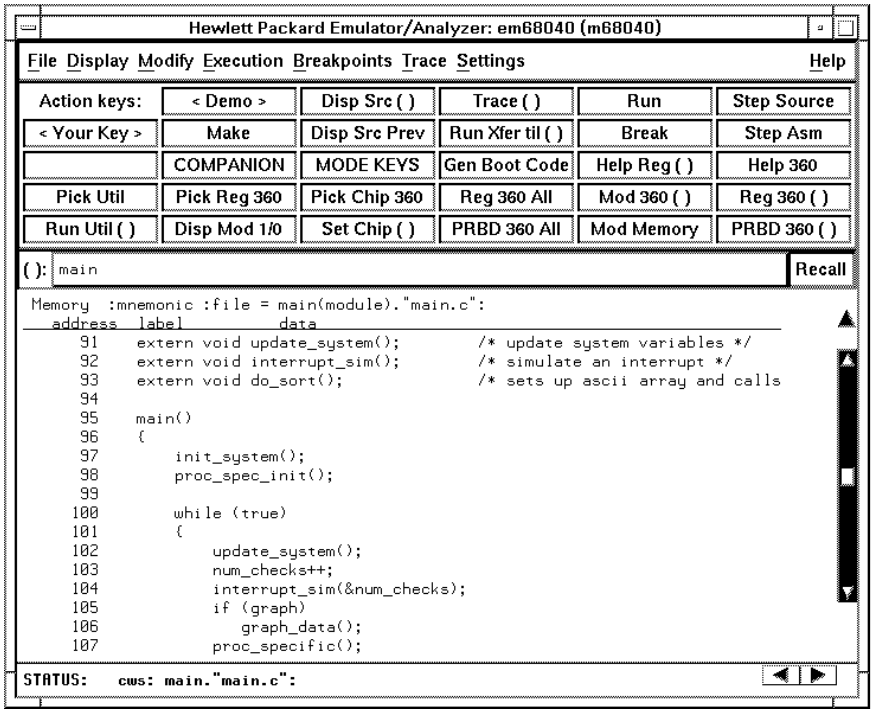
6 Add your special arrangement of action keys to your .Xdefaults file by typing the command:

cat MyActKeys.tmp >> $HOME/.Xdefaults

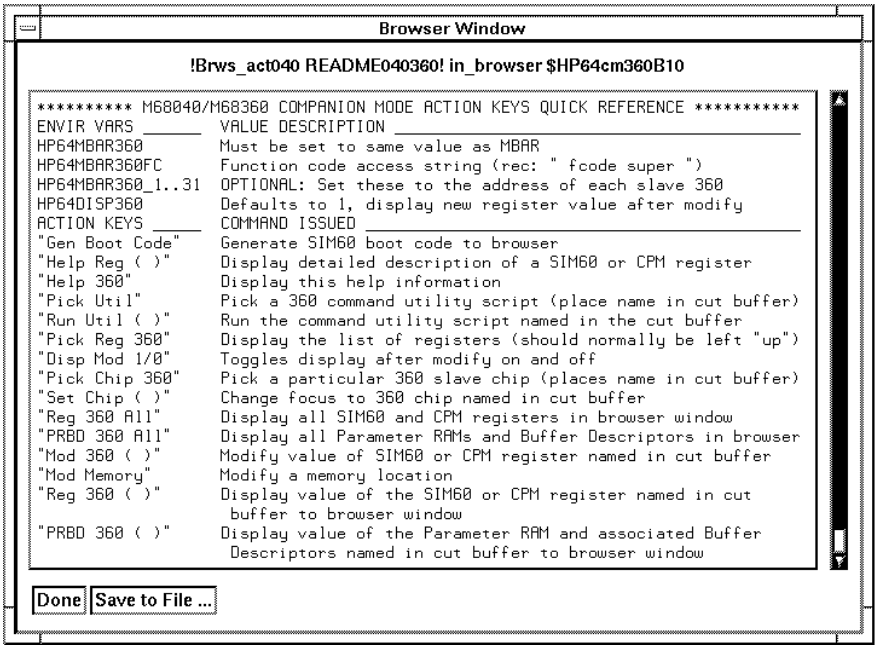
7 Make sure you export your ".Xdefaults" file so that it will be read when the M68040 interface starts. Use the following command:

export XENVIRONMENT=$HOME/.Xdefaults

8 Start your M68040 Graphical User Interface and see your special arrangement of action keys using your normal **emul700 ...** command. Your interface should have five rows of action keys, the last three rows being the M68040 Action Keys that support the M68360 Companion Mode. See below.



9 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 M68040 Action Keys. See below:



Tasks you may wish to perform when using the M68040/M68360 companion Mode

The following paragraphs show you how to perform typical development operations supported in the action keys of the M68040 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 appropriate 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 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 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 M68040 interface, press the "Pick Chip 360" Action Key. In the appropriate 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 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 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 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 browser window, highlight clean68360util and click Done. Press the "Run Util()" Action Key.
- To generate boot code for configuring the SIM60 unit, press the "Gen Boot Code" Action Key. When the 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 M68040/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 13, "Setting X Resources", and in the online file named **$HP64000/lib/X11/app-defaults/HP64_Softkey**, under the discussion called XcHotkey:Action Keys.

5

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.

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 Chapter 19, "Installation and Service", 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:

```
M68040                                          Sat Feb 20 12:16:02 1993

Trace List                  Offset=0
Label:  Address   Data                         Opcode or Status
Base:     hex     hex                             mnemonic
after  00003348 51FC137C  TRAPF
      =0000334A           MOVE.B    #$01,($001A,A1)
+001   0000334C 0001001A   $0001001A     sprog long read
+002   0007879F 00000001   $------01     sdata byte write
+003   00003350 528551FC  ADDQ.L    #1,D5
      =00003352           TRAPF
+004   00003354 BA866DEA  CMP.L     D6,D5
      =00003356           BLT.B     $00003342
+005   0007879E 00000100   $----01--     sdata byte write
+006   00003358 51FCB254  TRAPF
      =0000335A           CMP.W     (A4),D1
+007   0000335C 6E0E137C  BGT.B     $0000336C
      =0000335E           MOVE.B    #$01,($001C,A1)
+008   00003360 0001001C   $0001001C     sprog long read
+009   00003364 021300FD  ANDI.B    #$FD,(A3)
```

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, 1024 trace states were captured (1023 states plus the trigger state), and one analyzer sequence term was needed to satisfy the analyzer trigger.

```
M68040                                          Sat Feb 20 12:20:40 1993

Status

Emulator Status

  M68040--Running user program

Trace Status

  Emulation trace complete
  Arm ignored
  Trigger in memory
  Arm to trigger ?
  States 1024 (1024) 0..1023
  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 Chapter 19, "Installation and Service", 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 13, "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 popup 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 not on a long word boundary, 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 popup 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
```

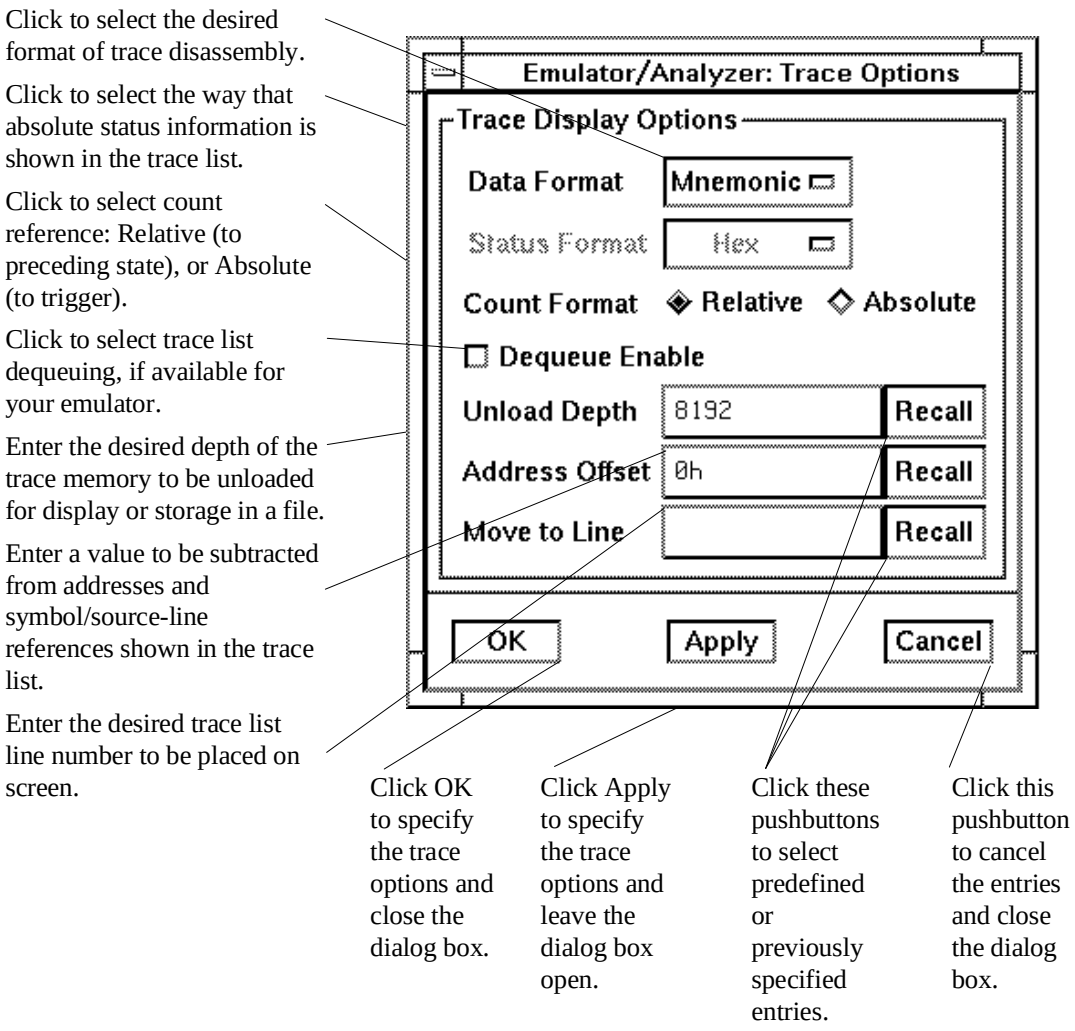
Displaying the Trace List

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 popup 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 Commands", for more information.

If you are using the emulator with the MMU enabled, you will need to enable and load the deMMUer before you can use source file symbols in your commands, display source file symbols in your trace lists, or see blocks of source code preceding related trace data. Refer to "Analyzing Program Execution when the MMU is Enabled" later in this chapter to see how to load and use the deMMUer.

Examples To use the Trace Options dialog box:



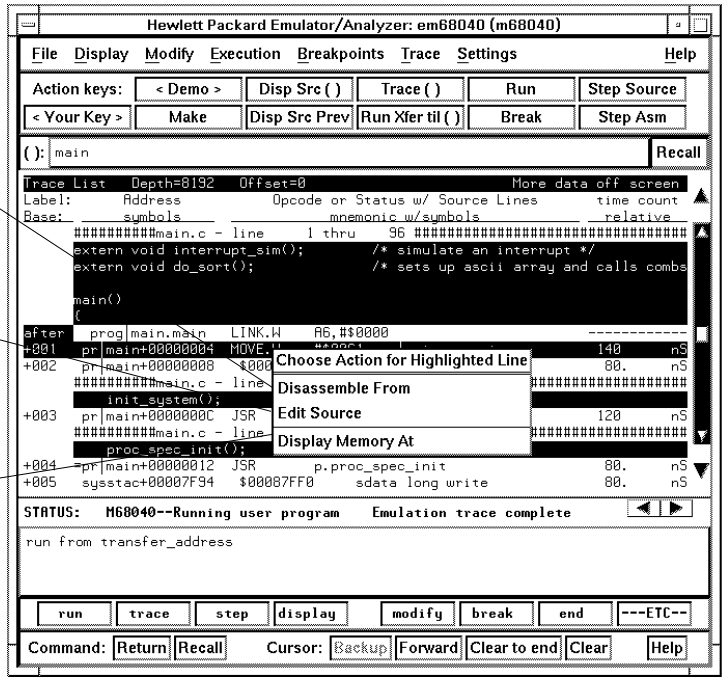
Examples

To use the trace list popup 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 popup 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 popup 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 MC68040 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
```

If you are using the emulator with the MMU enabled, you will need to enable and load the deMMUer before you can display source file symbols in your trace lists. Refer to "Analyzing Program Execution when the MMU is Enabled" later in this chapter to see how to load and use the deMMUer.

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 4, "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 4, "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 are using the emulator with the MMU enabled, you will need to enable and load the deMMUer before you can display source code preceding related trace data in your trace lists. Refer to "Analyzing Program Execution when the MMU is Enabled" later in this chapter to see how to load and use the deMMUer.

  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 4, "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 long_aligned update_sys counting time
```

(The **long_aligned** parameter is needed because the MC68040 fetches opcodes as 32-bit values and update_sys may not be the first part of that value.)

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 Chapter 19, "Installation and Service", 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 popup 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 popup 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.

Analyzing Program Execution When The MMU Is Enabled

Most emulation and analysis commands that require an address as part of the command use logical addresses. When the MC68040 MMU is enabled, physical addresses are placed on the emulation bus. The physical addresses may not be the same as the logical addresses. The deMMUer reverse translates the physical addresses back to logical addresses and supplies these to the analyzer so that the analyzer can:

- accept commands expressed in source file symbols.
- display trace lists with addresses expressed in source file symbols.
- display appropriate portions of source code preceding lists of trace data.

Refer to Chapter 10, "Using Memory Management", for detailed information to help you use the deMMUer more efficiently.

To program the deMMUer in a static memory system

1 Run your program to the point where you are sure the MMU is set up.

2 Break to the monitor program by choosing **Execution→Break.**

Using the command line, enter:

```
break
```

3 Choose **Settings→DeMMUer→Load from Memory.**

If you want the emulator to override one or more of the MMU register values with values you specify during the load process, choose **Settings→DeMMUer→Load from Memory...**, and specify the desired values in the dialog box.

To see a listing of the addresses that will be reverse translated by the DeMMUer during the loading process, choose **Settings→DeMMUer→Verbose** before you enter your **DeMMUer Load** command.

Note that **DeMMUer Load** commands automatically enable the deMMUer.

4 Using the command line, enter the following command:

```
load demmuer [verbose]
```

Note that the **load** command automatically enables the deMMUer.

5 Continue execution of your target program by choosing **Execution→Run→from PC** or **Execution→Run→from Reset**, or using the command line to enter **run**, or restart the program with the command: **run from reset**.

To pick the place to load the deMMUer, you might set an execution breakpoint in your code at a point where you are sure your MMU will be set up to translate the address space you want to analyze. After the breakpoint has executed (emulator running in foreground monitor), you can load the deMMUer.

Whether you continue your program or restart it, the deMMUer will have the ability to reverse translate the physical addresses according to the MMU setup at the time you issued the **load demmuer** command. The deMMUer will remain loaded even if you reset the emulation processor.

If you restart your program, you can use the analyzer to see how the MMU tables are created and how the program operates.

Address ranges will be reverse translated correctly if they are translated by the setup of the MMU that existed when you issued the **load demmuer** command. If context switches cause the MMU to access logical memory that was not represented in the MMU tables when you loaded the deMMUer, incorrect logical addresses will be provided by the deMMUer.

To store a deMMUer setup file

- Choose **File→Store→DeMMUer (From MMU Tables)** and enter the name to be used for the deMMUer file in the File Selection dialog box.

- Using the command line, enter:

  store demmuer <file>

  The deMMUer setup file is created by the emulator as it reads the present content of the MMU tables and creates a file of reverse translations appropriate for the deMMUer.

To load the deMMUer from a deMMUer setup file

- Choose **Settings→DeMMUer→Load from File**, and enter the name of the deMMUer file in the File Selection dialog box.

- Choose **File→Load→DeMMUer**, and enter the name of the deMMUer file in the File Selection dialog box.

- Using the command line, enter:

  load demmuer <file>

  Files that store setup information for the deMMUer have filenames that end in ".ED".

To trace program execution in physical address space

- Choose **Settings→Demmuer→Enable** to disable the deMMUer.
- Using the command line, disable the deMMUer with the command:

```
set demmuer off
```

The **Settings→Demmuer→Enable** command in the Graphical User Interface is a switch that enables and disables the deMMUer.

Now the analyzer will get its address information directly from the emulation address bus. This information is useful when you want to see behavior of your operating system.

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 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 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 and they are represented by the letters **X** or **x**. A zero must precede any numerical value that begins with an "X".

Symbols A symbol database is built when the absolute file is loaded into the emulator. Both global and local symbols can be used when entering expressions. Global symbols are entered as they appear in the global symbols display. When specifying a local symbol, you must include the name of the module ("anly.c") as shown below.

```
anly.c:cmp_function
```

Operators Analysis specification expressions may contain operators. All operations are carried out on 32-bit, two's complement integers. (Values which are not 32 bits will be sign extended when expression evaluation occurs.)

The available operators are listed below in the order of evaluation precedence. Parentheses are also allowed in expressions to change the order of evaluation.

-, ~	Unary two's complement, unary one's complement. The unary two's complement operator is not allowed on constants containing don't care bits.
***, /, %**	Integer multiply, divide, and modulo. These operators are not allowed on constants containing don't care bits.
+, -	Addition, subtraction. These operators are not allowed on constants containing don't care bits.
&	Bitwise AND.
|	Bitwise inclusive OR.

Values, symbols, and operators may be used together in analysis specification expressions. For example, if the local symbol exists, the following is a valid expression:

```
module.c:symb+0b67dh&0fff00h
```

However, you cannot add two symbols unless one of them is an EQU type symbol.

Emulation-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 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 equates. The descriptions identify the emulator status represented by the equates

68040 Equates

Name	Status Value	Description
ack	11xxxxxxxx1xxxxxy	Acknowledge access.
alt0	10xxxxxxxx1x000xy	Alternate logical function code 0.
alt3	10xxxxxxxx1x011xy	Alternate logical function code 3.
alt4	10xxxxxxxx1x100xy	Alternate logical function code 4.
alt7	10xxxxxxxx1x111xy	Alternate logical function code 7.
burst	0xxxxx0xxxxxxxxxxy	Burst cycle.
byte	0xxxxxx01xxxxxxxxy	Byte transfer request (SIZ1/SIZ0=01).
cpush	0xxxxxxxxx1x000xy	Data cache push access.
d_tblwk	0xxxxxxxxx1x011xy	Data translation table access.
data	0xxxxxxxxx1xx01xy	Data space access.
dma	0xxxxxxxxxx0xxxxxy	Direct memory access.
i_tblwk	0xxxxxxxxx1x100xy	Instruction translation tables access.
line	0xxxxxx11xxxxxxxxy	Line transfer request (SIZ1/SIZ0=11).
logical	0xx0xxxxxxxxxxxxxy	Logical memory address.
long	0xxxxxx00xxxxxxxxy	Longword transfer request (SIZ1/SIZ0=00).
physical	0xx1xxxxxxxxxxxxxy	Physical memory address.
prog	0xxxxxxxxx1xx10xy	Program space access.
read	0xxxxxxxxxxx1xxxxy	Read cycle.
retry	0xxxxxxxx00xxxxxxy	Retrying a previous bus cycle.
snp_hit1	0xx01xxxxx0xxxxxy	Snoop operation 1 (SC1/SC0=01)
snp_hit2	0xx10xxxxx0xxxxxy	Snoop operation 2 (SC1/SC0=10)
snp_inhb	0xx00xxxxx0xxxxxy	Snooping inhibited.
snp_miss	0xx11xxxxx0xxxxxy	Snoop miss.
sup	0xxxxxxxxx1x1xxxy	Supervisor space.
supdata	0xxxxxxxxx1x101xy	Supervisor data space.
supprog	0xxxxxxxxx1x110xy	Supervisor program space.
ta	0xxxxxxxx10xxxxxxy	Transfer acknowledge.

tea	0xxxxxxxx01xxxxxxy	Transfer error acknowledge.
upa0	0xx00xxxxxxxxxxy	User prog attributes UPA[1:0]=00.
upa1	0xx01xxxxxxxxxxy	User prog attributes UPA[1:0]=01.
upa2	0xx10xxxxxxxxxxy	User prog attributes UPA[1:0]=10.
upa3	0xx11xxxxxxxxxxy	User prog attributes UPA[1:0]=11.
user	0xxxxxxxxx1x0xxxy	User space.
userdata	0xxxxxxxxx1x001xy	User data space.
userprog	0xxxxxxxxx1x010xy	User program space.
word	0xxxxxx10xxxxxxxxy	Word transfer request (SIZ1/SIZ0=10).
write	0xxxxxxxxxxx0xxxxy	Write 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 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. See the *MC68040/EC040/LC040 Emulator/Analyzer Terminal Interface User's Guide.*)

  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 MC68040 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 8, "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 13, "Setting X Resources").

  See the "To use dialog boxes" description in Chapter 3, "Using the Emulator/Analyzer Interface", 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
```

Saving and Restoring DeMMUer Setup Files

To store a DeMMUer setup file

- Choose **File→Store→DeMMUer (From MMU Tables)...** In the dialog box, click on the name of the file you want to store your deMMUer setup (placing it in the Store Demmuer File box). Then click OK.

- Using the command line, store a deMMUer setup file by entering:

  store demmuer <filename>

  <filename> is any UNIX file name including paths. The extension **.ED** is assumed.

  Stores a deMMUer setup file (with a ".ED" suffix) by reading the present content of the MMU registers and the MMU tables.

To load a DeMMUer setup file

Choose **File→Load→DeMMUer...** In the dialog box, click the name of the file you want to load (placing it in the Load Demmuer File box). Then click OK.

- Using the command line, load a deMMUer setup file by entering:

  load demmuer <filename>

  <filename> is any UNIX file name including paths that was created by an appropriate store demmuer command. The extension **.ED** is assumed.

  The deMMUer setup file is loaded into the deMMUer. The present content of the MMU registers and the MMU tables are ignored.

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 statememts 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.

6

Making Coordinated Measurements

Using the Coordinated Measurement Bus to start and stop multiple emulators and analyzers

The Elements of Coordinated Measurements

The Coordinated Measurement Bus (CMB) connects multiple emulators and allows you to make synchronous measurements between those emulators.

For example, you might have a target system that contains an MC68040 processor and another processor. You use HP 64700 Series emulators to replace both target system processors, and connect the emulators using the CMB. You can run a program simultaneously on both emulators. Or, you can start a trace on one emulation-bus analyzer when the other emulator reaches a certain program address. These measurements are possible with the CMB.

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.

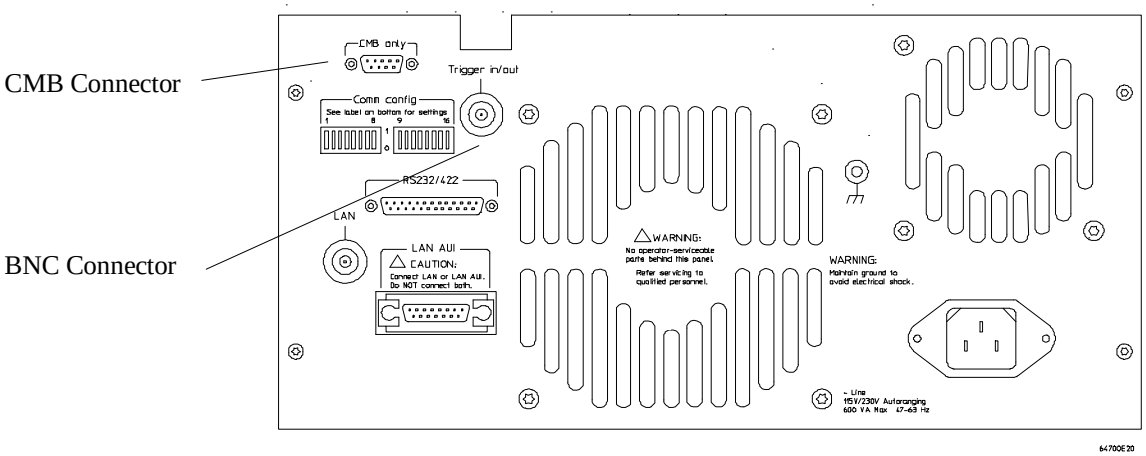
You can use the analyzer in one HP 64700 to arm (that is, activate) the analyzers in other HP 64700 Card Cages or to cause emulator execution in other HP 64700 Card Cages to break into the monitor.

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.

Tasks that you perform to make coordinated measurements include:

- 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.



There are three bidirectional 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 circuit 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.

Comparison Between CMB and BNC Triggers

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.

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 bidirectional. 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 ignored internally.

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.
- To connect the rear panel BNC.

For more information, refer to the *HP 64700 Series Installation/Service Guide.*

To connect the Coordinated Measurement Bus (CMB)

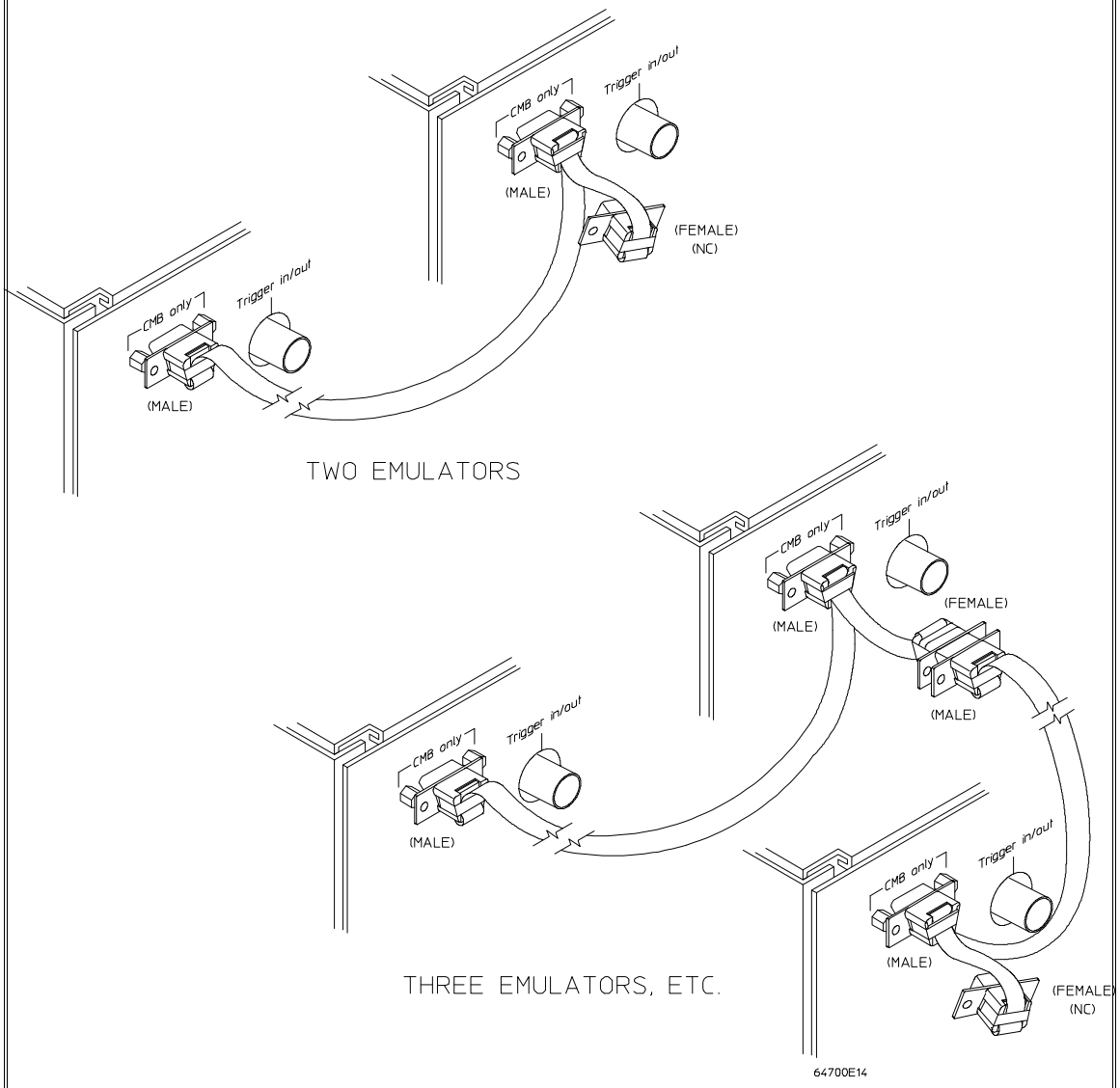
CAUTION Be careful not to confuse the 9-pin connector used for the CMB with those used by some computer systems for RS-232C communications. Applying RS-232C signals to the CMB connector is likely to result in damage to the HP 64700 Card Cage.

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.

Hewlett-Packard does not guarantee proper CMB operation if you are using a self-built cable!

1 Connect the cables to the HP 64700 CMB ports.



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

* A modification must be performed by your HP Sales Engineer.

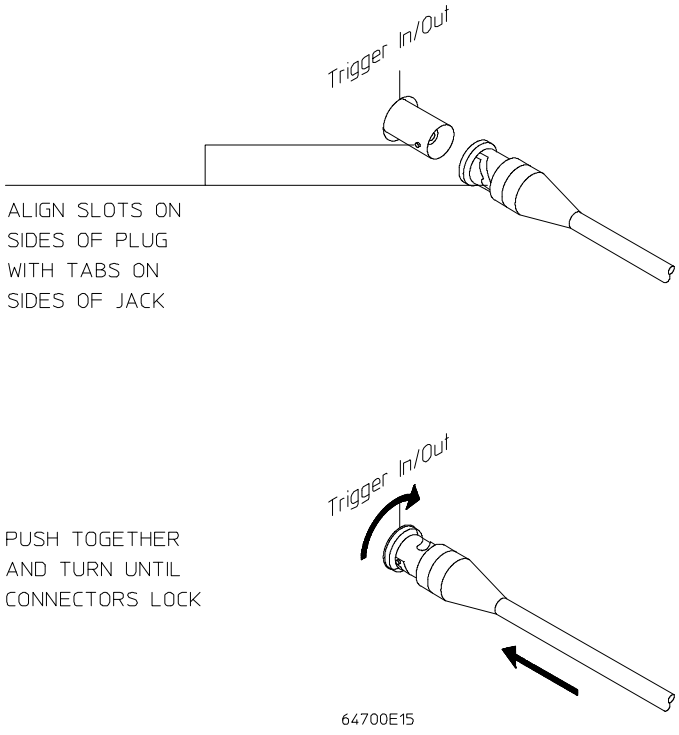
Emulators using the CMB must use background emulation monitors.

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 must 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.



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 on the command line.

  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 while stepping the processor. (See "To disable synchronous measurements" following.)

  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 from the transfer address:

```
specify run from transfer_address
```

To trace from an address when synchronous execution begins:

```
specify trace after address 10h
```

To start synchronous measurements

- Enter the **cmb_execute** command on the command line.

The **cmb_execute** command will cause the emulator to emit a pulse on the EXECUTE line, thereby initiating a synchronous measurement. You do not have to enable CMB interaction to use the **cmb_execute** command because by enabling CMB interaction, you are only specifying 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 the emulators participating in the synchronous measurement breaks to its monitor.

To disable synchronous measurements

- Enter the **specify run disable** command on the command line.

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**, that can carry trigger signals 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.*

- To configure the internal **trig1** and **trig2** lines, you must access the emulation configuration, either by choosing **Modify→Emulator Config ...** in the graphical user interface and then selecting Interactive Measurement Specification, or by entering the **modify configuration** command in the softkey interface, and then answering "yes" to the "Modify interactive measurement specification?" question. In the softkey interface, the following display appears.

```
                    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.)



STATUS:   Interactive Measurement Specification_____________________________.........
Should BNC drive or receive Trig1? neither


_drive__ receive_ neither_ __both__       _________ _________ _________ _RECALL_
```

This display illustrates the possible connections between the internal lines (**trig1** and **trig2**) and the emulator, analyzer, and external devices.

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 disable connections made by the internal trig1 and trig2 lines by answering "neither" or "no" to the appropriate interactive measurement configuration question.

These are some ways that you can use the internal trigger signals:

- 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.
- It is 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.

The following tasks show you how to set up the emulator and analyzer to:

- Drive the emulation trigger to the CMB and BNC.
- Break emulator execution on CMB and BNC signals.
- Arm the emulation-bus analyzer on CMB, BNC, and analyzer signals.

To drive the emulation-bus analyzer trigger signal to the CMB

- Choose **Modify→Emulator Config ...**

  1 In the top-level emulator configuration dialog box, click on Interactive Measurement Specification under Analyzer Configuration Sections.

  2 Choose “receive” beside the “CMBT on Trig1?” question.

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

  1 Answer “yes” to the “Modify interactive measurement specification?” question.

  2 Answer “receive” to the “Should CMBT drive or receive Trig1?” question.

  You could also drive the emulation-bus analyzer trigger to the CMB over the trig2 internal line by specifying that the CMBT should receive trig2 and that the emulation-bus analyzer should drive trig2.

To drive the emulation-bus analyzer trigger signal to the BNC connector

- Choose **Modify→Emulator Config ...**

  1 In the top-level emulator configuration dialog box, click on Interactive Measurement Specification under Analyzer Configuration Sections.

  2 Choose “receive” beside the “BNC on Trig1?” question.

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

  1 Answer “yes” to the “Modify interactive measurement specification?” question.

  2 Answer “receive” to the “Should BNC drive or receive Trig1?” question.

  You could also drive the emulation-bus analyzer trigger to the BNC over the trig2 internal line by specifying that the BNC should receive trig2 and that the emulation-bus analyzer should drive trig2.

To break emulator execution on signal from CMB

- Choose **Modify→Emulator Config ...**

  1 In the top-level emulator configuration dialog box, click on Interactive Measurement Specification under Analyzer Configuration Sections.

  2 Select "drive" for the "CMBT on Trig1" item.

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

  1 Answer “yes” to the “Modify interactive measurement specification?” question.

  2 Answer “drive” to the “Should CMBT drive or receive Trig1?” question.

  The trig1 signal is always supplied to the emulator. By entering the command, **trace break_on_trigger** emulation will break to the monitor when the CMB signal occurs.

  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 **Modify→Emulator Config ...**

  1 In the top-level emulator configuration dialog box, click on Interactive Measurement Specification under Analyzer Configuration Sections.

  2 Select "drive" for the "BNC on Trig1" item.

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

  1 Answer “yes” to the “Modify interactive measurement specification?” question.

  2 Answer “drive” to the “Should BNC drive or receive Trig1?” question.

  The trig1 signal is always supplied to the emulator. By entering the command, **trace break_on_trigger** emulation will break to the monitor when the BNC signal occurs.

  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-bus analyzer on signal from CMB

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

  1 Answer “yes” to the “Modify interactive measurement specification?” question.

  2 Answer “drive” to the “Should CMBT drive or receive Trig2?” question.

  3 Answer “receive” to the “Should Analyzer drive or receive Trig2?” question.

  4 Use the **arm_trig2** option to the **trace** command.

To arm the emulation-bus analyzer on signal from BNC

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

  1 Answer “yes” to the “Modify interactive measurement specification?” question.

  2 Answer “drive” to the “Should BNC drive or receive Trig2?” question.

  3 Answer “receive” to the “Should Analyzer drive or receive Trig2?” question.

  4 Use the **arm_trig2** option to the **trace** command.

Making Example Measurements

The following tasks show you how to:

- Start a simultaneous program run on two emulators.
- Trigger one emulation-bus analyzer with another.
- Break to the monitor on an analyzer trigger signal.

To start a simultaneous program run on two emulators

Before performing these steps, both emulators must be connected to the CMB. To connect the CMB, see "To connect the coordinated measurement bus (CMB)" at the beginning of this chapter.

1 Enable the CMB on each emulator.

2 Reset each emulator.

3 Set the run address for the first emulator.

4 Set the run address for the second emulator.

5 Start program execution on both emulators.

The procedure for starting a simultaneous trace on two emulators is similar. For each emulator, you should set up the trigger specification before enabling the CMB. Then start the analysis trace to enable trace on execute for each emulator. When the EXECUTE signal is received, both emulators will begin running and will start a trace according to the given trigger specification.

To trigger one emulation-bus analyzer with another

Before performing these steps, both emulators must be connected to the CMB. To connect the CMB, see "To connect the coordinated measurement bus (CMB)" at the beginning of this chapter.

1 Enable the CMB on each emulator.

2 Reset each emulator.

3 Set up the first emulator to drive the CMB trigger.

4 Set up the second emulator to receive the CMB trigger.

5 Start a trace on each emulation-bus analyzer.

6 Start a run on each emulator.

In the above steps, you set one emulation-bus analyzer to drive the CMB trigger, and set another to trigger on receipt of a CMB trigger. You can use the same concepts to trigger external instruments using the BNC connector on the rear panel of the HP 64700 Series Card Cage.

To break to the monitor on an analyzer trigger signal

1. Enter the emulation configuration.
2. Set the emulator to receive trig1.
3. Set the emulation-bus analyzer to drive trig1.
4. Specify the trigger conditions for the trace.
5. Start the trace.
6. Start the program run.

The trigger signals and the analyzer trigger capabilities allow you to specify breakpoints. You can use the trigger specification to specify complex sequences of address, data and status, then break the program to the monitor when the sequence is found. This is useful when you want to examine memory locations and registers after the trigger condition occurs, but before further program execution.

You can use a similar process to break to monitor when a BNC trigger or CMB trigger is received.

7

Making Software Performance Measurements

How to make software performance measurements on your programs

Using the Software Performance Measurement Tool

The Software Performance Measurement Tool (SPMT) is a feature included in the emulator/analyzer that allows you to make software performance measurements on your programs. Two types of software performance measurements can be made with the SPMT: activity measurements, and duration measurements.

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.

Use the Software Performance Analyzer (SPA) for more capability

For more capability in making measurements of the performance of your software, you can order the Software Performance Analyzer (SPA). SPA helps designers understand the execution of software modules in an absolute file.

SPA provides answers to questions such as:

- Why does it take so long to execute a program?
- Which modules are taking extra long time to execute?

While SPA performs a measurement, it shows the current measurement results. There is no need for you to transfer files; all you do is indicate the type of display desired (histogram or table listing). If you are interested in purchasing SPA, contact your HP Sales Representative.

Understanding activity 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 (such as reads, writes, and stack pushes).

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 in the next figure.

```
 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%  ******
```

Memory and Program Activity

```
         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
```

Memory and Program Activity (Cont’d)

Understanding duration measurements

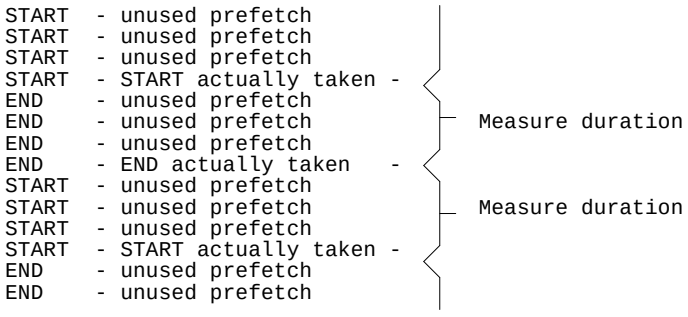
Duration measurements provide a best-case/worst-case characterization of code execution time. These measurements record execution times that fall within a set of specified time ranges. The **trace** command is set up to store only the entry and exit states of the module to be measured (for example, a 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.

Before you perform duration measurements, you should be aware of the prefetch and recursion considerations associated with these measurements.

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. Prefetches or 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 as shown in the prefetch correction listing. All of the entry and exit addresses which precede these last addresses are assumed to be unused prefetches, and are ignored during time measurements.



The SPMT makes its duration measurements from the last start address in the series of start addresses, to the last end address in the series of end addresses. The other start and end addresses are unused prefetches and are ignored by the software of the SPMT. Recursive procedures will still affect the accuracy of your measurements.

The prefetch correction has the following consequences:

- Prefetches are ignored. They do not affect the accuracy of the measurement in process.
- 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.

If you are making duration measurements on a function that is recursive, or one that has multiple entry and/or exit points, the result may be invalid information.

To use the Software Performance Measurement Tool

Activity and duration measurements are made with the SPMT in a five-step process, summarized as follows:

1 Set up the trace command.

2 Initialize the performance measurement.

3 Run the performance measurement.

4 End the performance measurement.

5 Generate the performance measurement report.

Step 1. Set up the trace command

Before you initialize and run performance measurements, the current trace command (the last trace command entered) must be properly set up.

1 Increase the trace depth to the maximum number by entering:

```
display trace depth 512
```

In general, you want to give the SPMT as many trace states as possible to post-process to increase statistical accuracy. Also it is important that "time" be counted by the analyzer; otherwise, the SPMT measurements will not be correct.

2 Choose to make either activity measurements or duration measurements.

- To make activity measurements (which measures activity as a percentage of all activity, the current trace command should be the default), enter:

```
trace counting time
```

The default trace command triggers on any state, and all captured states are stored. Also, since states are stored "after" the trigger state, the maximum number of captured states appears in each trace list.

You can use trace commands other than the default. 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.

- To make duration measurements, set up the trace command to store only the entry and exit points of the module of interest. For example:

```
trace after <symbol_entry> or <symbol_exit> only
symbol_entry or symbol_exit counting time
```

or

```
trace after <module_name> start or module_name end only
module_name start or module_name end counting time
```

Since the trigger state is always stored, you should trigger on the entry or exit points.

<symbol_entry> and <symbol_exit> are symbols from the user program.

<module_name> is the name of a C function or Pascal procedure (and is listed as a procedure symbol in the global symbol display).

Step 2. Initialize the performance measurement

After you set up the trace command, you must tell the SPMT the address ranges on which you wish to make activity measurements or the time ranges to be used in the duration measurement. This is done by initializing the performance measurement, which can be accomplished in various ways.

- To use the default configuration, enter the following command with no options:

```
performance_measurement_initialize
```

This 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.

- To initialize with user-defined files (activity or duration measurement), specify the SPMT address or time ranges to use by placing the information in a file and entering the file name in the **performance_measurement_initialize** command.

  The formats for the address range file (activity measurements) and time range file (duration measurements) are described in this chapter.

- To include program symbols (procedure name or static), user defined address ranges, and comments in address range files, refer to this example file:

```
# 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 through.  The above will define a range which starts with symbol1
# and goes through symbol2.  If both symbols are procedures then the range will
# be defined as the start of symbol1 through the end of symbol2.

dir1/dir2/source_file.s:local_symbol

# The above defines a range based on the address of local_symbol.
```

- To include comments and units for time ranges in time range files, refer to this example file:

```
# 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.
```

Time units can be in microseconds (us), milliseconds (ms), or seconds (s).

- To select duration measurements, enter:

  performance_measurement_initialize duration

  or

  performance_measurement_initialize <FILE> ***duration***

  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
```

- To initialize with global symbols, enter:

  performance_measurement_initialize

  or

  performance_measurement_initialize global_symbols

  Global symbols in the symbols database becomes 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.

- To initialize with local symbols, enter:

  performance_measurement_initialize local_symbols_in `<source file name>`

  The symbols associated with the 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.

  These are example commands showing performance measurement initialization with local symbols.

  performance_measurement_initialize local_symbols_in `spmt_demo.C:`

  performance_measurement_initialize local_symbols_in `spmt_demo.C:math_library`

  performance_measurement_initialize local_symbols_in `math_library`

- To restore the current measurement, enter:

  performance_measurement_initialize restore

  This 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 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 measurement. When you restore a performance measurement, make sure your current trace command is identical to the command used with the restored measurement.

When restoring old performance measurement data, the "restore" option determines if the current emulator software version matches the version used when the performance measurement data was stored (in the **perf.out** files). If the versions match, the restore will be performed. 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.

Step 3. Run the performance measurement

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.

- To process the current trace data, enter:

  performance_measurement_run

- To execute the current trace command consecutively, a certain number of times, enter:

  performance_measurement_run <COUNT>

  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 your results will be. At least four consecutive traces are required to obtain statistical interpretation of activity measurement results.

Step 4. End the performance measurement

- To end the performance measurement, enter:

  performance_measurement_end

  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.

  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.

Step 5. Generate the performance measurement report

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**.

- To save the current performance measurement information in a file called "perf1.out", and produce a histogram showing only the program activity occupied by the functions and variables.

```
mv perf.out perf1.out
perf32 -hpf perf1.out
```

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:

-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.
-c	Print only program and memory activity information consuming time.

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.

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.

- To interpret reports of activity measurements, understand the information described here. You will see this information in activity measurement reports.

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 (like reads and writes to memory and stack pushes).
Relative	A count or time value associated with activity in address ranges in the performance measurement.
Absolute	A count or time value associated with all trace state activity, not just activity in the address ranges defined for the performance measurement.
Mean	Average number of analyzer trace states in the range specified. 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. Some compilers emit more than one symbol for certain addresses. For example, a compiler may emit "math_library" and "_math_library" for the first address in a routine named math_library. 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 "math_library" example, it may show either "math_library" or "_math_library" 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 (the mean sample).

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.

- To interpret reports of duration measurements, understand the information described here. You will see this information in duration measurement reports.

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 (for example, mean of the average times in each time range).

8

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.
- Setting up the emulation monitor.
- Mapping memory.
- Modifying the general configuration items.
- Selecting analyzer trace options.
- Configuring simulated I/O.
- Specifying connections for interactive measurements.

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

The interactive measurement configuration options are described in this chapter, and additional information is given in Chapter 6, "Making Coordinated Measurements".

Using the Configuration Interface

This section shows you how to set up, modify, and store emulation 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 context sensitive (f1) help.
- Exit the configuration interface.

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

When you have developed an emulation configuration, saved it to a file, and closed the configuration interface, you can use the **File→Load→Emulator Config...** command and associated dialog box in the top level emulator/analyzer interface to load the configuration file into your emulator.

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.

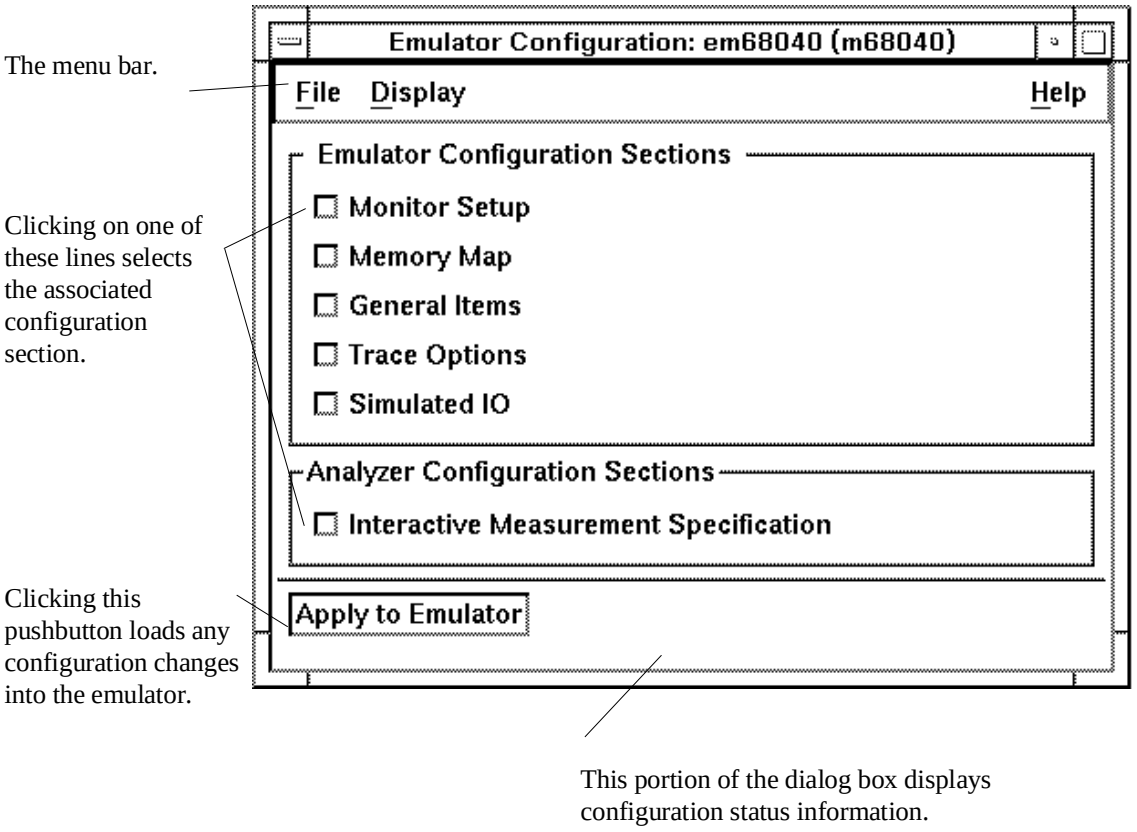
The configuration sections that are presented depend on the hardware and the features of your particular emulator.

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

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

Examples

The 68040 emulator configuration interface top-level dialog box is shown below.



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 "Monitor Setup" section are the first to be asked.

To access the memory map and define, modify, or delete map entries in the "Memory Map" section, answer "yes" to the "Modify memory configuration?" question.

To access the questions in the "General Items" section, answer "yes" to the "Modify emulator pod configuration?" question.

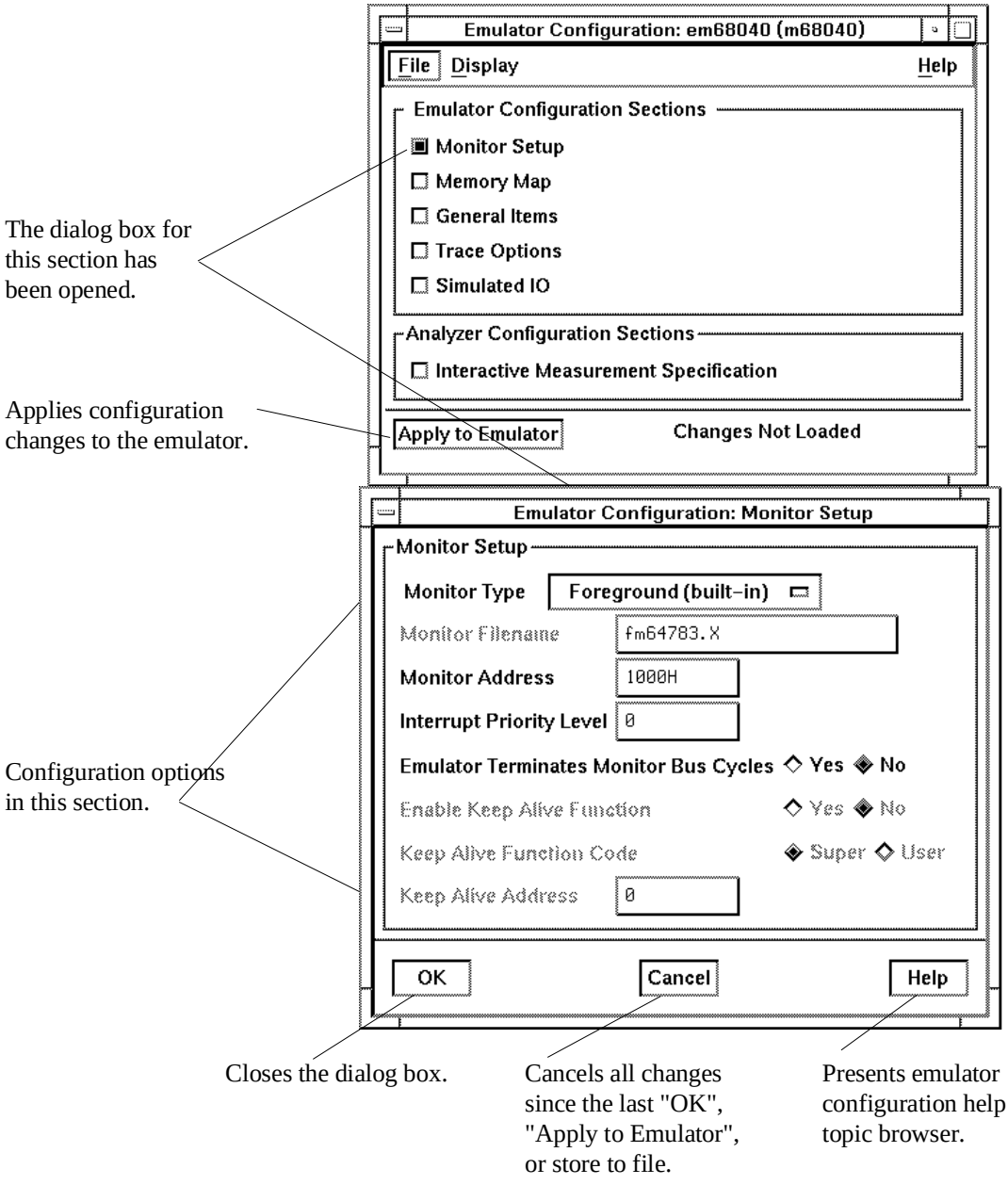
To access the questions in the "Trace Options" section, answer "yes" to the "Modify debug/trace options?" question.

To access the questions in the "Simulated IO" section, answer "yes" to the "Modify simulated I/O configuration?" question.

To access the questions in the "Interactive Measurement Specification" under "Analyzer Configuration" section, answer "yes" to the "Modify interactive measurement specification?" question.

Examples

Most configuration sections provide dialog boxes similar to the following.



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 box).

To apply configuration changes to the emulator

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

  This loads the configuration changes into the emulator. Status text to the right shows whether or not the load was successful.

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

  The "Apply to Emulator" pushbutton does not create a file to store configuration changes. To do that, choose the **File→Store...** pulldown in the top level interface window, described later.

  If you exit the configuration interface with configuration changes that have not been stored, you will be asked whether you want to store the changes, exit without storing, or cancel the exit.

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're using the Softkey Interface, the last configuration question, "Configuration file name?", lets you name the file to which configuration information is stored. If you don't enter a name, configuration information is saved to a temporary file (which is deleted when you exit the interface and release the emulation system).

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

Configuration information is saved in a file with an extension of ".EA".

When using the Graphical User Interface, do not try to modify the ".EA". It may not load correctly after modification. Instead, start the configuration interface, make desired modifications, and store the configuration file to a new, or existing, filename.

For more information on how to use dialog boxes, refer to the "Entering Commands", and "Using Special Features of the Graphical User Interface" sections in Chapter 3, "Using the Emulator/Analyzer Interface".

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.

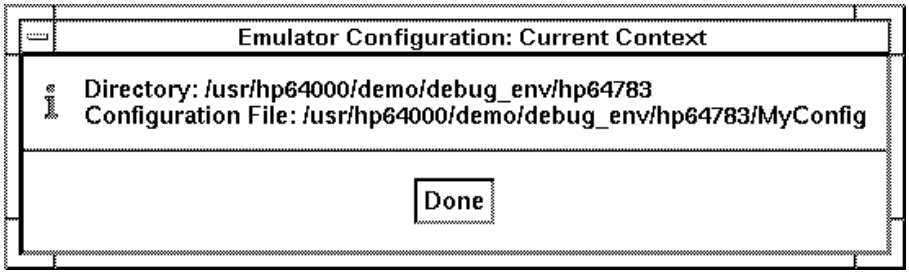
  The directory context specifies the directory to which configuration files are stored and from which they are loaded.

  For more information on how to use dialog boxes, refer to the "Entering Commands", and "Using Special Features of the Graphical User Interface" sections in Chapter 3, "Using the Emulator/Analyzer Interface".

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" pushbutton.

To access context sensitive (f1) help

- Place the mouse pointer over the item for which you want help, and press the f1 keyboard key.

- Choose **Help→On Item...** from the pulldown menu in the top-level configuration interface window. Notice that the mouse pointer changes to a question mark. Move the question mark over the item for which you want help in the top-level configuration interface window, and click the *select* mouse button.

  If you are having trouble using the f1 key to obtain help, you may be able to use the question mark obtained by the **Help→On Item...** pulldown. If the **Help→On Item...** pulldown does not obtain the desired result, try the f1 key. The operation of these two selections differs on different platforms.

  In some dialog boxes, the question mark obtained by the **Help→On Item...** pulldown may not obtain a help screen when you place it on a command name, but the help screen may be obtained when you place the question mark over an input field or pushbutton associated with the command name.

To exit the configuration interface

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

  This will close the top-level configuration interface window together with all of the children of the top-level configuration interface window.

  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 a configuration

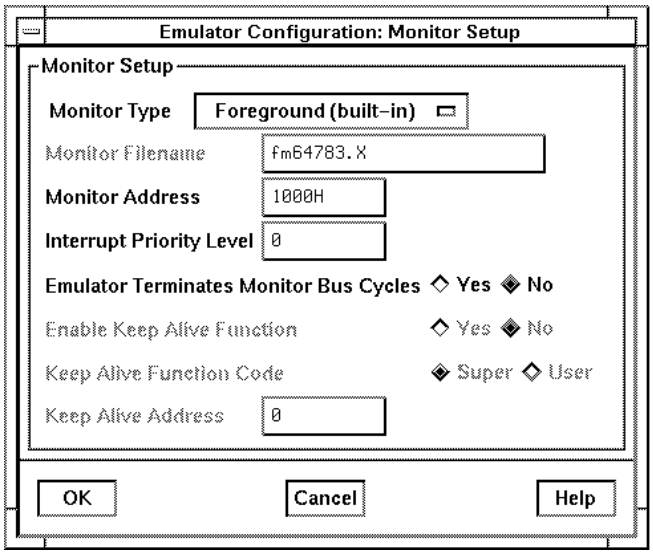
- In the emulator/analyzer interface, choose **File→Load→Emulator Config...** from the pulldown menu, and use the file selection dialog box to specify the configuration file to be loaded.

- 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.

Modifying the Monitor Setup

In order to modify the monitor setup, you must obtain the top-level Emulator Configuration dialog box. Simply click on the Monitor Setup pushbutton. The Monitor setup dialog box will appear.



This section shows you how to:

- Select the monitor type.
- Select the monitor filename.
- Select the monitor address.
- Select the monitor interrupt priority level.
- Select whether or not the emulator will terminate monitor bus cycles without regard to the state of the target system.
- Select whether or not there will be a keep-alive function, and if there will be a keep-alive function, select its address and function code, if desired.

To select the monitor type

- Choose "Background", "Foreground (built-in)", Foreground (customized)", or "None" for the "Monitor Type" configuration option.

  Choosing "Background" specifies that the background monitor will be used with the emulator. The background monitor is useful when you are first plugging into a target system. It occupies no address space that might be used by your target program. You cannot enable the MMU, use the processor caches, or perform dma cycles when the background monitor is in use, or when your system is arbitrating the bus. Also, when one of the routines of the background monitor is executing, the emulator cannot service any target system interrupts, even NMI interrupts.

  Choosing "Foreground (built-in)" specifies that the foreground monitor shipped with your emulator will be used. A foreground monitor must be used when the memory management unit or the caches of the MC68040 (or both) are enabled or bus arbitration is performed. Also, the emulator can be configured to service target system interrupts of any desired level during execution of foreground monitor routines.

  Choosing "Foreground (customized)" specifies that a custom foreground monitor will be used. With this selection, you will need to specify the Monitor Filename in this dialog box.

  Choosing "None" specifies that no monitor will be used. This option is useful when you are first connecting the emulator to a target system (refer to Chapter 18, "Connecting the Emulator to a Target System"). Sometimes the task of connecting an emulator to a target system can be complicated by characteristics of the emulation monitor. For example, foreground monitor bus cycles are visible to the target system. By selecting "None", you eliminate the question "am I having trouble connecting to my target system because of something the monitor is doing?"

  When you choose "None", you will be able to run the emulator from reset (if you previously loaded a program), and you will be able to take a trace with the analyzer to see what activity is being executed by your emulator. You will not be able to use any of the other emulator capabilities and features (such as loading a program or displaying memory). When your system is running successfully with the "None" selection, then choose one of the other monitor options to see if your target system will operate with the emulation monitor.

To select the monitor filename

- Type in the name of the custom foreground monitor in the text entry area beside "Monitor Filename".

  The custom foreground monitor absolute file will be automatically loaded after configuration is complete. The monitor must already be linked to the desired location and should not be linked with any of your target programs.

  The location for the monitor source file is: **/usr/hp64000/monitor/fm64783.s**.

  The file format for the monitor MUST be HP64000 absolute format (file.X).

  If using the HP 68030/68040 assembler/linker (B1465), use the -h option.

  If using Microtec Research, Inc. assembler/linkers, use the -h option.

  For other language systems, use the "HP 64000 Hosted Development System Absolute File Translator" program.

  No symbols are required for loading the custom foreground monitor.

To select the monitor address

- Type the base address of the foreground monitor in the text entry field beside "Monitor Address" in the dialog box.

  Enter a hexadecimal address on a 4-Kbyte boundary (XXXXX000h).

  If you are using a foreground monitor (either the built-in foreground monitor or a custom foreground monitor), you must set the base address where the monitor will be loaded.

  The emulator loads the foreground monitor into the 4-Kbyte block of dual-port emulation memory. It resets the memory map, and creates a map term at the address you enter in this dialog box. You cannot delete or alter this map term by using the map configuration commands. Instead, you must change the monitor configuration using this Monitor Setup dialog box.

  If the memory management feature of the MC68040 emulator is enabled, be sure the foreground monitor is mapped in an area that is translated 1:1, and it is not write-protected. Refer to the end of this chapter for instructions on how to map the foreground monitor to appropriate address space.

To select the monitor interrupt priority level

- Type in the desired interrupt priority level for your foreground monitor in the text entry area beside "Interrupt Priority Level".

  Enter a number from 0 to 7 in the text entry area. This is the interrupt priority level that will be held off by the emulation monitor during monitor execution. Interrupts having values higher than the number you enter here will be recognized by the processor.

  Set the interrupt priority level low enough to allow your target system to function correctly, but high enough to avoid excessive interrupt processing. The default value of 0 allows all target system interrupts to be recognized by the foreground monitor.

  The emulator uses a level 7, non-maskable interrupt to interrupt the target system and break into the monitor. When the foreground monitor is not executing critical code (such as monitor entry and exit), the foreground monitor will set the interrupt priority mask to the value you enter beside "Interrupt Priority Level", or to the interrupt level that was in effect before monitor entry, whichever is greater.

Example

Suppose your target system has a disk device driver that uses interrupt level 5, and the service routine must be run to prevent target system damage. To allow interrupts of higher priority than level 4 to be serviced during foreground monitor execution, enter 4 beside "Interrupt Priority Level". In this case, all interrupts with values of 4 or less will be ignored when the foreground monitor is executing.

To select whether or not the emulator will terminate monitor bus cycles

- Choose "Yes" or "No" for the "Emulator Terminates Monitor Bus Cycles" configuration option.

  Choosing "Yes" specifies that the emulator will terminate foreground monitor cycles using emulator-generated cycle termination signals. Cycle termination signals generated by the target system during access to the foreground monitor, including TEA, will be ignored.

  Choosing "No" specifies that foreground monitor cycles will be terminated when the target system TA, or TEA, or both signals are asserted.

  This configuration item only applies to the map term assigned to the foreground emulation monitor. If you choose "No", and the emulation monitor is in an address range where the target system does not return TA or TEA, the emulator will stop. If this happens, reset the emulation processor, and then choose "Yes" for this configuration option.

  Foreground monitor bus cycles are visible to the target system. If you choose "Yes", your target system may operate erratically if it is not expecting the emulation monitor bus cycles.

To select if there will be a keep-alive function, its address, and function code

- Choose "Yes" or "No" for the "Enable Keep Alive Function" configuration option (available for use with the background monitor).

  Choosing "Yes" specifies that a selected address will be read by the background monitor periodically. You must also specify the address to be read, and the function code used when reading that address, as follows:

  - Choose "Super" or "User" for the "Keep Alive Function Code" configuration option. This determines whether the supervisor function code or the user function code will be used when the emulator reads the keep-alive address.
  - Type in the keep-alive address in the text entry area beside "Keep Alive Address". This address in target memory will be read periodically during background monitor execution. The read accesses to the target memory address can be used to avoid a timeout of a target system bus or a watchdog timer during background monitor execution.

  Choosing "No" specifies that there will be no keep-alive memory read cycles in target memory during background monitor operation.

Mapping Memory

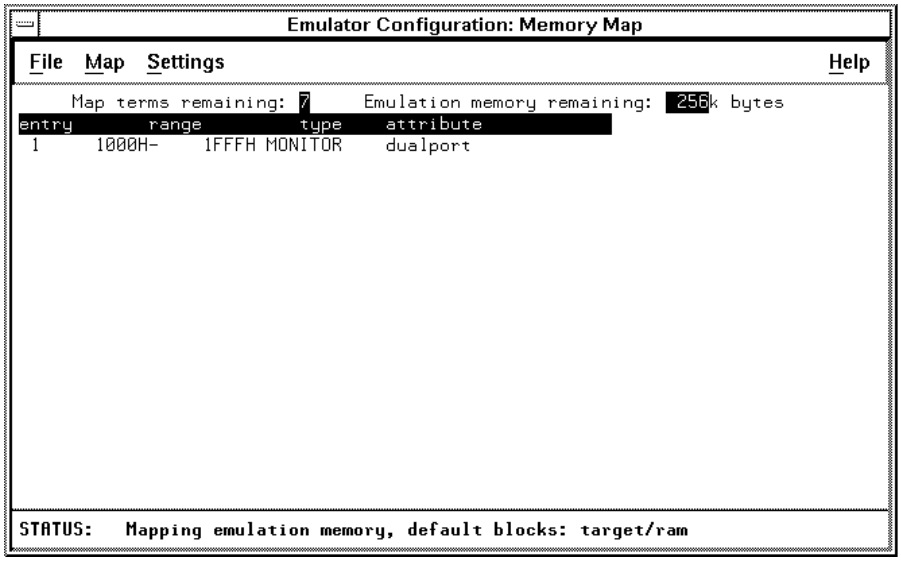
Because the emulator can use target system memory or emulation memory (or both), you must map the available ranges of memory so that the emulator knows where to direct its accesses.

All memory ranges used by your programs must be specified in the memory map before you load programs into memory.

Up to eight ranges of memory can be mapped, and the resolution of mapped ranges is 256 bytes, that is, the memory ranges must begin on 256-byte boundaries (numbers ending in 00h) and must end on 256-byte boundaries (numbers ending in FFh).

Emulation memory is made available to the mapper in 256-byte blocks. When you map an address range to emulation memory, at least one 256-byte block is assigned to the range. When a block of emulation memory is assigned to a range, it is no longer available, even though part of the block may be unused.

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



This section shows you how to:

- Add memory map entries.
- Modify memory map entries.
- Delete memory map entries.
- Characterize mapped and unmapped ranges.
- Specify whether or not read data will be inhibited from being loaded into the caches during transactions in the associated memory range.
- Map memory ranges in which the emulator will terminate bus cycles without regard to the state of the target system.
- Map memory ranges to be stored within the dual port memory.

To add memory map entries

- Choose **Map→Add New Entry** from the pulldown menu in the memory map window.

- Press and hold the *select* mouse button and choose **Add New Entry** from the popup menu.

- Using the command line, enter the address range, memory type, and possibly an **emulator_terminates_bus_cycles**, or **transfer_cache_inhibit** attribute, or both attributes for each emulation memory range.

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

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

  Writes to locations characterized as ROM will cause emulator execution to break into the monitor program if you choose "Yes" for the "Break processor on Write to ROM" option in the "Emulator Configuration: General Items" dialog box.

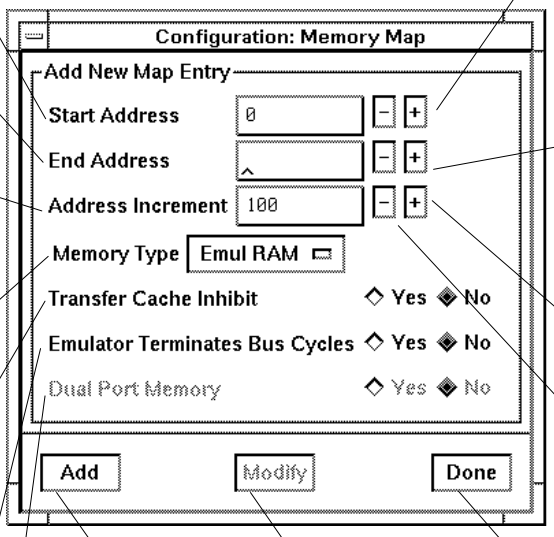
  RAM memory in the emulation or target system will be changed by processor writes, even if that memory has been characterized as ROM.

  You can include the transfer cache inhibit attribute with any memory range. If included, no data will be loaded into either the instruction cache or data cache during any transactions occuring in the associated memory range. This ensures that all activity will appear on the emulation bus and be available for tracing with the emulation-bus analyzer.

  You can include the "Emulator Terminates Bus Cycles" attribute with any memory range. If included, the emulator will terminate bus cycles without regard to the state of the target system. This is useful in ranges where the $\overline{TA}$, or $\overline{TEA}$, or both signals are not available from the target system. The danger of this option is that the emulator may become out of sync with the target system if the target system supplies these signals.

  You can specify that a particular address range be loaded into the dual-port memory if using of a background monitor in the Monitor Setup dialog box.

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



The starting address of the range to be added.

The ending address of the range to be added.

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

Specifies type of memory occupied by this range.

Specify if read data is inhibited from being loaded into the caches in this range.

Specify if the emulator terminates bus cysles in this range without waiting for the target system.

Only available when the background monitor is in use.

Adds the defined range to the memory map.

Inactive in the "Add" mode of map entry.

Closes the dialog box.

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

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

Multiply or divide the increment value by 2.

These pushbuttons may be held down to repeat the action.

Examples

Example 1: Suppose you're using the emulator in-circuit, and there is a 12-byte I/O port at 1c000 hex in your target system. You have ROM in your target system from 0 through ffff hex. Also, you want to use the dual-port emulation memory at 20000 hex. You could use the Memory Map dialog box to create the following three map entries:

Start Address **1c000h**, End Address **1c0ffh**, Memory Type **Target RAM**
Start Address **0h**, End Address **0ffffh**, Memory Type **Target ROM**
Start Address **20000h**, End Address **20fffh**, Memory Type **Emul RAM**, Dual Port Memory = **Yes**

Using the command line, you would enter:

```
1c000h thru 1c0ffh target ram
0 thru 0ffffh target rom
20000h thru 20fffh emulation ram dualport
```

Remember that the only way to make the dual-port emulation memory available for your target program is to use the background monitor. When a foreground monitor is in use, it occupies the dual-port emulation memory, by default.

Example 2: This second example shows the relationship between memory ranges and the block sizes of memory. Suppose you have installed 256-Kbyte SRAM memory modules in Memory slots 0 and 1 (called BANK 0 and BANK 1) on the emulation probe. This makes four 64-Kbyte blocks and two 128-Kbyte blocks available to the memory mapper. Then you enter the following map commands:

Start Address **0h**, End Address **7fffh**, Memory Type **Emul RAM**
Start Address **20000h**, End Address **3f000h**, Memory Type **Emul RAM**
Start Address **40000h**, End Address **4ffffh**, Memory Type **Emul RAM**
Start Address **50000h**, End Address **500ffh**, Memory Type **Emul RAM**
Map→Default Memory Type→Target RAM→Transfer Cache Inhibit ON

Using the command line, you would enter:

```
0 thru 7fffh emulation ram
20000h thru 3f000h emulation ram
40000h thru 4ffffh emulation ram
50000h thru 500ffh emulation ram
default target ram transfer_cache_inhibit
```

If you haven't used the dual-port emulation RAM, the first map term that is small enough to fit is assigned to that memory. In this example, that is the last term you

defined (the range from 50000..500ff). The entire 4-Kbyte block is reserved though you specified only a 256-byte range. Two 64-Kbyte blocks and one 128-Kbyte block are used from the SRAM emulation memory on the probe, leaving two 64-Kbyte blocks and one 128-Kbyte block. One of the 64-Kbyte blocks is used for the first map term, but 32 Kbytes of that block are unused and unavailable. The third term uses the other 64-Kbyte block. The second term uses part of the 128-Kbyte block, leaving the rest unavailable.

Mapper resolution is independent of block allocation. In the above example, if you had **default guarded** and your program accessed 8000h, the emulator would do a guarded memory break.

To modify memory map entries

- Choose **Map→Modify Entry** from the pulldown menu in the memory map window and select the entry number from the cascade menu.

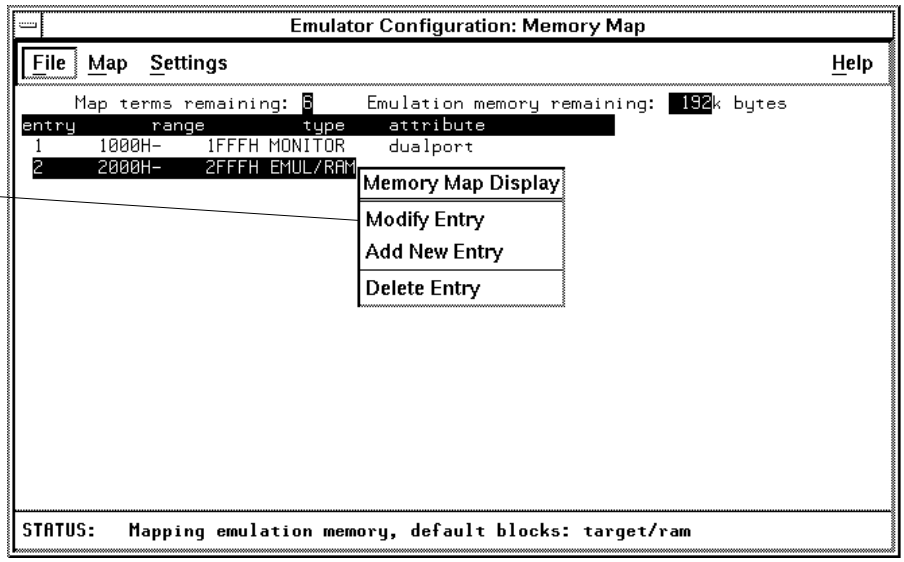
- Position the mouse pointer over the entry you wish to modify. Press and hold the *select* mouse button, and choose **Modify Entry** from the popup menu.

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

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

Examples

To modify a memory map entry using the popup menu:



Bring up the menu and choose this item to modify the highlighted memory map entry.

Use the Modify Map Entry dialog box (same as the Add New Map Entry dialog box) to modify the entry. Click the "Modify" pushbutton to modify the selected range in the memory map according to changes you make in the Modify Map Entry dialog box.

To delete memory map entries

- Choose **Map→Delete Entry** from the pulldown menu in the memory map window and select the entry number from the cascade menu.
- Position the mouse pointer over the entry you wish to delete. Press and hold the *select* mouse button and choose **Delete Entry** from the popup menu.
- Using the command line, enter the **delete <ENTRY#>** command.

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

To characterize unmapped ranges

- Choose **Map→Default Memory Type** from the pulldown menu in the memory map window and select the memory type from the cascade menu. If you **choose Target RAM** or **Target ROM**, you must also choose **Transfer Cache Inhibit OFF** or **Transfer Cache Inhibit ON**.
  - If you choose **Transfer Cache Inhibit OFF**, transactions that are sent to unmapped memory may also be loaded into the instruction cache, data cache, or both caches.
  - If you choose **Transfer Cache Inhibit ON**, no data that is sent to unmapped memory will be written into the caches.
- Using the command line, enter the **default <memory_type>** command.

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

To map memory ranges in which data is not loaded into the caches

- Choose "Yes" or "No" for the "Transfer Cache Inhibit" configuration option in the "Modify Map Entry" or "Add New Entry" dialog box.

  Choosing "Yes" specifies that no data will be loaded into either the instruction cache or data cache during any transactions occuring in the associated memory range. This choice is useful when you need to have all transactions appear on the external buses to allow the emulation-bus analyzer to capture complete traces of processor activity.

  Choosing "No" specifies that data will be loaded into either the instruction cache or data cache, as applicable, during transactions occurring in the associated memory range. This choice is useful when you need to have transactions completed in the fastest and most efficient manner. Use of processor caches increases the processor speed of execution.

To map memory in which the emulator will terminate bus cycles

- Choose "Yes" or "No" for the "Emulator Terminates Bus Cycles" configuration option in the "Modify Map Entry" or "Add New Entry" dialog box.

  Choosing "Yes" causes the emulator to terminate bus cycles without regard to the state of the target system. This is useful in ranges where the TA or TEA or both signals from the target system are not available. The danger of including this option is that the emulator may become out of sync with the target system if the target system provides these signals.

  Choosing "No" ensures that the timing of cycle termination signals will not cause the emulator and target system to become out of sync. No emulation bus cycle will be terminated until the TA or TEA signal is received from the target system.

To map memory to be stored within the dual-port memory

- Choose "Yes" or "No" for the "Dual Port Memory" configuration option in the "Modify Map Entry" or "Add New Entry" dialog box (available only when using the background monitor).

  Choosing "Yes" specifies that the map term must be stored in the 4-Kbyte dual-port emulation memory.

  Choosing "No" specifies that the map term must be stored in either emulation memory or target system memory, according to specifications made for its address range in the memory map.

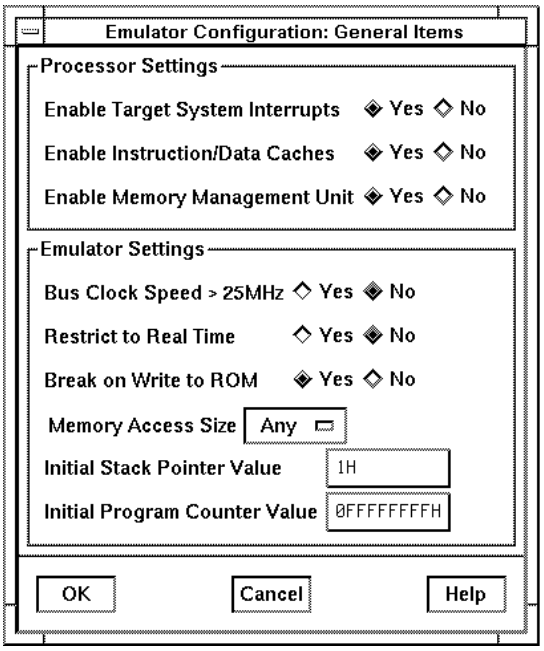
  This block can also be mapped by specifying the **dualport** attribute after the map address and memory type specification on the command line.

  There is one 4-Kbyte block of dual-port emulation memory on the emulator probe. (Dual-port means the emulation controller can access memory locations without interfering with program execution). If you use a foreground monitor, the monitor will be loaded into this space and you won't be able to map this memory for any other purpose.

  If you specify an address range less than 4 Kbytes to be placed in the dual-port memory, all 4 Kbytes of the dual-port memory will be allocated because that is the minimum block size for that memory. If you specify a block size less than 4 Kbytes and the dual-port memory is unmapped, the emulator will use that memory to more closely match the requested address range to the block size.

Configuring the Emulator General Items Screen

In order to configure the emulator pod, you must first start the configuration interface and access the "General Items" configuration section (refer to "Using the Configuration Interface" earlier in this chapter).



This section shows you how to:

- Configure items that affect operation of the emulation processor, such as:
  - Enable/disable target system interrupts.
  - Enable/disable the instruction and data caches.
  - Enable/disable the memory management unit (MMU).
- Configure items that affect operation of the emulator, such as:
  - Specify whether or not the bus clock speed is greater than 25 MHz.
  - Restrict the emulator to real time runs.

- Break from target program to monitor execution when the emulator detects an attempt to write to a ROM address.
- Set the memory access size.
- Set the initial value of the stack pointer.
- Set the initial value of the program counter.

To enable/disable target system interrupts

- Choose "Yes" or "No" for the "Enable Target System Interrupts" configuration option.

  Choosing "Yes" allows target system interrupts to be received by the emulation processor. This is useful to test whether your target system interrupt logic works correctly after the interrupt service routines have been designed and the interrupt vectors have been assigned.

  Choosing "No" causes all target system interrupts to be ignored by the emulation processor. You may want to disable target system interrupts if your target system interrupt logic doesn't work correctly or isn't finished. Target system interrupts are always ignored during execution of certain critical routines of the foreground monitor, such as monitor entry and monitor exit, and are always ignored in the background monitor.

To enable/disable the instruction and data caches

- Choose "Yes" or "No" for the "Enable Instruction/Data Caches" configuration option.

  Choosing "Yes" allows the instruction and data caches to be enabled. With this choice, the caches can still be disabled in selected memory ranges as specified when mapping memory.

  Choosing "No" causes the emulator to assert the $\overline{\text{CDIS}}$ signal to prevent instructions and data from being loaded into the respective caches during target program execution. This overrides any specifications you may make for individual entries on the memory map.

  When you disable the instruction and data caches, all activity appears on the emulation processor buses where it can be monitored and captured by the emulation-bus analyzer. When you allow the caches to be enabled, program execution is faster, but only partial information is available to be traced by the emulation-bus analyzer. This may cause confusing trace displays or failure to trigger, especially if the code being analyzed is a small loop where all the instructions and operands fit into cache and registers.

  When you are more concerned about measuring processor performance, you should enable the caches. If you are making analyzer measurements at the same time, you may need to experiment to find suitable trigger combinations.

  If you need to disable caching only for accesses to a specific memory block, enter that as part of your specification when defining the corresponding memory map term. This allows you to capture analysis information for specific memory ranges without dramatically affecting overall system performance.

To enable/disable the memory management unit (MMU)

- Choose "Yes" or "No" for the "Enable Memory Management Unit" configuration option.

  Choosing "Yes" allows the MMU of the emulation processor to control placement of the target program in physical memory. The target system will be able to enable and disable the MMU during program execution by using the MDIS signal.

  Choosing "No" causes the emulator to disable the MMU of the emulation processor by asserting the MDIS signal.

  The MC68040 MMU can manage a program that occupies a large space in logical (virtual) memory while running it from a much smaller space in physical memory. When you operate the emulation processor with the MMU enabled, you must ensure that the foreground monitor is contained in memory space that:

  - is not write-protected.
  - is mapped 1:1 (logical address = physical address). The reason that this mapping is important is that the MMU may be enabled or disabled at any time during program execution; whether or not the MMU is enabled, the emulator must be able to enter the foreground monitor to provide emulation features. Refer to the section titled "Mapping The Foreground Monitor For Use With The MC68040 MMUs" later in this chapter.

To specify whether the clock speed of the emulation bus is greater than 25 MHz

- Choose "Yes" or "No" for the "Bus Clock Speed >25MHz" configuration option.

  Choosing "Yes" causes the emulator to add one wait state to each synchronous and burst memory access.

  Choosing "No" allows all synchronous and burst accesses to be completed at full processor speed with no wait states.

  When the external bus clock (BCLK) is operating at a frequency above 25 MHz, this question must be answered "Yes".

  When operating above 25 MHz, the target system is responsible for adding a wait state to its accesses. The emulator will not attempt to add a wait state to target accessses, other than to ignore cycle terminations until a wait state has passed. The target system is responsible for making sure cycle terminations and data are valid after the wait state.

  The 4-Mbyte memory modules are not as fast as the 256-Kbyte and 1-Mbyte memory modules. The emulator always adds one wait state to accessess to emulation memory when it detects the presence of any 4-Mbyte memory modules on the emulation probe.

To restrict the emulator to real-time runs

- Choose "Yes" or "No" for the "Restrict to Real Time" configuration option.

  Choosing "Yes" causes the emulator to offer only a limited set of emulation features: reset, break, run, and step.

  Choosing "No" allows the emulator to offer its complete emulation feature set at any time during execution of your target program.

CAUTION If your target system could be damaged because the emulator is interrupted while running critical routines, choose "Yes" for this configuration option.

The emulator uses the emulation monitor program to implement some features, such as displaying processor registers. When the emulation processor executes a monitor routine, it is not executing your target program. This may cause problems in target systems that need real-time program execution (uninterrupted execution of the target program).

When you choose "Yes" for this configuration item, you must do an execution break in order to display registers or display target memory, and you will not be able to use simulated IO.

While this configuration item affects which commands will be accepted, it does not affect access breakpoints, such as break on write to ROM, break on analyzer trigger, or break on access to guarded memory. It also doesn't affect the emulator's response to execution breakpoints.

To enable/disable breaks on writes to ROM

- Choose "Yes" or "No" for the "Break on write to ROM" configuration option.

  Choosing "Yes" enables breaks on writes to ROM with the following results:

  - The emulator will stop executing the target program and begin execution in the emulation monitor whenever the target program attempts to write to a memory region mapped as ROM.
  - The emulator will modify the content of RAM memory that is mapped as ROM, even when write to ROM break is enabled.

  Choosing "No" disables breaks on writes to ROM. The emulator will continue to execute the target program even when it detects an attempt to write to an address mapped as ROM. Emulation writes will modify the content of RAM memory that has been mapped as ROM.

To specify the memory access size

- Choose "Any", "Bytes", "Words", or "Longs" for the "Memory Access Size" configuration option.

  Choose "Any" if you want the emulator to select the optimum access size for the transaction to be completed.

  Choose "Bytes" if the emulator should make only 8-bit accesses to memory.

  Choose "Words" if the emulator should make only 16-bit accesses to memory.

  Choose "Longs" if the emulator should make only 32-bit accesses to memory.

  When accessing memory locations, the access mode specifies the type of microprocessor cycles that are used to read or write the value(s). By default, "Any" is selected. In the "Any" mode, long-word accesses are made to memory, except when accessing an address not on a long-word boundary, or when only one byte or one word remains to be accessed. In these cases, the appropriate memory access mode ("Bytes" or "Words") will be used.

  If you choose the "Bytes" access mode, and a target system location is modified to contain the value 12345678H, byte instructions will be used to write the byte values 12H, 34H, 56H, and 78H to target system memory.

  If set to "Any", the size you include in your "display memory" or "modify memory" command will be used for the access. It will temporarily override the "Any" designation for the access. If set to "Bytes", "Words", or "Longs", the size selected in your "display memory" or "modify memory" commands will have no effect on the actual memory access; it will be what you specified for the memory access size.

To specify the initial value of the stack pointer

- Type in the address that is the initial value for the interrupt stack pointer in the text entry area beside "Interrupt Stack Pointer Value".

  Enter a 32-bit hexadecimal address for the initial value of the ISP. Normally, this value is the same as the value at memory address 0. The default value is 1H. This in an invalid value. It is given as the default to remind you to enter the correct hexadecimal address for the ISP before using the emulator.

  Normally, if you run the emulator from reset, the processor fetches the value at offset 0 from the vector table, and loads it into the interrupt stack pointer. There are cases where the interrupt stack pointer cannot be fetched from the reset vector table. For example, if you reset the emulator, break to the emulation monitor, and then run the emulator from the monitor, the stack pointer value will not be read from the normal location. In these cases, the stack pointer value will be read from the value you enter here.

To specify the initial value of the program counter

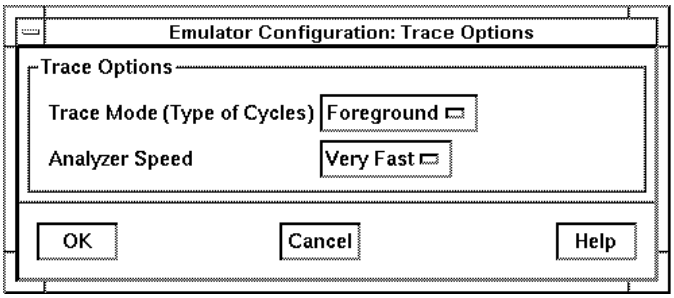
- Type in the address that is the initial value for the program counter in the text entry area beside "Initial Program Counter Value".

  Enter a 32-bit hexadecimal address for the initial value of the program counter. Normally, this value is the same as the value at memory address 4. The default value is 0FFFFFFFFH. This in an invalid value. It is given as the default to remind you to enter the correct hexadecimal address for the program counter before using the emulator.

  Normally, if you run the emulator from reset, the processor fetches the value at offset 4 from the vector table, and loads it into the program counter. There are cases where the program counter value cannot be fetched from the reset vector table. For example, if you reset the emulator, break to the emulation monitor, and then run the emulator from the monitor, the program counter value will not be read from the normal location. In these cases, the program counter value will be read from the value you enter here.

Setting the Trace Options

In order to set the trace options, you must first start the configuration interface and access the "Trace Options" configuration section (refer to the previous "Using the Configuration Interface" section).



This section shows you how to:

- Enable tracing emulation-bus activity in foreground address space, background address space (tracing execution of the background monitor), or both address spaces. This lets you select whether to include background monitor execution in analyzer traces when using the background monitor.
- Identify the data rate of your emulation system for the emulation-bus analyzer. The capabilites of the 1K analyzer differ with different data rates. The full capabilites of the deep analyzer are available at all data rates.

To include/exclude background monitor execution in the trace

- Choose "Foreground", "Background", or "Both" for the "Trace Mode (Type of Cycles)" configuration item.

  Choosing "Foreground" specifies that the analyzer trace only foreground cycles, including execution of your target program and of the foreground monitor, if you are using a foreground monitor. In this mode, the analyzer will not trace execution of a background monitor.

  Choosing "Background" specifies that the analyzer will trace only background cycles. This is rarely useful because it excludes target program execution.

  Choosing "Both" specifies that the analyzer trace both foreground and background cycles. This option allows all emulation processor cycles to be viewed in the trace display when you are using a background monitor.

To identify the data rate of your emulation system for the 1K analyzer

- If you are using the deep analyzer with your emulator, you can ignore this choice. The deep analyzer will provide its full capabilities regardless of the choice made here.

- If you are using the 1K analyzer, choose "Slow", "Fast", or "Very Fast" for the "Analyzer Speed" configuration item.

  Choosing "Slow" specifies that the burst data rate of the traced activity is not more than 16.67 MHz. The 1K analyzer can perform counts of selected states or counts of time between states when the traced data rate is "Slow".

  Choosing "Fast" specifies that the burst data rate of the traced activity is between 16.67 and 20.00 MHz. The 1K analyzer can perform counts of selected states when

the traced data rate is "Fast". It cannot perform counts of time between the traced states.

Choosing "Very Fast" specifies that the burst data rate of the traced activity is greater than 20.00 MHz. The 1K analyzer cannot perform any counts when the traced data rate is " Very Fast".

If burst cycles are not being used, set the "Analyzer Speed" to "Slow".

The MC68040 analyzer clock is set to "Very Fast" by default. The 1K analyzer can capture all types of bus cycles correctly up to the maximum clock rate of 40 MHz, but cannot correctly count states or time at higher speeds for certain bus cycle types.

The worst-case situation is one where a zero-wait state burst cycle is performed. The analyzer clock rate for burst cycles is given by the equation:

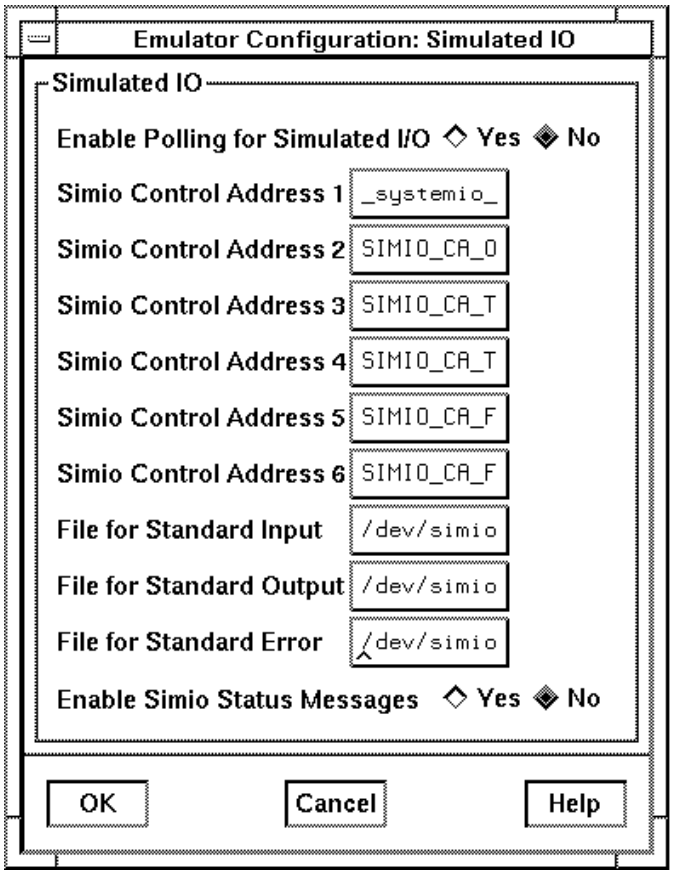
$$\text{Analyzer Clock Rate} = \frac{\text{Processor Clock Rate (BCLK)}}{(1 + \text{number of wait states})}$$

To determine the correct selection, calculate the maximum data rate by using the above equation. Remember that the emulator requires one wait state for all accesses when the external clock is greater than or equal to 25 MHz. Then choose the data rate option according to the data rate.

If no burst cycles are performed, the 1K analyzer clock speed can be set "Slow".

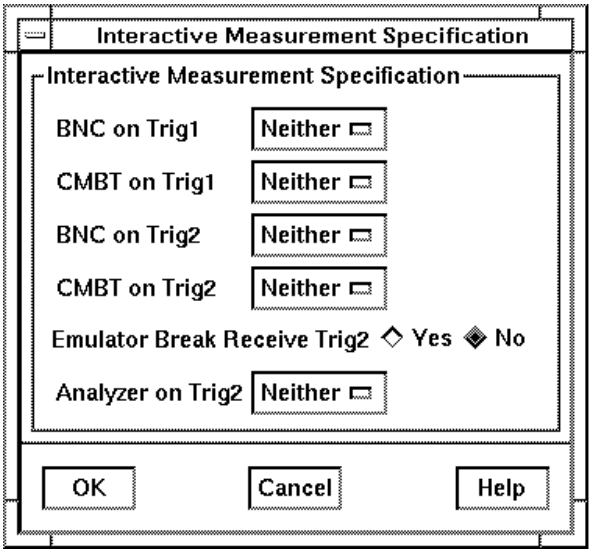
Modifying the Simulated IO Configuration Items

In order to modify the simulated I/O configuration items, you must first start the top-level configuration interface and select the Simulated IO pushbutton in the dialog box. The Simulated IO dialog box will appear on screen. Refer to the *Simulated I/O User's Guide* for details on configuring and using simulated I/O.



Modifying the Interactive Measurement Specification Configuration Items

In order to modify the interactive measurement configuration items, you must first start the configuration interface. In the top-level configuration interface dialog box, choose "Interactive Measurement Specification" under "Analyzer Configuration Sections".



This section shows you how to:

- Select whether or not the card cage rear panel BNC is connected to the Trig1 or Trig2 or both signals of the emulation-bus analyzer.
- Select whether or not the coordinated measurement bus connection on the card cage rear panel is connected to the Trig1 or Trig2 or both signals of the emulation-bus analyzer.
- Select whether or not the emulator will allow a signal on Trig2 to initiate an emulation break.
- Select whether or not the emulation-bus analyzer will ignore the Trig2 line of the coordinated measurement bus.

Refer to Chapter 6, "Making Coordinated Measurements", in this manual for further details on the use of signals shown in the Interactive Measurement Specification dialog box.

To select whether the card cage rear panel BNC is connected to the Trig1 or Trig2 or both signals

- Choose "Drive", "Receive", "Neither", or "Both" for the "BNC on Trig1" or "BNC on Trig2" or both configuration items.

  Choosing "Drive" specifies that the signal connected to the BNC will be driven onto the Trig1 or Trig2 or both lines of the emulation-bus analyzer.

  Choosing "Receive" specifies that the Trig1 or Trig2 or both signals from the emulation-bus analyzer will be available to be received by any connection to the BNC.

  Choosing "Neither" specifies that the BNC is isolated from the Trig1 or Trig2 or both lines of the emulation-bus analyzer.

  Choosing "Both" specifies that the BNC can both "Drive" and "Receive" the Trig1 or Trig2 or both signals of the emulation-bus analyzer.

  Refer to Chapter 16, "Specifications and Characteristics", in this manual for the electrical specifications of the Trig1 and Trig2 trigger signals.

To select whether the coordinated measurement bus is connected to the Trig1 or Trig2 or both signals

- Choose "Drive", "Receive", "Neither", or "Both" for the "CMBT on Trig1" or "CMBT on Trig2" or both configuration items.

  Choosing "Drive" specifies that the signal connected to the coordinated measurement bus connection will be driven onto the Trig1 or Trig2 or both lines of the emulation-bus analyzer.

  Choosing "Receive" specifies that the Trig1 or Trig2 or both signals from the emulation-bus analyzer will be available to be received by any connection to the coordinated measurement bus.

  Choosing "Neither" specifies that the coordinated measurement bus is isolated from the Trig1 or Trig2 or both lines of the emulation-bus analyzer.

  Choosing "Both" specifies that the coordinated measurement bus can both "Drive" and "Receive" the Trig1 or Trig2 or both signals of the emulation-bus analyzer.

  Refer to Chapter 16, "Specifications and Characteristics", in this manual for the electrical specifications of the Trig1 and Trig2 trigger signals.

To select whether the emulator will allow a signal on Trig2 to initiate a break from target program execution

- Choose "Yes" or "No" for the "Enable Break Receive Trig2" specification.

  Choosing "Yes" specifies that a trigger signal supplied on the Trig2 line will be supplied to the emulator where it can be used to initiate an emulation break from execution of the target program and begin execution in the emulation monitor.

  Choosing "No" specifies that no emulation break will occur in response to any trigger signal on Trig2.

  Use this selection if you intend to cause an emulation break in response to an event found by a device connected to the card cage rear panel BNC. The emulator always receives Trig1 from the emulation-bus analyzer; Trig1 can be used to cause a break from execution of the target program to the monitor when the emulation-bus analyzer detects a trigger condition during a trace.

  Refer to Chapter 16, "Specifications and Characteristics", in this manual for the electrical specifications of the Trig1 and Trig2 trigger signals.

To select whether or not the emulation-bus analyzer will operate with, or ignore, the Trig2 line of the coordinated measurement bus.

- Choose "Drive", "Receive", or "Neither" for the "Analyzer on Trig2" specification.

  Choosing "Drive" specifies that the emulation-bus analyzer will drive a trigger signal on the Trig2 line of the coordinated measurement bus when it recognizes its trigger condition during a trace.

  Choosing "Receive" specifies that the emulation-bus analyzer will receive the signal on the Trig2 line of the coordinated measurement bus when it is supplied, and it will mark the state that it is capturing as the trigger state at the instant when the Trig2 signal is detected.

  Choosing "Neither" specifies that the emulation-bus analyzer will ignore the Trig2 line of the coordinated measurement bus.

  Refer to Chapter 16, "Specifications and Characteristics", in this manual for the electrical specifications of the Trig1 and Trig2 trigger signals.

Providing MMU Address Translation for the Foreground Monitor

When using the memory management unit (MMU) of the MC68040, the target system must provide the proper address translation for the foreground monitor. To be able to do this, you will need to understand your target system's physical memory map and MMU address translation structure. You may need to modify your mapping scheme or some of its mapping protections.

In order for the monitor to operate after the MMU is turned on, the target system must provide 1:1 address translation (logical address = physical address) for the block of memory occupied by the monitor. The foreground monitor will reside in a 4-Kbyte block of emulation memory corresponding to a single page in the MMU. This memory can be mapped to begin on any 4-Kbyte address boundary. Simply specify an address ending in 000h when you answer the monitor address question when you set up the emulation configuration.

For example, if the monitor is located at logical address 0ffff1000h, then the MMU must translate that address to physical address 0ffff1000h, logical address 0ffff1004h to physical address 0ffff1004h, etc.

Do not write-protect the address range occupied by the foreground monitor.

There are two ways to provide the proper address translation for the memory space occupied by the foreground monitor:

- Locate the foreground monitor in a block of memory that is transparently translated via ITTx and DTTx transparent translation registers (TTRs). The monitor contains both code and data so two TTRs are needed to provide translations: one for instructions, and the other for data. When the MMU processes translations, it first compares the logical address with the parameters of the TTRs. If it finds a match, the MMU uses the logical address as the physical address for the access (obtaining the needed 1:1 translation).

  The minimum block size that can be transparently translated by the TTRs is 16 Mbytes. If your target system already sets one or both data and instruction TTRs for supervisor, or both supervisor and user, access and no write-protection, then you may be able to find an unused 4-Kbyte block within this 16-Mbyte range where the monitor can reside.

If your target system does not use the TTRs, then you may want to modify your MMU boot code to configure an instruction and data TTR specifically for the monitor.

Example

This example shows how to modify boot code to use a pair of TTRs. Assume your target system does not access any physical addresses in the 16-Mbyte range 02000000..02ffffffh, and DTT0/ITT0 are unused. By locating the monitor at address 02000000 and adding the following code fragment to your boot code, you should be able to break into the monitor while the MMU is turned on:

```
* configure ITT0/DTT0 for emulation monitor
MOVE.L   #$0200C000,D0
MOVEC    D0,ITT0
MOVEC    D0,DTT0
```

Without these transparent translations for the monitor, the MMU will probably generate an access fault when you attempt to break into the monitor. The access fault would occur because addresses in the 02000000 range would have no valid translations (they would be on a non-resident page).

If you cannot modify your boot code, you may be able to use an execution breakpoint to break into the monitor before the MMU is enabled and use the monitor to configure the TTRs. Do this only as a last resort because the MC68040 processor automatically disables all TTRs whenever an emulation or target reset occurs (and they must be reinitialized each time).

- The second way to provide proper address translation for the foreground monitor is to locate the monitor within a page that is controlled by the MMU address translation tables; one that is always resident, writeable, supervisor accessible, and translated 1:1. The monitor occupies one 4-Kbyte page of emulation memory. It will be stored in the 4-Kbyte range of the dual-port memory.

Locating the Foreground Monitor using the MMU Address Translation Tables

Locate the foreground monitor at a specific page address and add the proper address translation for this page in your supervisor address translation tables. The minimum page size is 4 Kbytes so the monitor only requires a single translation. The page that contains the foreground monitor must always be resident, translated 1:1 (logical address = physical address), and never be write-protected.

The most direct way to do this is to modify the address translation tables in your source code, rebuild your executable file, and download the executable into RAM, or reprogram the executable into ROM. For systems that use an operating system to manage dynamic translation tables in RAM, the page allocated to the monitor must not be allowed to be swapped out by the operating system. This may require that the page selected for the monitor reside in unused space within the operating system (assuming the operating system is translated 1:1). The easiest way to create unused space is to globally define an 8-Kbyte array of data that is never referenced by your software. After rebuilding your operating system software, refer to the linker symbol map file to determine the address range of this array. Use the lowest address that resides on a 4-Kbyte boundary within this range as the starting address for the monitor.

As a last resort, if your target system software cannot be rebuilt, you can use the emulator to modify your translation tables directly.

The emulator provides a command to display individual address translations in detail, including address, value, and mnemonic information about each descriptor from the translation tables. You may be able to provide the proper address translation for the monitor by simply modifying a single descriptor (long word) to convert an invalid page into a resident page.

If the translation tables are located in ROM, you will need to copy them into emulation memory before you attempt to modify them. This is done by storing all or part of your ROM to a file, and then mapping emulation memory over the ROM address range and reloading the file.

9

Solving Problems

What to do when the emulator doesn’t behave as expected

Sometime during your use of the emulator, you'll encounter a problem that isn't adequately explained by an error message or obvious target system symptoms. This chapter explains how to solve some of these more complex problems.

Consider the following sources of information in addition to the specific problems discussed in this chapter:

- Look at the error log. Sometimes a problem will cause several error messages to be generated. Only the last error message will be shown on the status line. You can see the last 100 error messages by viewing the error log. Refer to Chapter 3, "Using the Emulator/Analyzer Interface", for details of how to display the error log.

- Look at the event log. Changes in status of the emulator/analyzer may cause unexpected results. To see a list of the last 100 events that affected the status of the emulator/analyzer, view the event log. Refer to Chapter 3, "Using the Emulator/Analyzer Interface", for details of how to display the event log.

- Look at the present status of the emulator/analyzer to see if it will suggest the cause of your problems. Refer to Chapter 3, "Using the Emulator/Analyzer Interface", for details of how to display emulator/analyzer status.

If the emulator appears to be malfunctioning

☐ Check to make sure that the cables connecting the Emulation Control Board to the Emulation Probe are connected correctly. Refer to Chapter 19, "Installation and Service", in this manual for details.

☐ Run the performance verification procedure as described in Chapter 19, "Installation and Service", of this manual. If the emulator fails this test, contact your Hewlett-Packard representative.

☐ If the emulator passes the performance verification procedure, look for other reasons for the problem. Performance Verification is a thorough test, but it cannot find every hardware failure in the emulator. It is a good indication that the emulator is functioning correctly, but if you are still convinced the emulator is malfunctioning, contact your local Hewlett-Packard representative.

If the trace listing opcode column contains only the words "dma long write (retry)" repeatedly

☐ Check to see if the internal ribbon cable that connects the last sixteen channels of the 80-channel internal analyzer to the HP 64783 emulator control board is missing. If it is, locate the supplied ribbon cable and connect one end to the slot in the analyzer board and the other end to the slot in the 68040 control board. Refer to Chapter 19, "Installation and Service", in this manual to see the proper location of this cable.

If the analyzer fails to trigger on a program address

☐ Check to make sure that the program address is a long-word address (an address ending in 0, 4, 8, or C hex). The MC68040 fetches instructions on long-word addresses. Other instruction addresses never appear on the processor bus, and therefore are never seen by the analyzer. Modify the trigger address so that the two least significant binary digits of your trigger address are zeroes. For example, to trigger a trace on address 2316H, specify your trigger to occur on address 2314H. Note that this only applies to instruction fetches; data reads and writes are made directly to the destination address, regardless of whether it is a long-word address or not.

If triggering on the occurrence of a program symbol, the above can be achieved by including the **long_aligned** token in your trigger command.

If the analyzer triggers on a program address when it should not

- ☐ Check to see if the analyzer is triggering on an instruction prefetch. The analyzer cannot distinguish between prefetch and execution because the processor does not provide that information. Usually your actual trigger address is within 16 words of the address where trigger is occurring.

- ☐ Try to pad the program code with NOP instructions to move the trigger address away from the other code so that it won't be prefetched until it is time to trigger.

- ☐ You may be able to insert a write instruction to a meaningless variable in your code immediately before the trigger address. Then you can trigger on a write to the address of the meaningless variable. Write transactions never appear in instruction prefetches.

If trace disassembly appears to be partially incorrect

- ☐ Check to see if the analyzer began disassembly of the trace on a long-word boundary but the instruction started on the low word within the long word. This will make disassembly incorrect. You can start disassembly on the low word within the long word by use of **display trace disassemble_from_line_number <trace list line number> low_word**.

- ☐ If the trace list seems correct for a few states after disassembly starts, and then it seems incorrect, restart disassembly of the trace at the low word where disassembly first becomes incorrect **display trace disassemble_from_line_number <trace list line number> low_word**.

- ☐ If an instruction seems to have incorrect data associated with it, you can read down the trace list to see if you can find correct data for the instruction on another line. You can cause the disassembler to realign the instruction with the correct data by

entering a command like **display trace disassemble_from_line_number <trace list line number> align_data_from_line <trace list line containing data>.**

If there are unexplained states in the trace list

- ☐ Check that the sequence, storage and trigger specifications are set up to exclude the states you don't need.
- ☐ Try using the **disassemble_from_line_number <LINE#> align_data_from_line <STATE#>** option to the **display trace** command to inform the dequeuer which operand state belongs with the first instruction state.
- ☐ Try using **display trace dequeueing on**.
- ☐ Try using the **disassemble_from_line_number <LINE#> low_word** option to the **display trace** command to begin disassembly from the low word of the starting state, instead of the high word.
- ☐ Check to see if instruction or operand accesses in the range covered by the trace could be filled from cache memory. If so, these cycles won't appear in the trace list, which will confuse the disassembler. Either disable the cache memory entirely or disable caching for those address ranges by adding the **tci** (transfer cache inhibit) attribute to those ranges in the memory map. (See Chapter 8, "Configuring the Emulator.")

If you see negative time or negative states in the trace list

☐ If counter overflow occurs during a deep analyzer trace measurement, you may see a count of negative time or negative states in the trace list. This is a normal condition. It indicates that the counter value stored with the reference state was greater than the counter value stored with the present state. In absolute time counts, negative times will continue to be seen until a state is captured whose counter value is greater than the trigger state counter value. In relative time counts, negative time should only be seen beside the first state captured after the counter overflows.

If the analyzer won't trigger

☐ Instruction fetches from cache memory aren't visible to the analyzer. You can disable the cache while using the analyzer by answering **no** to the configuration question "Enable the 68040 instruction and data cache?" (Use the **modify configuration** command to access this configuration question.) Reenable the cache to improve performance when you're finished using the analyzer.

☐ The analyzer can be configured to trace background monitor execution, foreground monitor and target program execution, or both background and foreground operations. If you trace only background monitor execution, the analyzer will not see any foreground cycles and will not trigger the trace. (Use the **modify configuration** command to access this configuration question.)

☐ The MC68040 emulator only fetches instructions on long word boundaries (least significant hex digit of address is 0, 4, 8, or C). However, program labels can be aligned on word boundaries between long word boundaries. If you try to trace on a label located on a non-long-word boundary, the emulation-bus analyzer will never trigger because the address will never appear on the address bus. To mask an address so that it is on a long word boundary, use the **long_aligned** keyword with trace specifications.

If the emulator won't work in a target system

- ☐ Ensure that the probe is inserted into the target system in the correct manner—the pins should be lined up correctly.
- ☐ If you are using spacers to connect the emulator probe to the target system, make sure that the spacers are correctly connected.
- ☐ The emulator uses the clock from the target system. Unsupported or improperly configured clock speeds will affect emulator performance. (Use the **modify configuration** command to access this configuration question.)
- ☐ The emulator must recognize the target system's clock signal to function correctly. Power up the emulator and then apply power to the target system.
- ☐ Check to see that the signal timing specifications of your target system match the specifications in Chapter 16, "Specifications and Characteristics."

If you see multiple guarded memory accesses

- ☐ Check the stack pointer value. If it points to guarded memory, you will see multiple guarded memory accesses each time you press the <Enter> key (to get a new prompt). Reset the emulator and set the stack pointer to a correct value.

If you suspect that the emulator is broken

1. Shut off power to the target system first, and then the Card Cage.
2. Disconnect the emulator from your target system.
3. Connect the emulator to the demo board. Also connect the power cable from the emulator to the demo board and connect the reset flying lead. (See Chapter 19, "Installation and Service".)
4. Apply power to the Card Cage.
5. Run performance verification by entering the commands:

```
display pod_command
pod_command "pv 1"
```

Note that the emulator/analyzer interface will report an I/O error because the **pv** command initializes the emulator. You will need to enter the **end release_system** command to exit the emulator/analyzer interface.

If either the emulator or analyzer fail the performance verification, check the installation of those modules. See Chapter 19, "Installation and Service", for information on installation and an explanation of the performance verification software. If the installation is correct, contact your local HP Sales Office for assistance.

If you have trouble mapping memory

☐ The emulator uses a best fit algorithm to assign memory blocks to map requests. Because the memory block sizes available depend on the emulation memory module installations and the use of the dual-port memory, it's possible that a 256-byte map request may use 512 Kbytes. (The map term will be only 256 bytes.) Most systems won't have such differences between memory block size requirements and available memory. However, certain emulation memory module installations will aggravate the problem.

☐ Also, use of the dual-port memory is controlled first by monitor selection and next by explicit selection of a dual-port term in the map. If you choose a foreground monitor, the dual-port memory block is reserved for the monitor. If you choose a background monitor, and don't explicitly map a term with the **dp** attribute, the dual-port memory may be used to satisfy any map request. For example, if you request a 256-byte map term and this memory block is available, it will be used to satisfy the request because it is closest to the needed size. Or, if you request a term that is slightly larger than another available block, the dual-port memory will be used with another map term to satisfy the request. (For example, a 260-Kbyte request may use one 256-Kbyte block and the 4-Kbyte dual-port memory.)

Refer to the section "Mapping Memory" in Chapter 8, "Configuring the Emulator", for more information on memory allocation.

If emulation memory behavior is erratic

☐ Check to see if you have installed HP 64171A or HP 64171B memory modules on the emulation probe board. These memory modules are too slow to work with the MC68040 emulator. Use HP 64172A, HP 64172B, or HP 64173A memory modules.

If you're having problems with DMA

☐ Check to make sure that your DMA process doesn't access memory ranges mapped to **emulation ram** or **emulation rom**. DMA to emulation memory is not supported.

If you're having problems with emulation reset

☐ Connect the reset flying lead to some point in your target system that distributes the reset signal to components that need to be reset when the processor is reset. This will make sure that all critical components in your target system are reset by the emulator. Suppose your system reset circuit drives several critical system components as well as the processor. Suppose also that the critical components are memory-mapping circuits that locate ROM containing the vector table at address zero for startup, then move it to a high address range after system initialization. An emulator reset cannot drive your reset line directly. Therefore, an attempt to run after emulation reset will fail because the vector table is not located in the correct place. Use the target reset to reinitialize memory or use a **run** command instead of a **run from reset** command. For further information, refer to Chapter 18, "Connecting the Emulator to a Target System".

If the deMMUer runs out of resources during the loading process

- ☐ Check the physical address ranges that will be reverse translated by the present setup of the deMMUer. Enter **load demmuer verbose** to see a list of those physical address ranges. If all of the physical spaces where you have code under development are listed, ignore the "out of resources" message.

- ☐ Check to ensure that you have placed sufficient restrictions in the MMU mapping paths to prevent reverse translating physical address space where you have no memory.

- ☐ Check your emulation memory map to make sure you have entries to support each of the address spaces where you have code under development. Make sure those spaces are no larger than they need to be to accommodate your program code.

- ☐ Check if you are using both root pointers in your memory mapping scheme. The deMMUer may have run out of resources for only one of the root pointers.

- ☐ Read "Using the deMMUer" in Chapter 10, "Using Memory Management", for ways to make more efficient use of deMMUer resources.

If verbose mode shows less than eight mappings but the deMMUer is "out of resources"

- ☐ Check if you are using both root pointers in your memory mapping scheme? The deMMUer may have run out of resources for only one of the root pointers.
- ☐ Read "Using the deMMUer" in Chapter 10, "Using Memory Management", to understand how deMMUer resources are allocated when using both root pointers and separate mappings.

If you only see physical memory addresses in the analyzer measurement results

- ☐ Check to see if you enabled the deMMUer with the command: **set demmuer on**.
- ☐ Check to see if you loaded the deMMUer with the information needed to reverse translations made by the MMU with the command: **load demmuer verbose**.
- ☐ Read "Using the deMMUer" in Chapter 10, "Using Memory Management", to understand how the deMMUer selects physical address ranges to reverse translate for the analyzer.

If the deMMUer is loaded but you still get physical addresses for some of your address space

- ☐ Some physical accesses are normal, especially accesses to the MMU tables.

- ☐ Check to see which physical memory spaces are being reverse translated by the deMMUer. Enter the **load demmuer verbose** command to see a list of the physical address spaces that will be deMMUed.

- ☐ Check the setup of the MMU mapping tables. Make sure unused address spaces are marked with invalid descriptors in the mapping tables.

- ☐ Check the emulation memory map. Make sure you have allocated only the memory spaces needed to accommodate code you are developing in your map. Make sure you have mapped the smallest spaces that you can for the code you are developing.

- ☐ Check that the MMU had the setup you wanted to analyze when you loaded the deMMUer. If it was managing memory for some other MMU setup, break to the monitor and issue the **load demmuer** command again.

- ☐ Check to see if there was a context change in the MMU during execution of your program. If there was, the content of the root pointer may have changed for execution of the new context. The deMMUer tables were set up to reverse translate the MMU tables under the root pointer values that existed when you entered the **load demuer** command. If those root pointer values change (pointing to other translation tables), there is no way to automatically update the deMMUer. It will continue to provide reverse translations for the setup that existed at the time you issued the **load demmuer** command. Issue the **load demmuer** command again.

  Read "Using the deMMUer" in Chapter 10, "Using Memory Management", to understand how the deMMUer selects the physical addresses it will translate.

If you can’t break into the monitor after you enable the MMU

☐ Enter the commands: **reset**, and then **break**. If your MC68040 is now running in the monitor, look at your MMU tables or the transparent translation register that maintains 1:1 mapping for your foreground monitor. The mapping has failed. Modify your MMU tables or the transparent translation register to obtain the 1:1 mapping for the address space occupied by the foreground monitor.

☐ Refer to the end of Chapter 10, "Using Memory Management", for a detailed example that discusses how to solve a "can’t break into monitor" problem.

If the target system exhibits unexpected behavior after executing a breakpoint

☐ Normally when a software breakpoint instruction (BKPT) is executed, the emulation processor generates a breakpoint acknowledge cycle as part of the instruction. The emulator terminates the breakpoint acknowledge cycle and initiates a transition into the monitor. Some target systems cannot tolerate this behavior; they exhibit unexpected activity. When using such a target system, configure the emulator to wait for the target system to terminate breakpoint acknowledge cycles.

To configure the emulator to wait for the breakpoint acknowledge cycle to be terminated by the target system, enter the command **Settings→Pod Command Keyboard**. On the command line, enter the terminal interface command **cf bplock=en**. Then press the **suspend** softkey. With this selection, the target system is responsible for terminating breakpoint acknowledge cycles by asserting **TA** or **TEA**. If the target system fails to provide the required cycle termination signal, the processor will remain in a wait state indefinitely. Make sure your target system provides the required cycle termination signal.

Emulators with serial prefix numbers below 3343A may not support this feature even though they may appear to accept the **bplock** configuration item.

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 64783 product.

Chapter 10, "Using Memory Management", shows how to use the resources of the MC68040 emulator when developing virtual memory systems.

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

Chapter 12, "Emulator Messages," lists each of the messages that you may see while using the MC68040 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.

Chapter 13, "Setting X Resources," shows how you can change the appearance or behavior of certain elements in the graphical interface.

Chapter 14, "The SPARCsystem™ Graphical User Interface and Softkey Interface," discusses use of the emulator/analyzer interface on the SPARCsystem.

Chapter 15, "Microtec Language Tools Used With MC68040 Emulators," describes how to use the emulator/analyzer with Microtec language tools.

Chapter 16, "Specifications and Characteristics," lists the specifications and operating characteristics of the MC68040 emulator/analyzer.

10

Using Memory Management

Understanding logical and physical emulation and analysis

Understanding Emulation and Analysis Of The Memory Management Unit

You only need to read this chapter if you are using the on-chip MMU (Memory Management Unit) of the MC68040 or MC68LC040 microprocessor. If you are using an MC68EC040, or if you are using an MC68040 or MC68LC040 with its MMU disabled, you won't need the information in this chapter.

This chapter begins with a discussion of terms and conditions you need to understand when you are using the MC68040 or MC68LC040 emulator/analyzer with the MMU enabled. Under these conditions, many capabilities and features become available that are not otherwise offered. Also, some of the features you have been using behave differently. These are discussed in this chapter.

Terms And Conditions You Need To Understand

The following paragraphs explain the differences between logical and physical memory, and between static and dynamic virtual memory systems.

Logical vs Physical

When you develop a program, compile it or assemble it, and link it, addresses are assigned to contain each of the bytes of the program. These addresses are logical addresses. When the program is loaded into hardware memory so that it can be executed by the microprocessor, it is loaded into physical address space. When you are not using an MMU, the program is loaded into physical memory hardware at the logical addresses assigned in the linker load map. Under these conditions, there is no need to differentiate between logical addresses and physical addresses because they are the same (simply addresses). When you use the MMU, it becomes necessary to understand the difference between logical addresses and physical addresses.

Most emulation and analysis commands that require an address as part of the command use logical addresses. Some emulation and analysis commands will accept either logical or physical addresses.

What are logical addresses?

Logical addresses are the addresses that are assigned to your program code when you develop your program. They are the addresses represented by symbols in your symbols data base (the symbol "Main" represents a logical address).

What are physical addresses?

Physical addresses are the addresses assigned by the MMU to contain your program. Physical addresses identify locations where you actually have memory hardware in your target system. Physical addresses appear on the processor address bus instead of logical addresses.

Static and dynamic system architectures

There are several design strategies where memory management can help in developing a system or product. Three of these are described in the following paragraphs. One shows memory management used in a static memory system. The other two show memory management used in different dynamic memory systems. The MC68040 emulator is designed to work in any of these system types; however, the deMMUer which provides reverse translations to the analyzer is primarily intended for use in static systems.

Static system example

A static system design may use the MMU simply to protect supervisor code and I/O space against accesses from a user program. Once a static system is initialized, it never changes. Your HP emulator and analyzer can give you complete support for a static memory management system. After the MMU has been set up to manage memory in a static system, the deMMUer can be loaded with information to reverse the MMU translations over the entire range managed by the MMU.

Non-paged dynamic system example

Assume three programmers are developing separate programs to run in a real-time operating system environment. The programmers each write their programs to begin at address 0h. The operating system accepts the responsibility to know where

in physical memory space each of these programs will be located. The programmers don't have to worry that some additional code they write in their programs might overwrite some of the code that was written by another programmer. The operating system will place all of the code in available memory space and place appropriate translation mappings in the MMU to ensure that when the logical address for one of the programs (tasks) is present in the program counter, the appropriate physical address will appear on the bus to access the desired physical memory location.

Your HP emulator/analyzer can give you partial support for a non-paged, dynamic system. When the MMU has been set up to manage memory during execution of one of the above tasks, you can update the deMMUer to translate addresses for that task. When that task is executing, the analyzer will be able to make trace measurements and provide correct results. When any of the other tasks are executing, trace measurement results will be invalid because the other tasks will depend on different translation tables in the MMU and there is no way to automatically update the deMMUer when execution switches from one task to another.

Paged dynamic system example

Assume you have developed a program that occupies 10 megabytes of logical address space. Perhaps you have only 2 megabytes of physical address space in your system. Still, you want to be able to run the entire program. You set up a specification in the MMU translation control register to divide the address space into pages (the 68040 lets you divide the memory space into one of two page sizes: either 4 Kbytes or 8 Kbytes). Assume you set up the MMU to divide the memory into 4-Kbyte pages. Your program will occupy 2,500 pages of code, and 500 of these pages can be contained within your physical memory space at any given time.

As your program executes, the operating system moves pages of your program code into address space in physical memory. When execution goes beyond the addresses contained on the presently active page, the MMU checks to see if the next logical address is on a page that has already been placed in physical memory. If it is, the MMU performs the appropriate translation for the next logical address, placing the appropriate physical address on the bus, and execution continues. If it is not, the operating system moves the page that has the next address to be executed up from an external storage device to physical memory space, overwriting one of the pages that had occupied physical space before. The operating system updates the translation tables to identify the new logical address space that now occupies that 4 Kbyte of physical memory, and program execution continues.

As pages are swapped back and forth between an external storage device and the physical memory, the relationship between any one logical address and its corresponding physical address may change many times.

Your HP emulator will let you run a paged, dynamic system, but the analyzer will not be able to provide support for such features as symbolic addresses, or display of corresponding source files. The deMMUer cannot detect changes in the MMU mappings. The longer the system runs, the further out of date the deMMUer will become. Of course, the analyzer will still be able to show activity captured at physical addresses. By experimenting with several starting points for the inverse assembler, you can obtain a trace list with activity inverse assembled into an equivalent assembly language listing (**display trace disassemble_from_line_number <NO.> [low_word]**).

Where Is The MMU?

The MMU is located between the CPU core and the external address bus. The program counter always contains logical address values. When the MMU is turned off, the program counter value is placed directly on the address bus to access an address in physical memory. When the MMU is turned on, the MMU accepts the logical address value and translates it (by using its translation tables) to a physical address. The physical address from the MMU is placed on the processor address bus.

Using Supervisor and User Privilege Modes

The MMU allows separate tables to be set up for supervisor and user access. For example, you can create one set of mapping tables to translate addresses in supervisor space and another set of mapping tables to translate addresses in user space. The supervisor space uses the SRP (supervisor root pointer). The user space uses the URP (user root pointer). The supervisor address space can begin at supervisor address 0, and the user address space can begin at user address 0. The MMU must ensure that these addresses are placed in different physical spaces.

You can use the MMU to protect your program space from unauthorized accesses. If you map a portion of your program through the MMU and identify it as supervisor space, the MMU will not allow any access to that program space unless the privilege mode is supervisor at the time the access is attempted. Take care to ensure that supervisor or user is specified with addresses if the MMU will be making the distinction (example: **<address> supervisor emulation rom**).

How the MMU is enabled

The MMU depends on a hardware enable and a software enable. Both of these enables must agree to enable the MMU before it can translate logical addresses to physical addresses. If either one (or both) of these enables fail to enable the MMU, it will remain disabled.

Hardware enable

The hardware enable is performed by the $\overline{\text{MDIS}}$ signal. When $\overline{\text{MDIS}}$ is asserted, the MMU is disabled. When $\overline{\text{MDIS}}$ is negated, the MMU is enabled to translate addresses. The emulator controls the $\overline{\text{MDIS}}$ line according to the way you set the "Enable the MMU?" configuration parameter.

If you enter **no**, the $\overline{\text{MDIS}}$ line is held asserted. If you enter **yes**, the $\overline{\text{MDIS}}$ line is directly controlled by the target system. In this condition, your target system can hold the line high or low to enable or disable the MMU.

Software enable

The software enable is performed when the operating system loads the enable value into the translation control register (TC). If the enable bit of the TC register is "e=1", the MMU will be enabled. If the enable bit in the TC register is "e=0", the MMU will be disabled.

Restrictions when using the emulator with the MMU turned on

There are only three restrictions: you must use a foreground monitor, it must not be write-protected, and you must map it to address space that the MMU translates 1:1 (logical=physical) for supervisor accesses.

You must use a foreground monitor. The background monitor does not have the capabilities to support the MMU functions. The foreground monitor can operate with the MMU turned on.

You must map the monitor code to address space that the MMU translates 1:1 for supervisor accesses. The emulator executes monitor code to implement many of its emulation features. The emulator must be able to find the monitor code whether the MMU is turned on or off. By mapping the monitor into address space that has a 1:1 translation, the monitor stays within known address space at all times, and the emulator can always find it when it needs to use it. This mapping is described at the end of Chapter 8, "Configuring the Emulator."

Be sure that no write-protection exists in the MMU mapping for the monitor.

Caution

Make sure your translation tables are valid. Turning on the MMU can cause your program or emulator to fail if the MMU tables are not set up correctly. The address space where the program is executing can change when the MMU is turned on or turned off. Stack space or other data spaces can move. Breakpoints that have been set can be lost.

How the MMU affects the way you compose your emulation commands

When you display registers, the address registers, stack pointers, and program counter always contain logical addresses, even when the MMU is turned on.

If you place an address in the entry buffer and choose **Execution→Run→from()**, or enter a **run from <address>** command, the address you enter must be a logical address. The program counter will accept it and supply it to the MMU for translation before it places the address on the processor bus.

Breakpoint addresses in RAM space are always logical addresses. When you set a breakpoint at an address, that address is translated by the MMU and the BKPT instruction replaces the instruction at the appropriate physical address. When the breakpoint is executed, the emulator restores the original instruction to the physical address, by first translating the logical address through the MMU.

Consider what happens if you set a breakpoint at a particular address, and before the breakpoint is hit, you update the translation tables in the MMU, changing the mapping to the location where the breakpoint is set. This is discussed in detail under "Solving Problems" at the end of this chapter.

If you enter a command to display memory or modify memory, your command is directed to logical address space. If you want to display memory at a physical address, you have to change your command. For example, the command to display memory at address 100H (**Display→Memory→Hex()**, or **display memory 100h**) will show you the memory content at logical address 100H (which might be some other physical address). If you want to see the content at physical memory address 100H, you will have to enter the command **display memory physical 100h**.

Addresses expressed using symbols are always logical addresses. In the case of symbols, the emulator looks in the symbol data base and finds the logical address that corresponds to the symbol you used in your command, and it loads that logical address into the program counter.

If you attempt to modify a memory location that is write-protected by the MMU, the emulator will temporarily modify the translation tables to allow access.

Seeing Details of the MMU Translations

The following paragraphs discuss emulator displays that help you understand translations made by your MMU. There are three displays, each giving a different level of detail of the MMU translations.

- The present address mappings in your MMU tables.
- The translation table entries for a single logical address.
- The contents of a single level of the translation tables pointed to by a selected logical address.

How the emulator helps you see the details of the MMU mappings

To see all of the logical-to-physical translations presently mapped, choose **Display→MMU Translations** from the pulldown menu, or enter the command **display mmu_translations**. The emulator will read the present state of the translation tables and show all of the valid mappings in those tables. The display will be similar to the following:

```
Logical Address          Physical Address          Attributes
000089000..000089fff@s   0fff89000..0fff89fff@sa   S W
00008a000..00008afff@s   0fff8a000..0fff8afff@sa   S W
00008b000..00008bfff@s   0fff8b000..0fff8bfff@sa   S W
00008c000..00008cfff@s   0fff8c000..0fff8cfff@sa   S W
00008d000..00008dfff@s   0fff8d000..0fff8dfff@sa   S W
00008e000..00008efff@s   0fff8e000..0fff8efff@sa   S W
00008f000..00008ffff@s   0fff8f000..0fff8ffff@sa   S W
000090000..000090fff@s   0fff90000..0fff90fff@sa   S W
000091000..000091fff@s   0fff91000..0fff91fff@sa   S W
000092000..000092fff@s   0fff92000..0fff92fff@sa   S W
000093000..000093fff@s   0fff93000..0fff93fff@sa   S W
000094000..000094fff@s   0fff94000..0fff94fff@sa   S W
000095000..000095fff@s   0fff95000..0fff95fff@sa   S W
```

The above listing shows privilege modes were included in the mapping scheme. The logical and physical addresses were shown in supervisor space. Notice that the physical addresses also show "a" beside the privilege mode indication. The "a" indicates physical address space.

Note that the emulator enters the monitor to obtain the information it shows in the MMU displays. Execution of your target program is suspended while the emulator gathers information for an MMU display. If there are portions of your target program that should not be interrupted during execution, insert an execution breakpoint in some safe area of your program code and run until the breakpoint is executed. Then you can safely view the MMU mappings.

The display you get with the **Display→MMU Translations** or **display mmu_translations** command can show as much as one line per page (or group of adjacent pages) of mapped logical address space. Contiguous entries are shown on one line to make the display more readable. Early terminations (which result in contiguous translation of multiple pages) will also be shown on a single display line.

The display of MMU mappings will only show pages for which the system has valid mappings. No information is given in the default **mmu_translations** display for paths designated invalid, or for paths containing illegal entries.

To avoid a list of mappings that scrolls for a long time, include an address or address range in your command. By choosing **Display→MMU Translations...** and entering a limited address range in the dialog box, or using the command **display mmu_translations 0 thru 0ffffh**, for example, the emulator will show the valid mappings for only the logical addresses in the range you specify, instead of all possible mappings.

Another way to limit the number of address ranges shown in an MMU mappings display is limit the listing to only user or supervisor address space. By choosing **Display→MMU Translations...**, entering Start Address 0 and End Address 0ffffh, and clicking on Function Code user in the dialog box, or using the command **display mmu_translations fcode user 0 thru 0ffffh**, the display will show only the mappings for addresses 0 through 0ffff in user address space.

Note: For convenience, **display mmu_translations** will use the logical address range from the most recent **display mmu_translations <ADDRESS> thru <ADDRESS>** command, if possible. To change the default logical address range back to the full address space, use the command: **display mmu_translations 0 thru 0ffffffffh**, or obtain the **Display→MMU Translations...** dialog box and enter the desired address.

The display shows TT beside address ranges that are overridden by the transparent translation registers. In these ranges, logical-to-physical address translation will be 1:1. The MC68040 always compares logical addresses to the content of the transparent translation registers before it attempts a translation. If it finds a match in the transparent translation registers, it accepts the logical address as the physical address and performs no translation.

Supervisor/user address mappings

If you are using separate supervisor and user mappings, the emulator will support this choice and show appropriate information.

- To see only the mappings in supervisor address space, choose **Display→MMU Translations...** and in the dialog box obtain Function Code super and enter the desired address range, or use the command: **display mmu_translations fcode super [<address>[thru <address>]]**. This tells the emulator to show the supervisor mapping for the associated logical address or address range.
- To see only the mappings under the URP, choose **Display→MMU Translations...** and in the dialog box obtain Function Code user and enter the desired address range, or use the command: **display mmu_translations fcode user [<address>[thru <address>]]**.
- If you do not enter a specification for address space, mappings will be shown for both root pointers.

The MC68040 uses the URP as the root pointer for user address space, and the SRP as the root pointer for supervisor address space. No distinction is made between program and data space.

Translation details for a single logical address

To see translation details for a logical address, choose **Display→MMU Translations...** and click on MMU Tables in the dialog box; then type in an address, such as 40f8h, in the field beside Address. Or in the command line, enter a command such as: **display mmu_translations tables <address>.** The **tables** option tells the emulator to show the translation details for the specified address. The display will show the way the logical address is mapped through the tables to reach its corresponding physical address.

```
Logical Address (hex)     0    0    0    0    4    0    F    8
Logical Address (bin)  0000 0000 0000 0000 0100 0000 1111 1000
Table Level            AAAA AAAB BBBB BBCC CCCC PPPP PPPP PPPP

LEVEL INDEX LOCATION   CONTENTS   TBL/PAGE G Ux S CM M U W UDT/PDT
SRP                    02028200   02028200                 RESIDENT
A      000  02028200   ffffffff   fffffe00             y y RESIDENT
B      000  fffffe00   ffffffff   ffffff00             y y RESIDENT
C      004  ffffff10   ffffffff   fffff000 y 11 y in y y y RESIDENT

Physical Address (hex) = fffff0f8
```

Address mapping details

The example display shows:

- The translation mapping for logical address 40f8H in supervisor space. Both the hexadecimal and binary values are shown for the logical address.
- The Table Level line shows how each address bit is mapped. The first seven bits are used as an offset into Table A. The next seven bits offset into Table B. The next six bits offset into Table C. The example display was made with 4-Kbyte pages selected; only five bits index into Table C when 8-Kbyte pages are selected. The lowest-order 12 bits contain the offset into the physical page.
- The index used in Table A is 0 which points to physical address 2028200. The content of this address is ffffffff, indicating a B level table located at base address fffffe00. The status also indicates that this table has been used "U", and the address is write-protected.
- The physical address is finally calculated by adding the physical page offset to the base address of the physical page.

Status information

The status "U" and "W" can be assigned to an address at any point in its mapping. You must OR these two status values at each level of the mapping. For example: if the "W" bit shows "y" at the level of table A, then all of the pages under this entry in table A are write-protected.

Note that the address shown in the example display was mapped through the supervisor root pointer. If you wanted to see the mapping through the tables under the user root pointer, you could choose **Display→MMU Translations...** and click on MMU Tables in the dialog box; then type in an address, such as 40f8h, in the field beside Address and select User beside the Function Code option. Or in the command line, you would use a command like **display mmu_translations tables fcode user 40f8h**. You can add the desired function code table index to your command to see how any address is mapped through the tables under the selected root pointer (e.g. **user** or **super**).

The specific status bits shown beside each table entry are defined as follows:

- TBL/PAGE indicates the base address of the next table.
- G means the entry is global.
- Ux shows the values of the user programmable attributes (signals UPA0 and UPA1).
- S means supervisor mode protection.
- CM identifies the cache mode: cw (cachable, writethrough), cc (cachable, copyback), is (inhibited, serialized), or in (inhibited, nonserialized).
- M means the page has been modified.
- U means the page (or pages) has been used, or previously accessed.
- W means the page is (or pages are) write-protected.
- UDT/PDT indicates whether the page at the next level is RESIDENT or INVALID.

Table details for a selected logical address

The lowest level of detail you might like to see is the content of one of the tables used to map a particular logical address. You might choose **Display→MMU Translations...** and in the dialog box, click on MMU Tables, enter Address 40f8h, and click on Table Level to obtain C; or enter a command like: **display mmu_translations tables 40F8h level C**. The emulator would show the details of Table C where it is used to map logical address 40F8. There might be a great many Table C's, but this command will only show the Table C that is used to map the logical address you specified in your command.

In the example display of table details:

- The LOCATION column shows the physical address of each indexed location in Table C.
- The TBL/PAGE column shows the base addresses of physical pages indicated by each location in Table C.
- The first indexed location in Table C shows that its associated physical page has been accessed, but not modified ("U" bit = "y", and "M" bit = "n").

```
Logical Address (hex)     0    0    0    0    4    0    F    8
Logical Address (bin)  0000 0000 0000 0000 0100 0000 1111 1000
Table Level            AAAA AAAB BBBB BBCC CCCC PPPP PPPP PPPP

LEVEL INDEX LOCATION   CONTENTS   TBL/PAGE G Ux S CM M U W UDT/PDT
SRP                    00000200   00000200                 RESIDENT
A      000  00000200   0000040b   00000400             y n RESIDENT
B      000  00000400   0000060b   00000600             y n RESIDENT
C      000  00000600   0000008f   00000000 n 00 y cw n y y RESIDENT
C      001  00000604   00001087   00001000 n 00 y cw n n y RESIDENT
C      002  00000608   00002087   00002000 n 00 y cw n n y RESIDENT
C      003  0000060c   00003087   00003000 n 00 y cw n n y RESIDENT
C      004  00000610   00004087   00004000 n 00 y cw n n y RESIDENT
C      005  00000614   00005087   00005000 n 00 y cw n n y RESIDENT
C      006  00000618   00006087   00006000 n 00 y cw n n y RESIDENT
```

Using the DeMMUer

The deMMUer circuitry reverses the translations made by the MMU (translates the physical addresses it finds on the processor buses back to their corresponding logical addresses) before sending the addresses to the analyzer.

What part of the emulator needs a deMMUer?

Actually, the emulator doesn't need the deMMUer; the analyzer does. It can't provide its full symbolic features unless it has help from the deMMUer. The analyzer normally receives its address information directly from the processor address bus. It uses the symbols data base created for the program loaded in memory to cross reference the addresses it receives to the symbols and corresponding code in your source files. When the MMU is used, logical addresses are translated to physical addresses before they are placed on the processor address bus. Therefore, they no longer match the symbols data base.

What would happen if the analyzer didn't get help from the deMMUer?

The analyzer would get its address information directly from the address bus of the emulation processor. It would have no way to know what translation had occurred in the MMU. Therefore, it could not trigger or qualify its trace on any symbol defined in the symbols data base. Further, its trace list could only show you the physical address value it found on the address bus; it would not be able to show any symbols associated with that physical address, or any corresponding source file lines. You would have to figure out for yourself what portion of your program address space was executing when that physical address appeared on the bus.

How does the deMMUer serve the analyzer?

The analyer does not get its information directly from the processor address bus when the deMMUer is turned on. Instead, the deMMUer accepts the physical address from the processor address bus, reverse-translates it to its logical address value, and supplies it to the analyzer. By having the logical address corresponding to the transactions on the processor address bus, the analyzer can accept trace

specifications expressed in source file symbols, show symbols in its trace lists, and show the regions of the source files that were executing when the bus activity occurred.

Reverse translations are made in real time

The deMMUer performs its reverse translations without slowing down the measurement. For this reason, the analyzer that obtains its information from the deMMUer is able to provide its full feature set.

DeMMUer options

- **Settings→DeMMUer→Enable** or **set demmuer off**. Your analyzer receives physical addresses if the MMU is enabled. The analyzer can only show hexadecimal values for those physical addresses. They may not correspond to the logical addresses of your program code. Note that until the MMU is enabled in hardware and software, addresses will be logical. Only after the MMU is enabled is there a distinction between physical and logical.
- **Settings→DeMMUer→Enable** or **set demmuer on**. Your analyzer receives logical addresses translated by the deMMUer according to the tables in place when you last loaded the deMMUer. The DeMMUer must be loaded with data to reverse translations before it can be enabled.
- **Settings→DeMMUer→Load from Memory** or **load demmuer**. The emulator reads the MMU registers and interprets the translation tables to load the deMMUer. The deMMUer is also enabled after it is loaded.
- **Settings→DeMMUer→Load from Memory...** This opens a dialog box in which you can specify values to override the present values in MMU registers when loading the DeMMUer.

- **Settings→DeMMUer→Load from File...** This opens a dialog box in which you can specify the name of a file of reverse translations. This file will be loaded directly into the DeMMUer instead of reading the present translation tables and MMU register values to create the reverse translations. Note that this file must have been previously created using the **File→Store→DeMMUer (From MMU Tables)** command from the pulldown menu. This file will have a ".ED" filename extension.

- **Settings→DeMMUer→Verbose**. This sets verbose mode for the deMMUer load function. In the pulldown menus, this is simply a switch. When "Verbose" is selected for loading the DeMMUer, a list is displayed of the physical address ranges that will be reverse-translated by the deMMUer. Addresses will be shown as <address>..<address>@s
This means <address> through <address> at supervisor address space.

What the emulator does when it loads the deMMUer

When the emulator loads the deMMUer from memory, it does the following:

- Temporarily breaks into the monitor.
- Reads the MMU registers and translation tables from memory to determine all logical-to-physical address translations.
- Loads the reverse translations into the deMMUer hardware.
- If the accessible physical memory exceeds the 256-Mbyte limitation of the deMMUer, the emulator reads the emulation memory map for help in selecting appropriate address ranges to reverse translate.

When the emulator loads the deMMUer from a file, the only difference in the above algorithm is that the address translations are obtained from the file instead of by reading MMU registers and translation tables from memory.

Restrictions when using the deMMUer

Keep the deMMUer up to date

When you load the deMMUer, the emulator reads the present value of the TC, SRP, and URP registers in the MMU, and the present translation tables, and calculates the address translations that can be performed (all possible physical-to-logical translations are determined during this process). Then the emulator loads the deMMUer to reverse those translations. After the deMMUer is loaded, any change to the MMU, its registers, or its translation tables will make the deMMUer out of date. The only way to update the deMMUer for changes in the translation setup is to load the deMMUer again.

The target program is interrupted while the deMMUer is being loaded

The emulator uses the foreground monitor to load reverse translations into the deMMUer. Depending on the complexity of your tables, this process can take a long time. If there are portions of your target program that must not be interrupted for long periods of time, make sure your code is executing in safe regions before you load the deMMUer. You might set a breakpoint in a region of your target program that is outside the time-critical regions and perform the load of the deMMUer after the breakpoint is executed.

The analyzer must be off

Your analyzer must not be making a trace when you load the deMMUer. Otherwise, part of the trace will be based on physical addresses and the other part will be based on logical addresses.

Expect strange addresses if you analyze physical memory with multiple logical mappings

The deMMUer can only translate a physical address into one logical address. If two programs both use the same physical space (such as when two programs use a single data location), they might refer to that space by two different logical address values (and two different logical address symbols). The deMMUer translation RAM will be loaded with only one of the logical addresses. This means that you

might be analyzing execution of your program and find it accesses a data space at an address you don't recognize, even though the data may be what you expect to see. The unexpected address will be the logical address known to the other program that also uses this location.

The way the deMMUer selects a logical address for a physical address when two or more logical addresses are available is as follows:

- The deMMUer selects the logical address with the lowest address value.
- If one of the addresses is controlled by the MMU tables and one by a transparent translation register, the deMMUer sends the address defined in the MMU tables.
- If one of the logical addresses is within a range defined in the emulation configuration memory map and another is not, the logical address defined in the memory map is sent.

Resource limitations

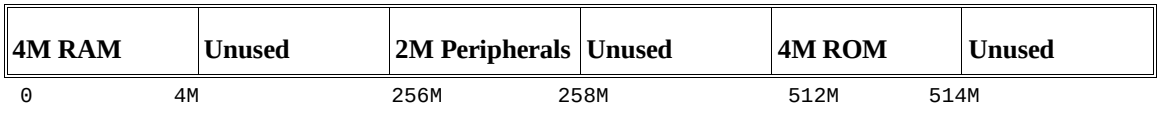
If you load the DeMMUer by using one of the **Settings→DeMMUer→Load from Memory/Memory...** or **load demmuer** commands and your emulator performs its task and returns a prompt to the command line, you won't need to know about the deMMUer resource limitations. When the deMMUer is loaded without any problems, the prompt simply shows on screen and you can proceed with your measurement. The following information will help you deal with problems when you try to load the deMMUer and receive a message such as "Out of deMMUer resources". Note that when you see one error message, there may have been other messages generated at the same time. Display the error log to see all of the error messages that were generated. This will give you additional information about the error that caused the message to appear.

The deMMUer has a table where it records ranges of physical addresses that it can reverse translate to logical addresses. This table has eight entries, and each entry contains a single physical address range. Each address range in the table will be 32 Mbytes. Up to 256 Mbytes of physical addresses can be reverse translated. Normally, entries in this table are allocated automatically, without intervention.

address..address
address..address
address..address
address..address
address..address
address..address
address..address

Example to show resource limitations

Consider the following program arrangement:



Assume a system contains memory and peripherals at three different ranges: one from 0 to 4 Mbytes, one from 256 to 258 Mbytes, and one from 512 to 514 Mbytes. The rest of the physical address space is unused.

If your MMU mapping tables are set up to only allow access to memory in these ranges, your deMMUer will load properly and you can proceed with your measurements. If you failed to restrict your MMU mappings to these physical ranges (instead you provided valid address translations for the entire 4-Gbyte address space), the deMMUer will allocate all eight of its resource blocks in the first 256-Mbyte range, and no deMMUing will be provided for the peripherals and ROM space in the above program.

The Emulation Memory Map Can Help

When the emulator tries to load the deMMUer and finds more physical memory identified in the MMU mapping tables than it can translate in its deMMUer table, it will assign resources to terms defined in the emulation memory map. If the emulation memory map is arranged as follows, the deMMUer will load in a way that ensures the physical ranges of interest will be in the deMMUer.

0 to 3FFFFFH EMUL/RAM
10000000H to 101FFFFFH TARGET/RAM
20000000H to 203FFFFFH TARGET/ROM
default TARGET RAM

When the emulator reads the emulation memory map for help in loading the deMMUer, it sorts the entries: first by size, and second by address range. The smallest address range (256M to 258M) will occupy the first resource block in the deMMUer translation table. Address range (0 to 4M) will occupy the second resource block, and address range (512M to 514M) will occupy the third resource block. The remaining five resource blocks will be assigned to other physical ranges found in the MMU tables, beginning with the lowest addresses. You may see a message indicating some physical addresses will not be translated by the

deMMUer, or Out of DeMMUer resources, because the deMMUer might run out of resource blocks before all of the physical addresses have been assigned reverse translations, but the program spaces you care about will all be reverse translated. You can use the **verbose** option of the deMMUer load command to make sure the program spaces you care about will be reverse translated.

If you map a space greater than 256 Mbytes in the emulation memory map, you will run out of resource blocks before you satisfy the map.

The best way to ensure that all of the address ranges you care about will be reverse translated is to compose an emulation memory map that allocates blocks of physical memory only large enough to accommodate the address space occupied by code you are trying to develop. The deMMUer algorithm will allocate resource blocks in its eight-entry table to reverse translate only those physical address ranges.

With the above example, you could have avoided running out of resources. If you had placed invalid descriptors in your MMU tables in the paths that lead to unused physical address ranges, the deMMUer would have had more than enough resource blocks in its eight-entry table to reverse translate the valid address ranges.

Finally, you can store the present setup of the MMU to a file, and then use an editor to eliminate address ranges that do not need to be reverse translated. This only leaves address ranges that need to be reverse translated in the file. Then you can load this file into the deMMUer. When this file is loaded, the deMMUer creates a set of reverse translations for it, ignoring the MMU setup in the emulator. Refer to "Saving and Restoring DeMMUer Setup Files" in Chapter 3, "Using the Emulation-Bus Analyzer", for how to store and load a deMMUer file.

Dividing the deMMUer table between user and supervisor address space

You can have two sets of MMU translation tables, one under each root pointer (URP and SRP). In this case, the emulator divides the deMMUer table into two equal address spaces. The first four resource blocks provide reverse translations for user physical address ranges, and the last four resource blocks provide reverse translations for supervisor physical address ranges.

There are cases where the deMMUer table will not be divided into two sets of four resource blocks each, even if you are using both root pointers (URP and SRP). If the values of your user root pointer and supervisor root pointer are the same (URP = SRP), and if the user and supervisor function codes are ignored in all of the transparent translation registers, then the deMMUer table will not be divided. It will make its eight resource blocks available to reverse translate either user or supervisor space.

If the user root pointer and supervisor root pointer contain different values, or if function-code mapping is used in any of the transparent translation registers, the deMMUer table will be divided into two 4-block tables as shown below.

address..address@u
address..address@u
address..address@u
address..address@u
address..address@s
address..address@s
address..address@s

Solving Problems

Your program and emulator may be running fine until you turn on the MMU. Then program execution may fail. You may not be able to use features of your emulator. How can this happen? It can happen if the MMU mapping tables are incorrect. When the MMU turns on and starts managing memory by performing tablewalks in tables that are invalid, pages of logical memory may overwrite your stack space, your emulation monitor, or any other address space, making your entire system unusable. If this happens, note where the program is executing. The stack may be inaccessible. The monitor (with its emulation feature set) may be inaccessible. The vector table may be placed in guarded memory. Program data space may become inaccessible.

Using the "display mmu_translations" command to overcome plug-in problems

Plug-in problems involving the MMU are often caused by incorrect mappings in your translation tables. If your logical address is translated to an incorrect physical address, the **Display→MMU Translations** or **display mmu_translations** command can show the details of how your logical addresses are mapped to the wrong physical addresses.

To display the MMU translations when the TC register contains a disable value, include the enable value (8000h) in the Override Processor Register Values section of the Display MMU Translations dialog box, or include **use_value TC** 08000h in your command line entry.

You can also enter the **Display→MMU Translations...** or **display mmu_translations** command to test your mappings before you enable the MMU. This command by itself reads all present translations in your MMU tables. No invalid or illegal paths are shown in the listing. You can read through the display on screen to see if all of your address ranges are represented, and if they are mapped to appropriate space in physical memory.

When you enter the **Display→MMU Translations** or **display mmu_translations** command, the emulator reads the MMU registers (TC, URP, and SRP) and MMU tables. If you do not have correct values in the TC, URP, and SRP registers, the

emulator will let you specify correct values to be used when composing the display of translations You can use the dialog box that is called by the **Display→MMU Translations...** selection and click on Override Processor Register Values and enter the desired values for the TC, SRP, URP, ITT0, ITT1, DTT0, and DTT1 registers (or RECALL desired values from earlier usages). If you are using the command line, you can use the **use_value** option to the **display mmu_translations** command and specify any of the above registers with desired values to obtain the display of MMU translations.

Use the analyzer with the deMMUer to find MMU mapping problems

If your system operates properly until you turn on the MMU, and then it fails, the problem is most likely in the mappings used by the MMU to translate logical addresses to physical addresses. You could go down the list of logical-to-physical translations to see the mapping scheme used to translate each logical address to its corresponding physical address, but normally that would take too much time. The analyzer can help you identify the one, or few, logical addresses that are being mapped incorrectly by the MMU. Then you can choose **Display→MMU Translations...**, click on MMU Tables in the dialog box, and enter the suspect addresses to see the mapping tables used to translate those addresses. On the command line, you can use the **display mmu_translations tables <address>** command to look at the mapping tables used to translate the suspected addresses.

Failure caused by access to guarded memory

If the problem is an access to guarded memory, remember that guarded memory is guarded physical memory. You need to find the logical address that the MMU improperly translated to guarded physical memory and then investigate the mappings the MMU used to perform the translation.

Begin by looking at the registers display (**Display→Registers→BASIC** or **display registers**) to see the value of the logical address in the program counter. Then choose **Display→MMU Translations...**, and use the dialog box, or use the **display mmu_translations tables <address>** command to see the path through the tables that the MMU took when it translated that logical address to a guarded address in physical memory. Note that the value of the program counter may have

changed after the guarded access occurred. In this case, the present address in the program counter may map to proper physical memory.

If the present program counter address does not translate to an address in guarded physical memory, the access to guarded memory may have been caused when your program read or wrote to data memory before the present program counter address appeared. Set up the analyzer to make a trace (with the deMMUer turned on) and trigger at the logical program counter address (by placing the PC address in the entry buffer and choosing **Trace→About ()**, or using the command **trace about address <pc address>**). Selecting a center trigger lets you see activity preceding and following the trigger point. In order to capture every transaction on the emulation bus, qualify all states for capture (don't use the **trace only** option).

If the access occurs again just before the program counter address you used as your trigger specification, you should be able to read back in the trace list and find one or more addresses that could be causing the problem. Then you can try those suspected addresses in commands (choose **Display→MMU Translations...**, click on MMU Tables in the dialog box, and enter the suspect addresses, or on the command line, **display mmu_translations tables <suspect_address>**) to see how each of them is mapped through the MMU tables. This should identify the error in the MMU mapping tables.

If you find a particular address that is mapped to guarded memory, and if the problem seems to be in Table B (for example), you can look at the details of Table B for that address by clicking on Table Level B in the **Display MMU Translations...** dialog box, or by using a command, such as **display mmu_translations tables <address> level B**.

Failure due to system halt

If the emulator or target system or both simply stop operating, set up the analyzer to trace with a trigger-never specification (**Trace→Until Stop** or **trace on_halt**) so that the trace will run continuously until the system stops again. After the system halt occurs again, read the trace list to find the addresses preceding the system halt. Check suspected addresses by choosing **Display→MMU Translations...**, clicking on MMU Tables in the dialog box, and entering the suspect addresses. Or, on the command line, enter **display mmu_translations tables <address>** commands to see how the MMU maps each one to physical memory.

Execution breakpoint problems

If you set a breakpoint in RAM, the emulator modifies memory using a logical address. If you set a breakpoint in ROM, the emulator translates a logical address into a physical address and remembers the physical address as the address where it will jam the breakpoint instruction when it is fetched. If your MMU address translations change while breakpoints are activated, you can get the "undefined software breakpoint" message when you run your program or or the "breakpoint code already exists" message when you attempt to modify the breakpoints.

You set a breakpoint. Then the MMU changes its mappings. Now the logical address where the breakpoint is to occur is translated to a different physical address. No emulation break occurs when the logical address is translated to the new physical address. Some different logical address is translated through the MMU to reach the physical breakpoint address, and the emulator jams the BKPT instruction. When the BKPT instruction is executed, it is at a point in your program where you never set a breakpoint.

You should disable any hardware breakpoints before changing the MMU address translations. Reenable the hardware breakpoints after the MMU address translations have been modified.

A "can’t break into monitor" example

The following example assumes you mapped your foreground monitor beginning at address 4000H. You connected your emulator into your target system and ran your target program (which set up the MC68040 MMU). You tried to break into the emulation monitor and got the message, "Can’t break into monitor."

The emulator can’t break into the monitor because it can’t find the monitor. The MMU mapped the foreground monitor to physical address space that is not a 1:1 translation from logical address space.

A variety of failure modes can happen at this point. Your emulation system may execute unknown code, or it may simply halt.

To analyze this problem, reset into the monitor by choosing **Execution→Reset** and then **Execution→Break**, or using command-line commands: **reset**, and then **break**.

The **reset** does not change the content of the MMU mapping tables or registers. It only disables the "enable bit" in the TC register of the MMU. Now you can look at the translations that are performed by the MMU to find the translation that was applied to your foreground monitor. Choose **Display→MMU Translations...**, and override the TC register with 8000h in the dialog box, or enter the command: **display mmu_translations use_value TC** 8000h.

Remember to override the TC register value with 8000h in order to enable the reading of the MMU tables by the emulator.

The display will show a list of the logical-to-physical address translations that will be performed when the MMU is enabled. Find the logical address range that contains your foreground monitor and see the physical address where it is mapped. The physical address range needs to be the same as the logical address range for the emulator to be able to find the monitor.

The display you get by choosing **Display→MMU Translations...**, or entering the command: **display mmu_translations** (with the required value to override the TC register), might show the logical address range of your foreground monitor mapped to physical addresses beginning at C000H, as follows:

```
Logical Address       Physical Address
00004000..00004fff    0000C000..0000cfff@a
```

The next step in this analysis is to display the MMU mapping table for the logical base address of the foreground monitor. You might choose **Display→MMU Translations...**, and in the dialog box, click on MMU Tables and type in Address 4000H (plus the TC register override). Or, you could enter the command **display mmu_translations tables 4000h use_value TC 8000h**. In this example, you would see the following display of mappings:

```
Logical Address (hex)     0    0    0    0    4    0    0    0
Logical Address (bin)  0000 0000 0000 0000 0100 0000 0000 0000
Table Level            AAAA AAAB BBBB BBCC CCCC PPPP PPPP PPPP

LEVEL INDEX LOCATION   CONTENTS   TBL/PAGE G Ux S CM M U W UDT/PDT
SRP                    00000200   00000200                 RESIDENT
A      000  00000200   0000040b   00000400              y n RESIDENT
B      000  00000400   0000060b   00000600              y n RESIDENT
C      016  00000640   0000c01f   0000C000 n 00 y cw y y n RESIDENT

Physical Address (hex) = 0000c000
```

In the example display, the foreground monitor whose logical address is 4000 was placed in physical address C000. Table C points to the page containing the foreground monitor. The base address of Table C is 00000600, and the content used by logical address 4000 is at index 016 whose physical address is 00000640. The content of this address is 0000C000H (the address of the page containing the monitor).

To solve the problem in this example, you can obtain the needed 1:1 mapping by modifying the content of the MMU table directly with the following command-line command: **modify memory physical 00000640h to 0000401fh**.

After this modification, you can get a new display of the mapping tables for logical address 4000h to see if your modified MMU tables now map your foreground monitor correctly. Choose **Display→MMU Translations...**, and in the dialog box, click on MMU Tables and type in Address 4000H (and include the value 8000h to override the TC register). Or, enter the command
display mmu_translations tables 4000h use_value TC 8000h.

```
Logical Address (hex)     0    0    0    0    4    0    0    0
Logical Address (bin)  0000 0000 0000 0000 0100 0000 0000 0000
Table Level            AAAA AAAB BBBB BBCC CCCC PPPP PPPP PPPP

LEVEL INDEX LOCATION   CONTENTS   TBL/PAGE G Ux S CM M U W UDT/PDT
SRP                    00000200   00000200                 RESIDENT
A      000  00000200   0000040b   00000400              y n RESIDENT
B      000  00000400   0000060b   00000600              y n RESIDENT
C      016  00000640   0000401f   00004000 n 00 y cw y y n RESIDENT

Physical Address (hex) = 00004000
```

The above modifications will provide the proper mapping for your system until you rerun the portion of your target program that sets up the MMU. Then the same problem will occur again. To fix the problem permanently, you need to modify your target program so it provides a 1:1 mapping for the address space where the foreground monitor is located.

11

Emulator Commands

The command syntax reference for the emulator softkey interface

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

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→Load→Trace Data	load trace
File→Load→Trace Spec	load trace_spec
File→Load→DeMMUer	load demmuer <file>
File→Store→Trace Data	store trace
File→Store→Trace Spec	store trace_spec
File→Store→BBA Data	bbaunload
File→Store→DeMMUer (From MMU Tables)	store demmuer <file>
File→Copy→Display	copy display to
File→Copy→Memory	copy memory to
File→Copy→Data Values	copy data to
File→Copy→MMU Translations	copy mmu_translations 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
File→Log→Playback	<command file>
File→Log→Record	log_commands to
File→Log→Stop	log_commands off
File→Emul700→Emulator/Analyzer	N/A
File→Emul700→<other products>	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	!

Pulldown	Command Line
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→Real ()→extended	display memory --EXPR-- real extended
Display→Memory→Real ()→packed	display memory --EXPR-- real packed
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>
Display→MMU Translations	display mmu_translations
Display→Trace	display trace
Display→Registers→<type>	display registers <type>
Display→Breakpoints	display software_breakpoints
Display→Status	display status
Display→Simulated IO	display simulated_io
Display→Global Symbols	display global_symbols
Display→Local Symbols ()	display local_symbols_in --SYMB--
Display→Pod Commands	display pod_command
Display→Error Log	display error_log
Display→Event Log	display event_log
Modify→Emulator Config	modify configuration
Modify→Memory	modify memory
Modify→Memory at ()	modify memory --EXPR--
Modify→Register	modify register

Pulldown	Command Line
Execution→Run→from PC	run
Execution→Run→from ()	run from --EXPR--
Execution→Run→from Transfer Address	run from transfer_address
Execution→Run→from Reset	run from reset
Execution→Run→until ()	run until --EXPR--
Execution→Step Source→from PC	step source
Execution→Step Source→from ()	step source from --EXPR--
Execution→Step Source→from Transfer Address	step source from transfer_address
Execution→Step Instruction→from PC	step
Execution→Step Instruction→from ()	step from --EXPR--
Execution→Step Instruction→from Transfer Address	step from transfer_address
Execution→Break	break
Execution→Reset	reset
Breakpoints→Display	display software_breakpoints
Breakpoints→Enable	modify software_breakpoints enable/disable
Breakpoints→Temporary()	modify software_breakpoints set --EXPR-- temporary
Breakpoints→Permanent()	modify software_breakpoints set --EXPR-- permanent
Breakpoints→Force HW→Permanent()	modify software_breakpoints set --EXPR-- permanent force_hw
Breakpoints→Force HW→Temporary()	modify software_breakpoints set --EXPR-- temporary force_hw
Breakpoints→Set All	modify software_breakpoints set
Breakpoints→Clear ()	modify software_breakpoints clear --EXPR--
Breakpoints→Clear All	modify software_breakpoints clear

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→DeMMUer→Load from Memory	load demmuer
Settings→DeMMUer→Load from File	load demmuer <file>
Settings→DeMMUer→Verbose	N/A (toggles the verbose mode of the DeMMUer load function)
Settings→DeMMUer→Enable	set demmuer on/off
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)

Emulator Configuration: Memory Map

Pulldown	Command Line
File→Print Map	print
File→Exit	end
Map→Add New Entry	N/A (calls the associated dialog box)
Map→Modify Entry→<entry#>	N/A (calls the associated dialog box)
Map→Delete Entry→<entry#>	delete <entry#>
Map→Delete All	delete all
Map→Default Memory Type→Target RAM→ Transfer Cache Inhibit OFF	default target ram
Map→Default Memory Type→Target RAM→ Transfer Cache Inhibit ON	default target ram transfer_cache_inhibit
Map→Default Memory Type→Target ROM→ Transfer Cache Inhibit OFF	default target rom
Map→Default Memory Type→Target ROM→ Transfer Cache Inhibit ON	default target rom transfer_cache_inhibit
Map→Default Memory Type→Guarded	default guarded
Settings→Command Line	N/A (toggles the command line)

How Popup Menus Map to the Command Line

Mnemonic Memory Display Popup	Command Line
Set/Clear Software Breakpoint	modify software_breakpoints set/clear --EXPR-- permanent
Edit Source	! vi +<line> <file> ! no_prompt_before_exit
Run Until	run until fcode sp --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 Popup	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 Popup	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

Trace Display Popup	Command Line
Disassemble From	display trace disassemble_from_line_number <line#>
Edit Source	! vi +<line> <file> ! no_prompt_before_exit
Display Memory At	display memory <address> mnemonic

Status Line Popup	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 Popup	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)

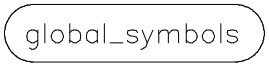
Emulation Configuration: Memory Map Display Popup	Command Line
Modify Entry	delete <entry#>, <add <entry#>>
Add New Entry	<add <entry#>>
Delete Entry	delete <entry#>

Syntax Conventions

Conventions used in the command syntax diagrams are defined below.

Oval-shaped Symbols

Oval-shaped symbols contain command tokens. Command tokens, together with symbols and numeric values, make up complete Softkey Interface commands. Most command tokens appear on softkey labels. Those that do not appear as softkeys must be typed into the command line (for example, **log_commands** and **wait**). An example of an oval-shaped symbol is as follows:



Rectangular-shaped Symbols

Rectangular-shaped symbols contain prompt softkeys, softkey-changers, and references to other syntax diagrams. Prompt softkeys are enclosed in angle brackets (< and >). Softkey-changers are enclosed in dashes (--). References to other diagrams are shown in all capital letters without any enclosing symbols.

Examples of all three kinds of rectangular symbols follow:

Prompt Softkey. Press to get a hint about the kind of information needed.

Softkey Changer. Press to get another set of softkeys. Some softkey changers have their own syntax diagrams in this chapter.

Reference to a diagram showing details in this chapter.

Circles

Circles contain operators and delimiters used in expressions and on the command line. An example of a circle symbol is as follows:

The —NORMAL— Key

The softkey labeled **—NORMAL—** is a special softkey-changer; use it to return to the former set of softkeys. For example, you can press the **—EXPR—** softkey to call up a set of prompt softkeys to help you complete an expression. After you complete the expression, you can return to the set of softkeys containing the **—EXPR—** softkey by pressing the **—NORMAL—** softkey.

Summary of Commands

Softkey Interface commands are summarized in the following table:

Emulator Commands

UNIX_COMMAND[3]	display global_symbols	modify keyboard_to_simio[2]
break	display local_symbols_in	modify memory[4]
cd (change directory)[3]	display memory[4]	modify register[1]
cmb_execute	display pod_command	modify software_breakpoints[1]
<command file>[3]	display local_symbols_in	name_of_module[3]
copy data[4]	display mmu_translations[5]	performance_measurement_end
copy display	display registers[1]	performance_measurement_init
copy error_log	display simulated_io[2]	performance_measurement_run
copy event_log	display software_breakpoints	pod_command
copy global_symbols	display status	pwd (print working directory)[3]
copy help	display trace	pws (print working symbol)[3]
copy local_symbols_in	end	reset
copy memory[4]	help[3]	run
copy mmu_translations[5]	load <absolute_file>	set
copy pod_command	load configuration	specify
copy registers[1]	load demmuer[5]	step
copy software_breakpoints	load emul_mem	stop_trace
copy status	load trace	store demmuer[5]
copy trace	load trace_spec	store memory
cws(change working symbol)[3]	load user_memory	store trace
display data[4]	log_commands[3]	store trace_spec
display error_log	modify configuration	trace
display event_log		wait[3]

[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 target memory or in emulation memory that is *not* dual-port.
[5] These commands are available only when the MMU is enabled.

break

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

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

running	Break diverts the processor from execution of your program to the emulation monitor.
reset	Break releases the processor from reset, and diverts execution to the monitor.
running in monitor	The **break** command does not perform any operation while the emulator is executing in the monitor.

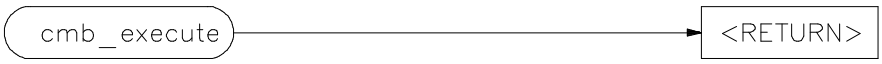
Example

```
break
```

See Also

```
help break
run
step
```

cmb_execute



The **cmb_execute** command causes the emulator to emit an EXECUTE pulse on its rear panel 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.

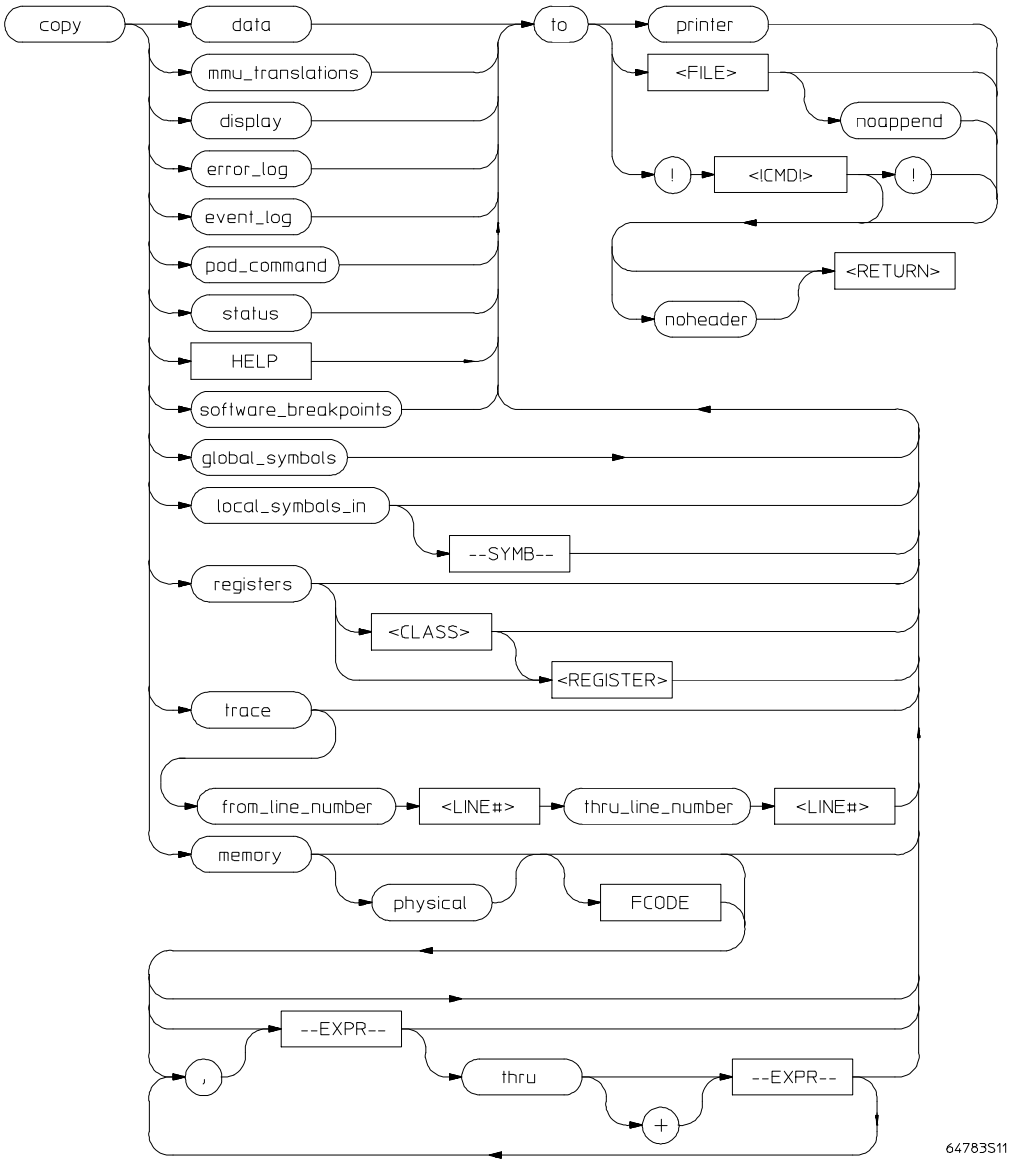
Example

```
cmb_execute
```

See Also

```
help cmb
help cmb_execute
help specify
specify run
specify trace
```

copy


copy
data
mmu_translations
display
error_log
event_log
pod_command
status
HELP
software_breakpoints
global_symbols
local_symbols_in
--SYMB--
registers
<CLASS>
<REGISTER>
trace
from_line_number
<LINE#>
thru_line_number
<LINE#>
memory
physical
FCODE
--EXPR--
,
thru
--EXPR--
+
to
printer
<FILE>
noappend
!
<!CMD!>
!
<RETURN>
noheader
64783S11

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.

, 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.

<CLASS> Specifies a particular class of the emulator registers. See the "registers" parameter in this section for more information about processor registers.

< !CMD!> This represents a 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 Softkey Interface command line execution after the UNIX command executes. Emulation is not affected when using a UNIX command that is a shell intrinsic.

data This allows you to copy a list of memory contents formatted in various data types (see display data).

display This allows you to copy the display to a selected destination.

error_log This allows you to copy the most recent errors that occurred.

event_log This allows you to copy the most recent events that occurred.

! 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.

—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.

<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**

from_line_number — This specifies the trace list line number from which copying will begin.

global_symbols — This lets you copy a list of global symbols to the selected destination.

help — The help command is not shown on the softkeys. It allows you to obtain a copy of a help file on your printer. For details of the help command, refer to the help syntax diagram in this chapter.

<LINE#> — Use this with **from_line_number** and **thru_line_number** to specify the starting and ending trace list lines to be copied.

local_symbols_in — This lets you copy all the children of a given symbol to the selected destination. See the **—SYMB—** syntax page and the *SRU User's Guide* for information on symbol hierarchy.

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.

memory — This allows you to copy a list of the contents of memory to the selected destination. 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 target memory or in emulation memory that is not dual-port.

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.

mmu_translations — This allows you to copy a list of all present translations of all logical addresses 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 — This copies the information into a file without headings.

physical — This allows you to specify that the addresses to be copied are addresses in physical memory.

pod_command — This allows you to copy the most recent commands sent to the HP 64700-Series emulator/analyzer. For details, refer to the syntax page for the **pod_command** in this chapter.

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, 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 — This allows you to copy a list of the contents of the emulation processor registers to the selected destination. The **copy register** command is not available if the emulator is configured for real-time-only operation.

With no options specified, the basic register class is copied. Basic registers include: PC, STATUS, USP, ISP, MSP, CACR, D0 through D7, A0 through A7, VBR, DFC, and SFC.

Other registers you can copy include:

The FPU registers: FPCR, FPSR, FPIAR, and FP0 through FP7.

The MMU registers: ITT0, ITT1, DTT0, DTT1, MMUSR, TC, SRP, and URP.

<REGISTER> — Specifies the name of an individual register.

software_ breakpoints	This option lets you copy a list of the current execution breakpoints to a selected destination.
status	This allows you to copy emulation and analysis status information.
—SYMB—	This option represents the symbol whose children are to be listed. See the **—SYMB—** syntax diagram and the *SRU User's Guide* for information on symbol hierarchy.
thru_line_number	Specifies the last line number of the trace list to include in the copied range.
to	This allows you to specify a destination for the copied information.
trace	This lets you copy the current trace listing to the selected destination. Initial values are the same as specified by the last **display trace** command.

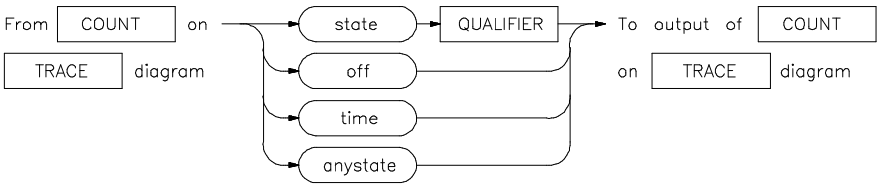
Examples

```
copy local_symbols_in prog68k.S: to printer

copy local_symbols_in cmd_rdr.s: to myfile

copy memory START to printer

copy memory 0 thru 100H , START thru +5 , 500H ,
TARGET2 to memlist

copy memory 2000h thru 204fh to memlist

copy registers BASIC to printer

copy registers to reglist

copy trace to tlist

copy trace from_line_number 0 thru_line_number 5
to longtrac

copy trace to !mail myfriend@col.hp!
```

See Also

display local_symbols_in <SYMB>
display memory
display registers
display trace
modify memory
modify registers
store memory
store trace
help ***copy***
help ***registers***
help ***trace***

COUNT



The trace depth of the deep analyzer depends on whether or not you installed memory modules on the analyzer card (refer to Chapter 19, "Installation and Service", for details). The maximum depth of the 1K analyzer is 1024 states. A state is a unique combination of address, data, and status values occurring on the emulation bus simultaneously. When counting is off, the 1K analyzer can store 1024 states in the trace buffer. When counting is on, the 1K analyzer can only store 512 states in the trace buffer. That is because the 1K analyzer must now use two states in the trace buffer for each state captured on the analyzer bus. One of the two states stores the state information itself. The other state stores the count information associated with the state information.

The deep analyzer can always store the same maximum number of states in its state buffer, regardless of whether counting is turned on or off.

By default, the interface displays a maximum of 256 of states. You can increase the depth of the trace display buffer to display up to the maximum amount of states that are available for display. The following command increases the trace display depth to 512 states:

```
display trace depth 512
```

By default, the counting function is turned off in the 1K analyzer, and set to time in the deep analyzer.

anystate — This option allows you to set up the **counting** parameter for the analyzer to count on any state.

off — This option turns off trace counting capability. As previously explained, turning off counting provides a larger trace depth if using the 1K analyzer.

QUALIFIER — This is defined by you and used with the **state** option to define the states to be captured by the analyzer.

state — This causes the emulation-bus analyzer to count occurrences of the specified state during a trace measurement.

time — This option causes the emulation-bus analyzer to count the time between states captured during the trace measurement.

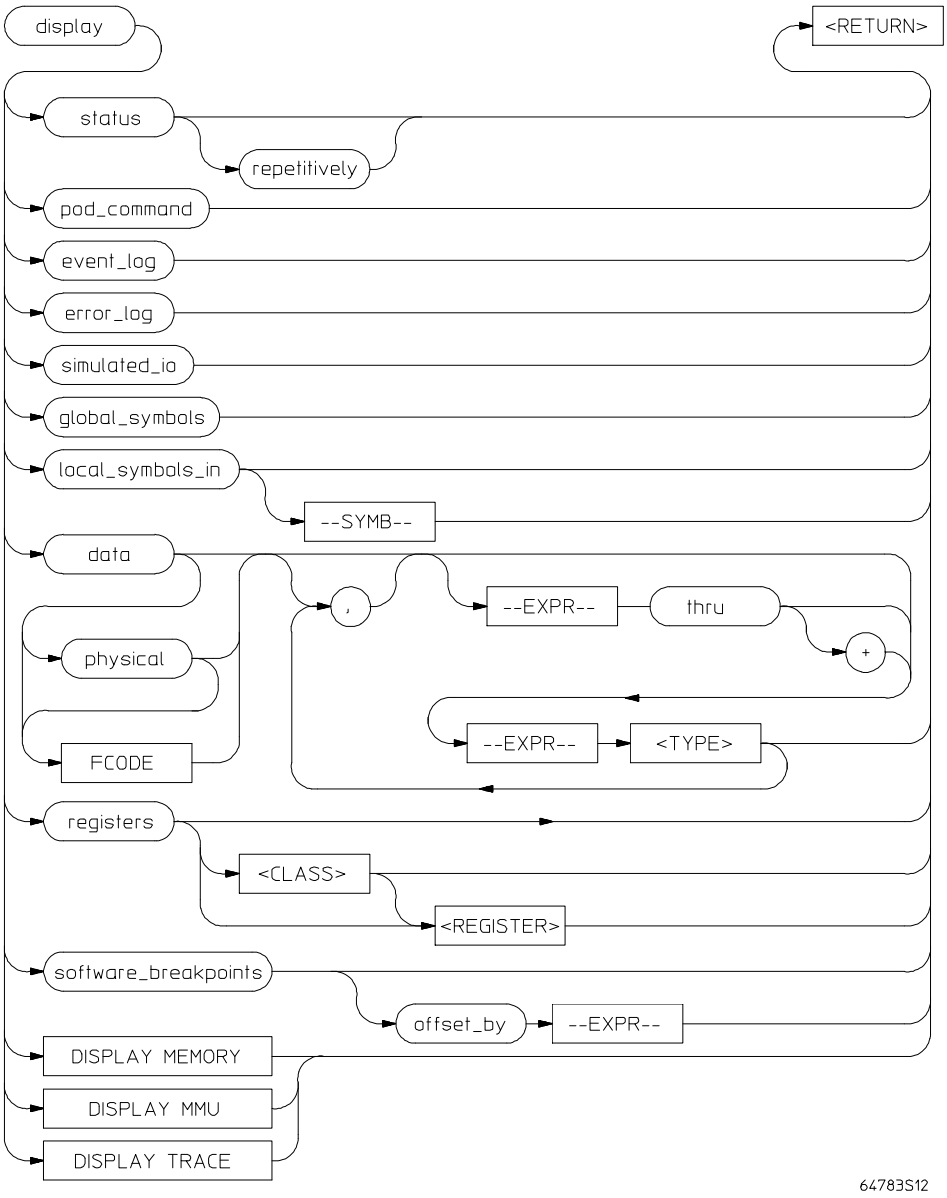
Examples

```
trace after START counting state LOOP2

trace counting time
```

See Also

help ***trace***
trace

display


display
<RETURN>
status
repetitively
pod_command
event_log
error_log
simulated_io
global_symbols
local_symbols_in
--SYMB--
data
,
--EXPR--
thru
+
physical
FCODE
--EXPR--
<TYPE>
registers
<CLASS>
<REGISTER>
software_breakpoints
offset_by
--EXPR--
DISPLAY MEMORY
DISPLAY MMU
DISPLAY TRACE
64783S12

You can use the **up arrow**, **down arrow**, **PREV PAGE**, and **NEXT PAGE** 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.

, 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.

data You can display the values of simple data types in your program. This command saves you time; without it 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.

For 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.

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.

—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.

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.

global_symbols — This option lets you display a list of all global symbols in memory.

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.

local_symbols_in — This option lets you display all the children of a given symbol. Local symbols of **—SYMB—** are the ones that are children of the file or scope specified by **—SYMB—**. That is, they are defined in that file or scope. Displaying the local symbols sets the current working symbol to the one specified.

—SYMB— — This option represents the symbol whose children are to be listed. See the **—SYMB—** syntax diagram and the *SRU User's Guide* for more information on symbol hierarchy and representation.

memory — This option allows you to display the contents of memory.

offset_by — This option allows you to offset the listed execution 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.

physical — This allows you to specify that the addresses to be displayed are addresses in physical memory.

pod_command — This option lets you display the Terminal Interface screen. You must display the Terminal Interface screen to view the output of any Terminal Interface (pod commands) you have used or are going to use. For details, refer to the syntax page for the pod_command in this chapter.

repetitively — This optional part of the **display status** command causes the emulator status screen to be updated continuously.

registers — This allows you to display the contents of emulation processor registers.

If a **step** command was 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 (refer to **set symbols** in this chapter).

With no options specified, the basic register class is displayed. Basic registers include: PC, STATUS, USP, ISP, MSP, CACR, D0 through D7, A0 through A7, VBR, DFC, and SFC.

Other registers you can display include:

The FPU registers: FPCR, FPSR, FPIAR, and FP0 through FP7.

The MMU registers: ITT0, ITT1, DTT0, DTT1, MMUSR, TC, SRP, and URP.

<CLASS> This allows you to display a particular class of processor registers.

<REGISTER> This displays an individual register.

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.

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 on setting up simulated I/O, refer to the question “Modify simulated I/O configuration?” in Chapter 8, "Configuring the Emulator".

software_ breakpoints This option lets you display the current list of execution breakpoints in software.

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 temporary, pending, permanent, inactivated, or unknown. Refer to "Using Execution Breakpoints" in Chapter 4, "Using the Emulator".

An “unknown” breakpoint status will occur if you set the breakpoint, and then remap the breakpoint address as guarded. A pending breakpoint causes the processor to enter the emulation monitor or background memory upon execution of that breakpoint. Executed breakpoints are listed as inactivated.

Breakpoint entries that show an inactive status can be reactivated by executing the following command:

```
modify software_breakpoints set
```

A label column also may be displayed for addresses that correspond to a symbol. See the **set** command for details.

status — This displays the emulator and trace status screen.

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-dimensional and multi-dimensional arrays. Arrays are displayed in the order specified by the language definition, typically row major order for most Algol-like languages. Including the + operator allows you to specify an address range as an offset from the 'base' address specified in —EXPR—.

trace — This displays the current trace list.

<TYPE> — Specifies the format in which to display the information, as shown in the following table. (Data type information is not available from the symbol database, so you must specify it.)

<TYPE>	**Meaning**
byte	Hex display of one 8-bit location.
word	Hex display of one 16-bit location.
long	Hex display of one 32-bit location.
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..127. Control characters and values in the range 128..255 are displayed as a period (.).

Examples

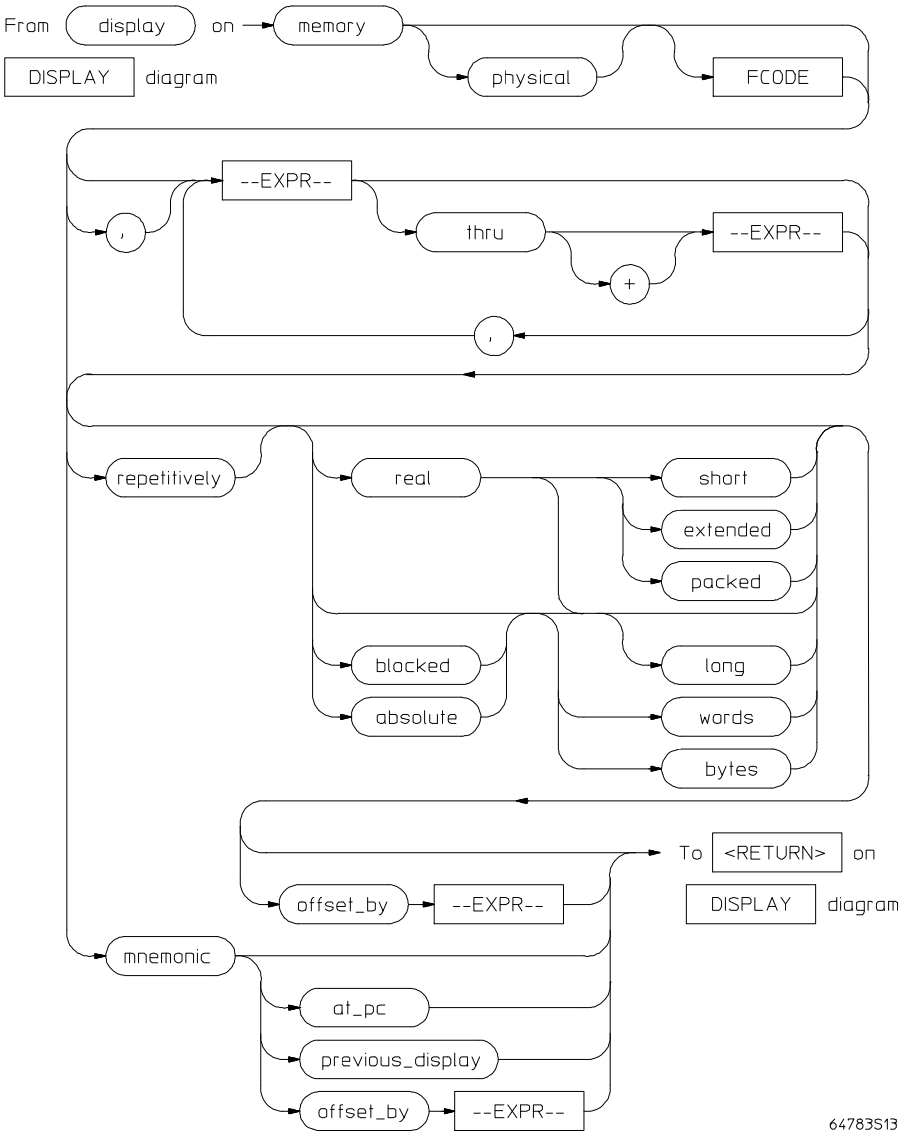
```
display  event_log
display  local_symbols_in  cmd_rdr.s:
display  global_symbols
display  local_symbols_in  temp1.S:
display  local_symbols_in  prog68k.S:main
display  simulated_io
display  software_breakpoints
display  software_breakpoints  offset_by  1000H
display  registers
display  registers  BASIC

display  data Msg_A thru +17 char, Stack long
set symbols on
set width label 30
display data , _clocktic thru +4 char, _duration thru
+4 char
```

See Also

```
copy
help  copy
help  display
help  registers
help  software_breakpoints
modify  registers
modify  software_breakpoints
set
step
cws
pws
```

The following pages describe various **display memory, display MMU,** and **display trace** syntax diagrams.

DISPLAY MEMORY



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 from the one displaying memory mnemonic (see the discussion on windowing capabilities in Chapter 3, "Using the Emulator/Analyzer Interface").

Pending execution breakpoints in software 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
```

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. If you are running a program when you enter this command, the memory will be displayed at the address pointed to by the program counter at the moment you enter the command.

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.

extended	Displays memory as 96-bit IEEE-754 real numbers.
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 the memory listing as long word values. When used with the **real** parameter, **long** displays memory in a 64-bit real number format.
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.
packed	Displays memory as 96-bit Motorola-format packed real numbers.
physical	This allows you to specify that the addresses to be displayed are addresses in physical memory.
previous_display	Places the previous display window on screen. This token lets you switch the display back and forth between the previous window and the new window. This is useful when you are stepping a program in memory and it suddenly jumps to a different location in memory, changing your display window.
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, because 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, or other feature is used that modifies memory, 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 PAGE**, and **PREV PAGE** keys to view additional memory locations.

words — Displays the memory listing as 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.

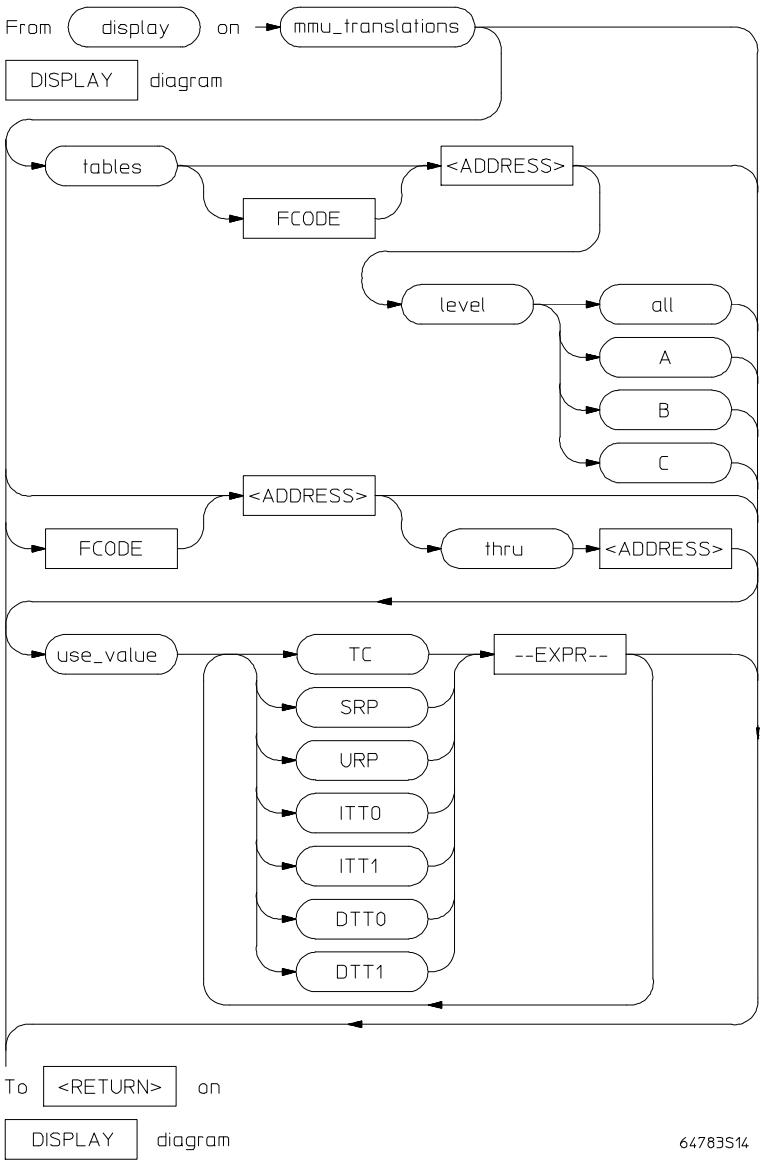
Examples

```
display memory 2000h thru 204fh blocked words

display memory 2000h thru 202fh , 2100h real long

display memory 400h mnemonic
set symbols on
set source on
display memory main mnemonic
```

See Also

copy memory
cws
help display
modify memory
pws
set
store memory

DISPLAY MMU


From
display
on
mmu_translations
DISPLAY
diagram
tables
<ADDRESS>
FCODE
level
all
A
B
C
<ADDRESS>
FCODE
thru
<ADDRESS>
use_value
TC
--EXPR--
SRP
URP
ITT0
ITT1
DTT0
DTT1
To
<RETURN>
on
DISPLAY
diagram
64783S14

The **display mmu_translations** command is used to display all of the present translations for all logical addresses, or for only a limited range of logical addresses. Further, you can display the details of how a single logical address is mapped through the tables to its corresponding physical address. Finally, you can display the details of a single translation table used by a selected logical address.

You can use the **display mmu_translations** command to view the present set of valid translations, even when the TC register and the root pointer registers are invalid. Parameters in the **display mmu_translations** command let you specify values to use when none exist in these MMU registers.

<ADDRESS> The address or address range specifies a logical address reference for the MMU information to be displayed. This takes the form of --EXPR-- (see the --EXPR-- syntax diagram).

—EXPR— An expression is a combination of numeric values, symbols, operators, and parentheses, specifying a value, such as an 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.

level

all	Shows the details of each table for the single address you specify with this comand. This is the same as ignoring the **level** option.
A	Shows the content of Table A for the logical address you included in your command.
B	Shows the content of Table B for the logical address you included in your command.
C	Shows the content of Table C for the logical address you included in your command.

mmu_translations — Shows a list of the valid MMU mappings. One entry in the list is allocated for each page or group of adjacent pages in the system.

tables — Shows the details of the translation through the tables for the logical address you included in your command.

thru — Allows you to enter the upper address in a range of addresses.

use_value — Lets you specify a value to be used in place of the present content of the TC, SRP, URP, ITT0, ITT1, DTT0 and DTT1 registers when reading the tables and showing the address mappings.

Examples

Show all of the valid logical-to-physical mappings in the MMU:

```
display mmu_translations
```

Show all of the logical-to-physical mappings for logical addresses in the range of 7FF0 through 800F:

```
display mmu_translations 7ff0H thru 800fH
```

Show the table details used to translate logical address 400:

```
display mmu_translations tables 400H
```

Show the details of Table A used to translate logical address 40FC.

```
display mmu_translations tables 40fcH level A
```

Show the present MMU mappings based on a URP register value of 80002000 instead of the present URP register value:

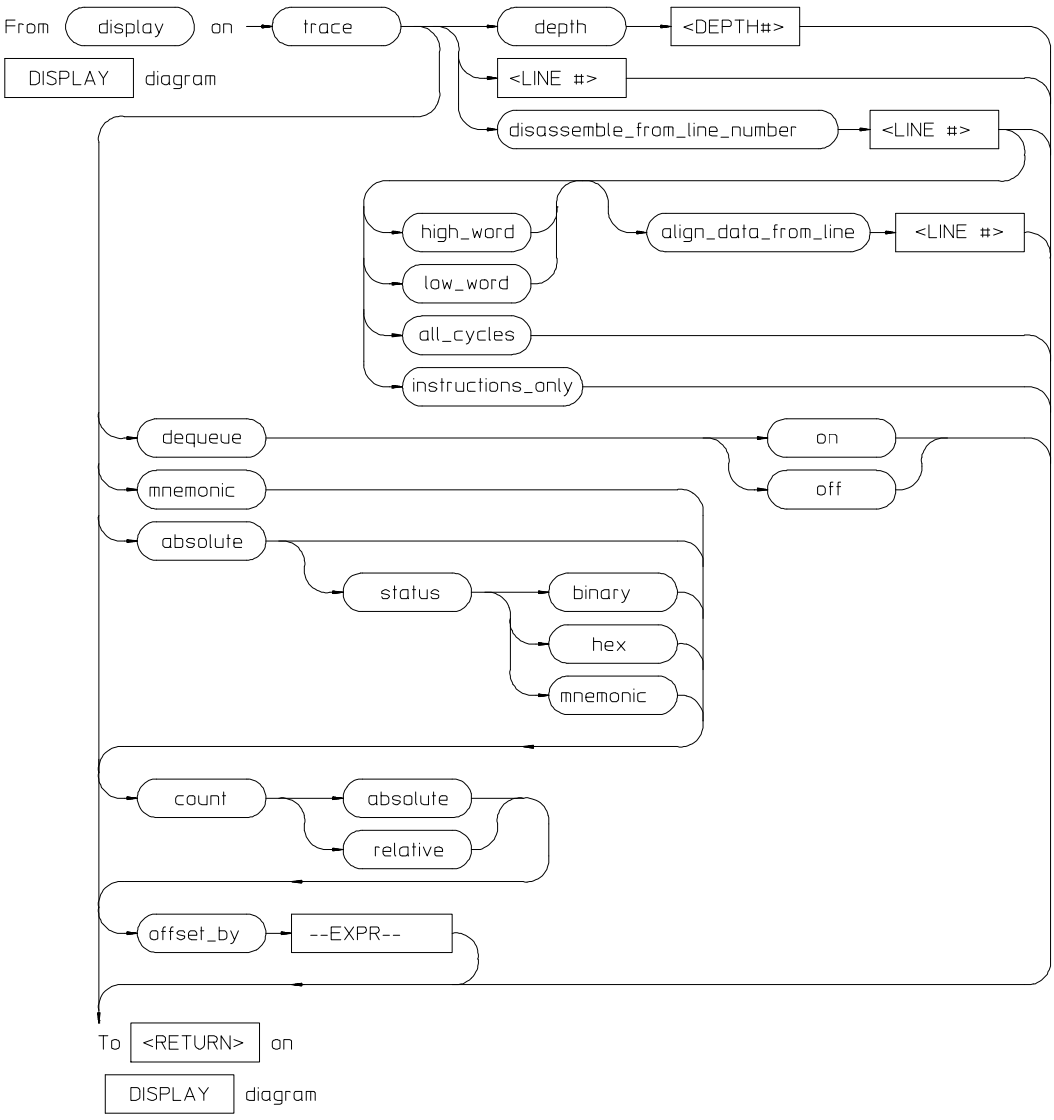
```
display mmu_translations use_value URP 80002000h
```

See Also

load demmuer (Preparing the deMMUer to reverse MMU translations.)

set demmuer on/off (Obtaining logical or physical addresses in the trace list.)

DISPLAY TRACE



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_by**, code that has been relocated (making symbolic information invalid) will have its addresses set so that symbolic information is again valid. If desired, **offset_by** can be used to show instructions in the listed trace at the same addresses where they appear in the assembled or compiled program listing.

The **count** parameter lists the current trace of time or state either relative to the previous event in the trace list or as an absolute count measured from the trigger event. If time counts are currently selected, the **count** parameter causes an absolute or relative time count to be listed. If the current trace contains state counts, a relative or absolute state count results.

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
```

absolute — Lists trace information in hexadecimal format, rather than mnemonic opcodes.

align_data_from_line — Use this to correct data-alignment problems if you see any in a dequeued trace list. If you see that the dequeuer has aligned data with the wrong instructions, use this token to select the correct data alignment by specifying the line that should begin a data realignment (**align_data_from_line** 36).

all_cycles — Used to specify that all cycles should be included in the inverse assembled information shown in the trace list.

count

absolute	This lists the state or time count for each event of the trace as the total count measured from the trigger event.
relative	This lists the state or time count for each event of the trace as the count measured relative to the previous event.

depth

<DEPTH#>	This defines the number of states to be uploaded by the Softkey Interface. After you have changed the trace depth, execute the command **wait measurement_complete** before displaying the trace. Otherwise the new trace states will not be available.

dequeue	This obtains a trace list showing the activity of the emulation processor during the trace. Unused prefetches are eliminated from this display, and data transactions are aligned with the instructions that caused them to occur.
disassemble_from_line_number	This causes the inverse assembly software to begin disassembling the trace code from the specified line number. This feature is required for processors where the inverse assembler cannot uniquely identify the first state of an instruction on the processor bus. The command is not available on emulators where the corresponding inverse assembler can identify instructions on the processor bus.
—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.
high_word	Causes inverse assembly to begin with the opcode in the high word of the long word located in the specified trace-memory line number.
instructions_only	Causes the trace list to contain only those lines that show an instruction opcode.
<LINE#>	When used alone, this is the trace list line number to be centered in the display. When used with **disassemble_from_line_number**, this is the line number from which the inverse assembler attempts to disassemble data in the trace list. When used with **align_data_from_line**, this is the line number from which to begin aligning data. The line number specified for data alignment must be the same as, or higher than, the line number specified for the beginning of the trace disassembly.

low_word

Causes inverse assembly to begin with the opcode stored in the low word of the long word at the specified trace memory line number.

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 fill immediately with the captured assembly language instructions.

When using **set source only**, no inverse assembled code is displayed. Therefore, the emulation software will try to fill the display with high-level source code. This requires the emulation software to search for any captured analysis data generated by a high-level language statement.

In conclusion, you should not set the trace list to **set source only** when tracing assembly code. This will result in optimum analyzer performance.

status

binary — Lists absolute status information in binary form.

hex — Lists absolute status information in hexadecimal form.

mnemonic — Lists absolute status information in mnemonic form.

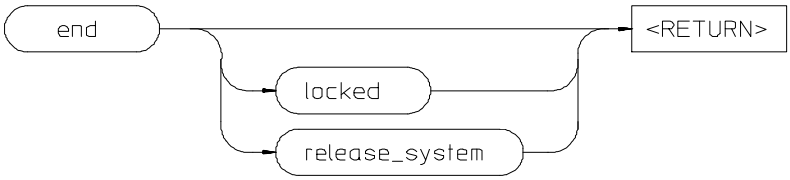
Examples

```
display trace count absolute

display trace absolute status binary

set source on
display trace mnemonic set source only
```

See Also

copy trace
help ***display***
help ***trace***
store trace
set

end



You can end the emulation session and leave the emulator in one of three states.

- You can exit the interface and leave the emulator in a continue state which allows you to re-enter the emulation session.
- You can exit the interface and leave the emulator in a locked state so that it cannot be accessed by other users.
- You can exit the interface and release the emulator for use by others.

Pressing **CNTL d** performs the same operation as pressing **end** .

Pressing **CNTL ** or **CNTL |** performs the same as **end release_system** .

end

If the command is specified without any options, two different things can occur. What occurs depends on whether the interface instance being ended is the only one currently executing. If it is, then this command ends the interface, but leaves the emulator in a continue state. The interface, if restarted, will reload the last configuration used.

If the interface instance is one of several into the same emulation session, then the end command ends the interface instance where the end is issued. Other interface instances into the same emulation session are not affected.

locked

This option allows you to stop all active instances of an emulator Softkey Interface session in one or more windows or terminals. When the emulation session ends, control returns to the UNIX shell, but the emulator is still locked to your user id and is not available to others.

release_system

This option stops all instances of the Softkey Interface in one or more windows or terminals. The emulation system is released for other users. If you do not release the emulation system, others cannot access it.

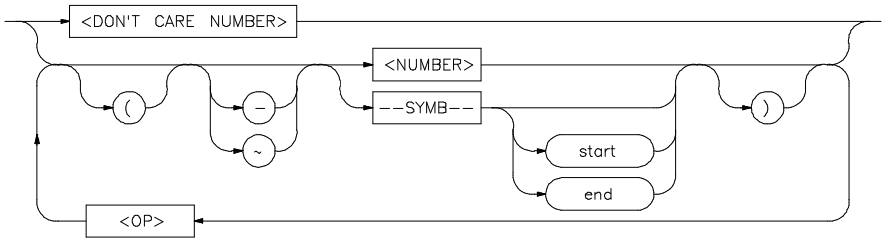
Examples

```
end

end locked

end  release_system
```

See Also

```
emul700  <emulator_name>
help  end
```

—EXPR—



The function of an expression (—EXPR—) is to let you define the address, data, or status expression that fits your needs. You can combine multiple values to define the expression.

Some 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
```

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.

"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):

Operator	Description
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 *SRU 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 the 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.

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.

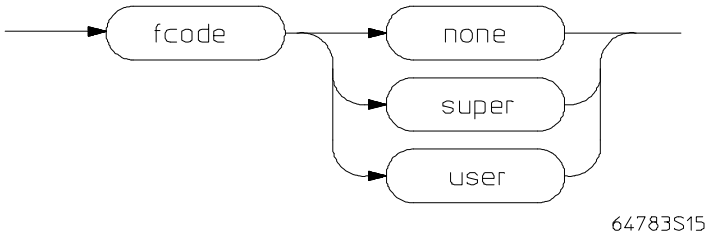
Examples

```
05fxh
DISP_BUF + 5
SYMB_TBL + (OFFSET / 2)
START
cprog.C: line 15 end
```

See Also

```
help  expressions
SYMB
```

FCODE



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

none	Causes the emulator to use the default supervisor address space.
super	Supervisor address space.
user	User address space.

Examples

To copy a portion of user address space to a file:

```
copy memory fcode user 1000H thru 1fffH to mymem
```

To modify a location in supervisor address space:

```
modify memory fcode super 5000h long to 12345678h
```

HELP

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 at the keyboard. You may use a question mark in place of **help** to access the help information.

<HELP_FILE> This represents the name of one of the available help files on the softkey labels. You can either press the softkey that represents the desired help file, or type in the help file name. If you are typing in the help file name, make sure you use the complete syntax. Not all of the softkey labels reflect the complete file name.

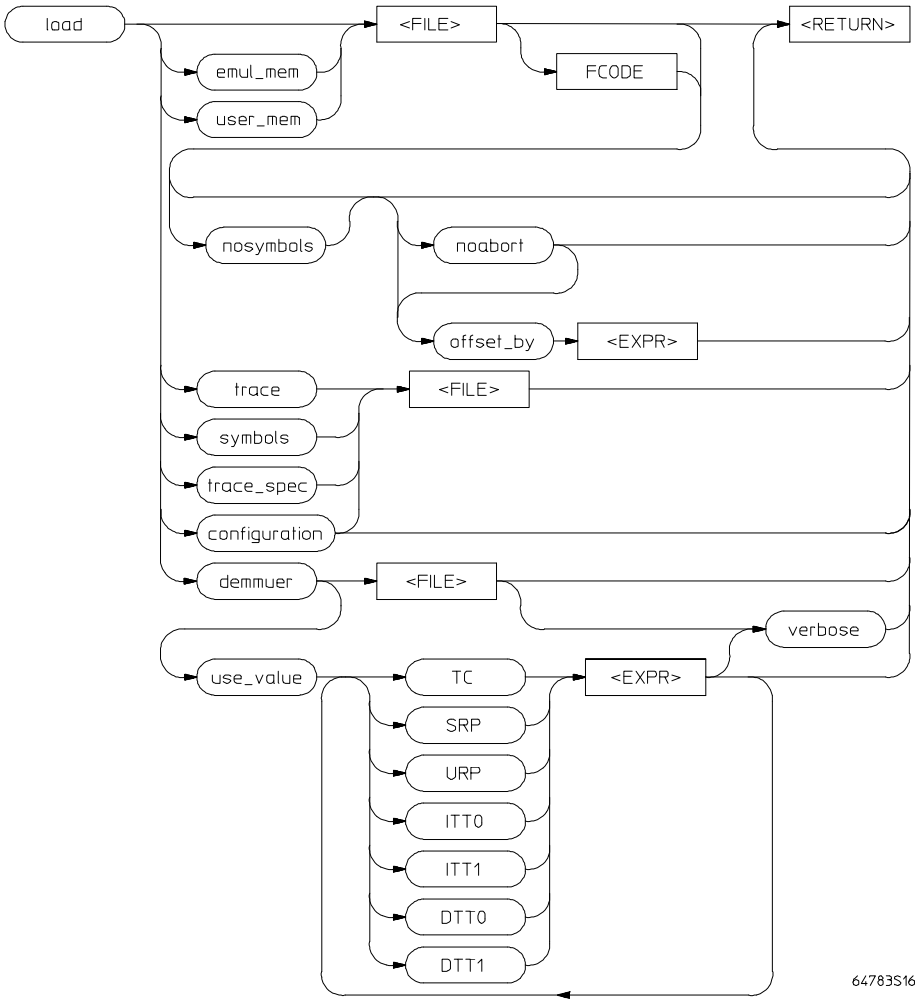
Examples

```
help  system_commands
?  run
```

To display information about the command named **pod_command**, enter:

```
help  pod_command
```

load



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.

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 emulation system as ordinary file names.

The absolute file is loaded into emulation memory by default.

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.

demmuer — This option causes the emulator to read the MMU registers and MMU tables and load the deMMUer with appropriate information to reverse-translate the physical addresses it receives from the emulation bus so it can deliver corresponding logical addresses to the analyzer.

emul_mem — Load the file into emulation memory hardware.

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 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.

offset_by — This option allows you to offset the listed address values from the actual addresses. The offset value you specify will be subtracted from the actual addresses.

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.

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.

use_value	Lets you specify a value to be used in place of the present content of the TC, SRP, URP, ITT0, ITT1, DTT0, and DTT1 registers when reading the tables to determine the reverse translations.
user_mem	Load the file into user (target system) memory hardware.
verbose	This options sets the verbose mode for the deMMUer load function. The verbose mode shows a list of the physical address that can be translated by the deMMUer after loading the deMMUer. If these address translations include function codes, the function codes are shown beside the addresses (example: 000000000..003fffff@s).

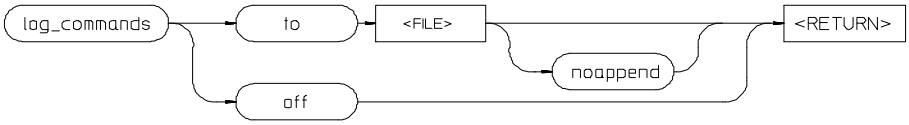
Examples

```
load  sort1

load  configuration  config3

load  trace  trace3

load demmuer verbose
```

See Also

```
display  trace
display  mmu_translations
help  load
```

log_commands



A command file is an ASCII file containing Softkey Interface commands. The interface can read a command file and execute its commands as if they were typed into the command line. Simply type the filename on the command line.

This interface command lets you create command files by logging. When the interface is in logging mode, all commands entered and executed on the command line are also copied to the named file. Once started, logging continues until either logging is turned off or the emulation session is ended.

The **log_commands** command is not on the softkeys. You must type it into the command line to access the remainder of the log_commands softkeys. See the User's Guide for information about entering Softkey Interface commands.

<FILE> This represents the file where you want to store interface commands. If the file does not exist, a new file is created. If the file already exists, the new commands are appended to the present content in the file, unless the **noappend** option is specified.

off This option stops command logging.

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.

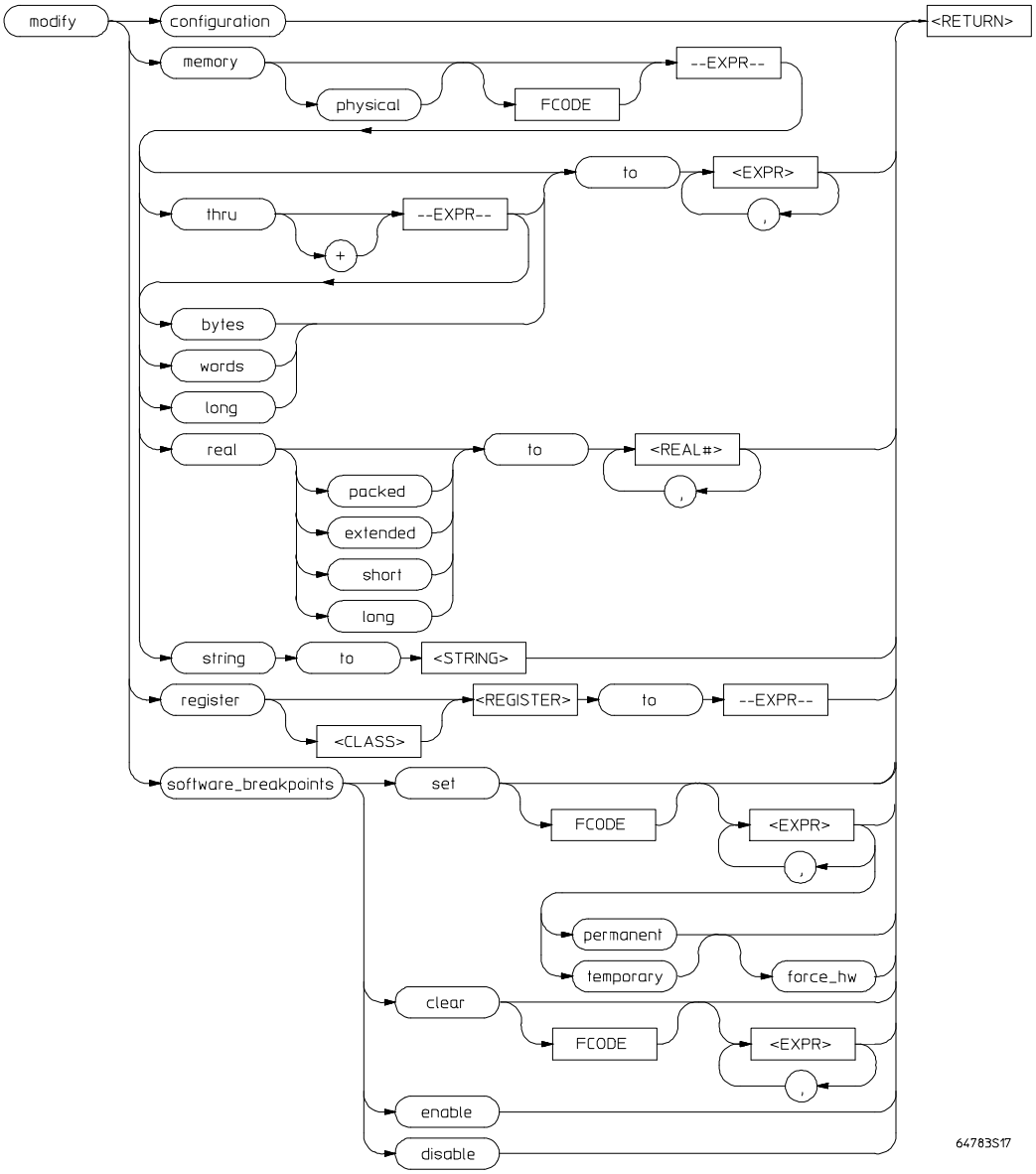
Examples

```
log_commands  to  logfile

log_commands  off
```

See Also

```
help  system_commands
```

modify


modify
configuration
<RETURN>
memory
--EXPR--
physical
FCODE
to
<EXPR>
,
thru
--EXPR--
+
bytes
words
long
real
to
<REAL#>
,
packed
extended
short
long
string
to
<STRING>
register
<REGISTER>
to
--EXPR--
<CLASS>
software_breakpoints
set
FCODE
<EXPR>
,
permanent
temporary
force_hw
clear
FCODE
<EXPR>
,
enable
disable
64783S17

The **modify** command is used to:

- View or edit the current emulation configuration.
- Modify the contents of memory (as integers, strings, or real numbers).
- Modify the contents of the processor registers.
- Write specified values to I/O port addresses.
- Modify the software breakpoints table.

bytes — Modify using byte values.

<CLASS> — This represents the name of a processor register class. The register classes are also displayed on the softkey labels. See the "register" option for more information.

clear — This option erases the specified breakpoint address and restores the original content of the memory location. (The location must not have changed (by loading a file or modifying memory) after the breakpoint was set.) If no breakpoints are specified in the command, all currently specified breakpoints are cleared and the memory locations are restored to their original values.

, — A comma is used as a delimiter between values when modifying multiple values.

configuration — 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 the carriage return key. Otherwise, you can modify the response as you desire, and then press the carriage return key.

For each emulator, default responses defined on powerup are displayed. For more information see Chapter 8, "Configuring the Emulator".

—EXPR— — An expression is a combination of numeric values, symbols, operators, and parentheses, specifying a value. See the **—EXPR—** syntax diagram.

extended — This option allows you to modify memory as 96-bit IEEE real numbers.

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.

force_hw — This option forces the emulator to use one of its eight hardware resources to store the breakpoint instead of writing it in software, if possible.

keyboard_to_simio

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** and the carriage return key will deactivate keyboard simulated I/O and return the keyboard to normal emulation mode. For details about setting up simulated I/O on your HP 9000 host computer system, refer to the *Simulated I/O User's Guide.*

long

Modify memory as long word values. When used with the **real** parameter, **long** specifies that memory be modified as a 64-bit IEEE-754 real number value.

memory

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.

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.

Memory modifications of integer values use the currently set display memory mode. Byte is the default.

Memory modifications of "real" values use the currently set display memory mode. Short is the default.

packed	This option allows you to modify memory as 96-bit Motorola-format packed real numbers.
permanent	This option allows you to set a software breakpoint that remains at the breakpoint address and causes a break each time it is hit during execution. It is not automatically replaced by the target-program instruction when it is hit. Permanent breakpoints can only be set if your version of HP 64700 system firmware is A.04.00 or greater.
physical	This allows you to specify that the addresses to be modified are addresses in physical memory.
real	Modify memory as real number values.
<REAL#>	This prompts you to enter a real number as the value.
register	The entry you specify for <REGISTER> determines which register is modified. Register modification cannot be performed during real-time operation of the emulation processor. A **break** command, or a condition that causes the emulator to break to the monitor, must occur before you can modify registers. Basic registers include: PC, STATUS, USP, ISP, MSP, CACR, D0 through D7, A0 through A7, VBR, DFC, and SFC. The FPU registers include: FPCR, FPSR, FPIAR, and FP0 through FP7. The MMU registers include: ITT0, ITT1, DTT0, DTT1, MMUSR, TC, SRP, and URP.
<REGISTER>	This represents the name of a register that you specify.
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.
short	Modify memory values as 32-bit IEEE-754 real numbers.
software_breakpoints	Software breakpoints allow you to stop execution of your target program and begin execution in the monitor when the breakpoint instruction is executed. Any valid address (number, label, or expression) may be specified as a breakpoint. Valid addresses identify the first byte of valid instructions. Operation of the program can be resumed after the breakpoint is encountered by specifying either a **run** or **step** command.

While the memory mnemonic display is on screen, execution breakpoints are indicated by an "*****" in the leftmost column of the instruction containing the breakpoint.

Do not modify execution breakpoints while your target program is running. If you do, program execution may be unpredictable.

You must enable breakpoints before you can perform an action on them.

When you set breakpoints, the emulator will search through the existing breakpoint list and reactivate all entries that are inactivated.

When you clear breakpoints, the entire execution breakpoint list is deleted and memory is restored to its original values.

disable — This option turns off the execution breakpoint capability.

enable — This option allows you to modify the execution breakpoint specification.

string — Modify memory values to the ASCII character string given by <STRING>.

<STRING> — A quoted ASCII string, which may include the following special characters:

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)

temporary — This option allows you to set an execution breakpoint that is in effect only the first time it is hit during execution. It is automatically replaced by the target-program instruction when it is hit. Temporary breakpoints (the default breakpoints) differ from permanent breakpoints which can only be set if your version of HP 64700 system firmware is A.04.00 or greater.

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 or register will be changed.

words — This lets you modify memory in word format.

Examples

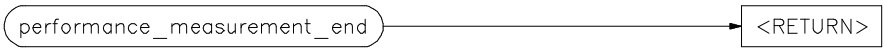
```
modify configuration

modify keyboard_to_simio

modify memory 00A0H words to 1234H

modify memory DATA1 bytes to 0E3H , 01H , 08H

modify memory DATA1 thru DATA100 to 0FFFFH

modify memory 0675H real to -1.303

modify memory TEMP real long to 0.5532E-8

modify memory buffer string to "This is a test \n\0"

display memory blocked bytes

modify memory Msg_Dest thru +50 to 41h, 42h, 43h

modify memory Msg_Dest string to "HP 64000 Softkey
Interface"

modify memory Msg_Dest thru +50 to 0

modify register D0 to 9H

modify register BASIC PC to 2000H

modify software_breakpoints enable

modify software_breakpoints clear 99H , 1234H

modify software_breakpoints set LOOP1 END ,
LOOP2END , 0EH
```

modify software_breakpoints set

See Also

copy memory
copy registers
copy software_breakpoints
display memory
display registers
display software_breakpoints
help ***modify***
help ***registers***
help ***software_breakpoints***
modify registers
store memory

performance_measurement_end



This command stores data previously generated by the **performance_measurement_run** command, in a file named “perf.out" in the current working directory.

The file named “perf.out” is overwritten each time this command is executed. This command does not alter current measurement data in the emulation system.

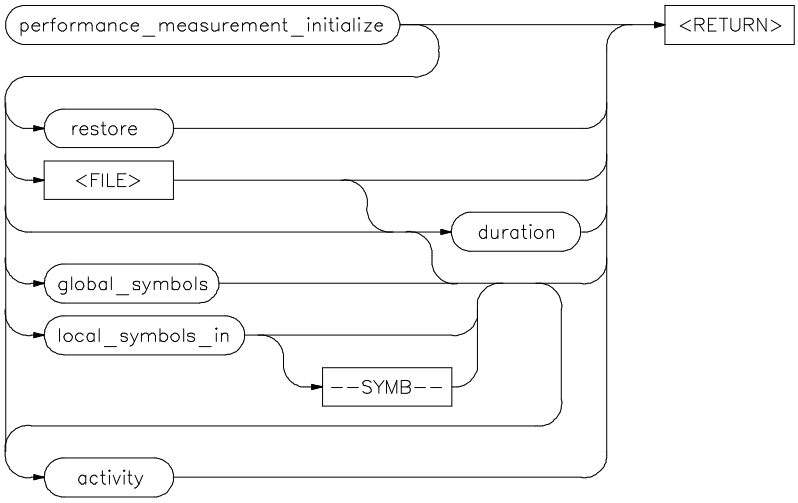
Example

```
performance_measurement_end
```

See Also

help ***performance_measurement_initialize***
help ***performance_measurement_run***
performance_measurement_initialize
performance_measurement_run

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.

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 uses local symbols as the default ranges for the measurement.
restore	This restores old measurement data so that the current measurement can be continued when using the **trace** command used previously.
—SYMB—	This represents the source file that contains the local symbols to be listed. This also can be a program symbol name, in which case all symbols that are local to a function or procedure are used. See the SYMB syntax diagram.

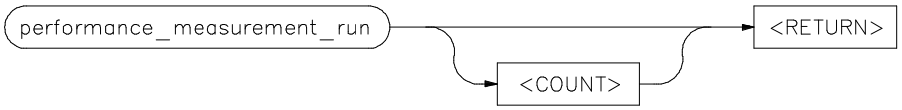
Examples

```
performance_measurement_initialize

performance_measurement_initialize  duration

performance_measurement_initialize  local_symbols_in
prog68k.S:
```

See Also

```
help  performance_measurement_initialize
help  performance_measurement_run
performance_measurement_run
performance_measurement_end
```

performance_measurement_run



This command begins a performance measurement. It causes the emulation system to reduce trace data contained in the emulation-bus analyzer. The resulting data is used for analysis by the performance measurement software.

The default is to process data presently contained in the analyzer.

<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.

The **trace** command must be set up correctly for the requested measurement. For an activity measurement, you can use the default **trace** command (**trace counting time**).

For a duration measurement, you must set up the trace specification to store only the points of interest. To do this, for example, you could enter:

trace only <symbol_entry> ***or*** <symbol_exit>

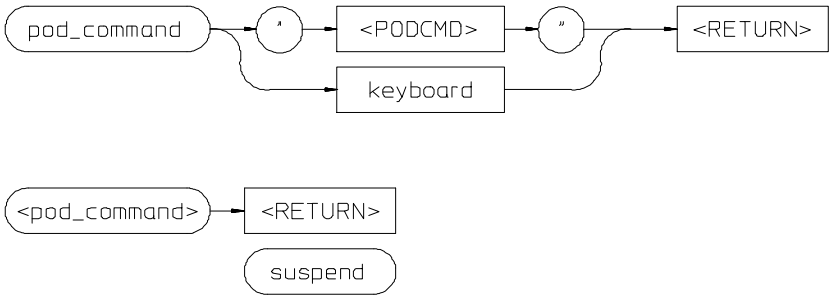
Examples

```
performance_measurement_run  10

performance_measurement_run
```

See Also

```
help  performance_measurement_initialize
help  performance_measurement_run
performance_measurement_end
performance_measurement_initialize
```

pod_command



The HP 64700 Series emulators contain a low-level Terminal Interface, which allows you to control the emulator's functions directly. You can issue Terminal Interface commands through the Softkey Interface by using **pod_command**. With **pod_command**, you can issue a single command, or enter the keyboard mode which allows you to enter a series of commands.

The *MC68040/EC040/LC040 Emulator/AnalyzerTerminal Interface User's Guide* is an excellent source of information about using the Terminal Interface to control the emulator. There are certain commands you should avoid while using the Terminal Interface through **pod_command**, however, because these commands can affect the operation of the Softkey Interface. See the User's Guide and the Terminal Interface screen for more information about Terminal Interface commands that should not be used.

Although you can issue Terminal Interface commands via this command, you cannot see the results of those commands unless you display the Terminal Interface screen with **display pod_command**.

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 the carriage return key.

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 Softkey Interface.

Examples

This example shows a simple interactive session with the Terminal Interface.

```
display pod_command
pod_command keyboard

cf

tsq

tcq
```

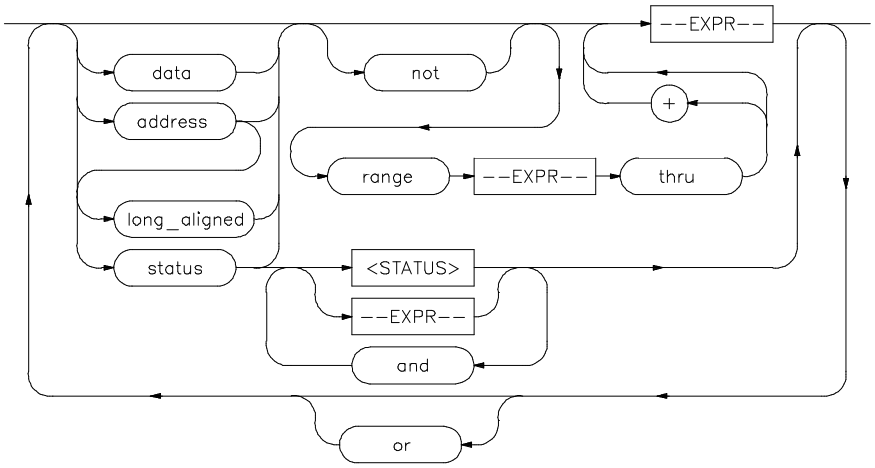
Enter **suspend** to return to the Softkey Interface.

See Also

```
display pod_command
help pod_command
```

Also see the *MC68040/EC040/LC040 Emulator/AnalyzerTerminal Interface User's Guide.*

QUALIFIER



The **QUALIFIER** parameter is used with **trace only**, **trace prestore**, **TRIGGER**, and **trace counting** to specify states captured during the trace measurement.

You may specify a range of states or specific states to be captured. You can specify a unique combination of address/data/status values as conditions for the trace measurement. 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 or followed (or both) by a number. Some examples include 1fxxh, 17x7o, and 011xxx10b. "Don't care numbers" may be entered in binary, octal, or hexadecimal base.

Expression types are "address" when none is chosen. The default is to qualify on all states.

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.

QUALIFIER

data	This specifies that the expression that follows is a data value on the emulation processor data bus.
—EXPR—	An expression is a combination of numeric values, symbols, operators, and parentheses, specifying an address, data, or status value. See the EXPR syntax diagram.
long_aligned	Causes a mask to be applied to the address to force it to a long word boundary (least significant hex digit is 0, 4, 8, or C). This is necessary because the emulation-bus analyzer may not otherwise see the address on the address bus due to the way the processor fetches instructions.
not	This specifies that the analyzer search for the logical “not” of the specified range or state, that is addresses not in the specified range or an address not in the specified state.
or	This option allows you to specify multiple states to be captured during a trace measurement.
range	This indicates a range of addresses to be specified (—EXPR— thru —EXPR—).
status	This specifies that the expression following, or status word, is a status value for the processor.
<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 include symbols, operators, and parentheses. See the EXPR syntax diagram. Refer to Chapter 5, "Using the Emulation-Bus Analyzer", for a list of predefined status equates that cover common processor operations.

thru — This indicates that the following address expression is the upper address in a range.

Examples

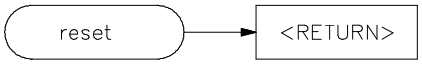
```
trace only address prog68k.S:READ_INPUT

trace only address range prog68k.S:READ_INPUT thru
OUTPUT

trace only address range prog68k.S:CLEAR thru
READ_INPUT
```

Also see the **trace** command examples.

See Also

```
help trace
trace
```

reset



This command suspends target system operation and asserts the reset signal on the emulation processor. The reset signal is latched when the **reset** command is executed, and is released by either the **run** or **break** command.

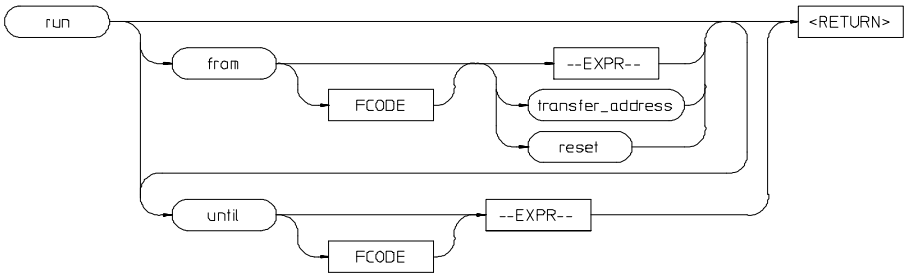
Example

```
reset
```

See Also

```
help reset
```

run



This command causes the emulator to execute a program. If the processor is in the reset state, **run** releases the reset condition. If you specify **run from —EXPR—** or **run from transfer_address**, the processor is directed to the particular address. If the processor is running in the emulation monitor or background memory, a **run** command causes the processor to exit into your program. The program can either run from a specified address (—EXPR—), from the address stored in the emulation processor program counter, or from a label specified in the program.

For an explanation of how the emulator runs from a reset condition (using the **run from reset** command), refer to the paragraph titled "To run a program" in Chapter 4, "Using the Emulator".

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.

—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 where program execution is to begin.

reset This option starts the processor executing from the reset vector address.

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.

until — Specifies an address where execution is to stop. The emulator will execute your target program up to the point where the **until** address is found. Execution will stop at the **until** address, and the emulator will begin executing the emulation monitor.

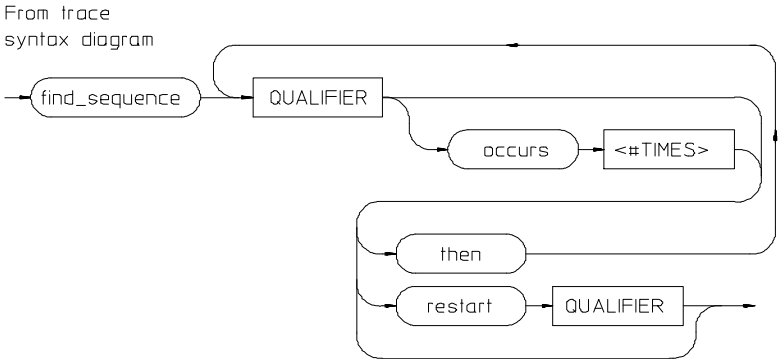
Examples

```
run

run  from  810H

run  from  COLD_START

run from TEST_START until TEST_POINT_1
```

See Also

```
help  run
help  step
step
```

SEQUENCING



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.

find_sequence — Specifies that you want to use the analysis sequencer. You must enter at least one qualifier.

QUALIFIER — Specifies the address, data, or status 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

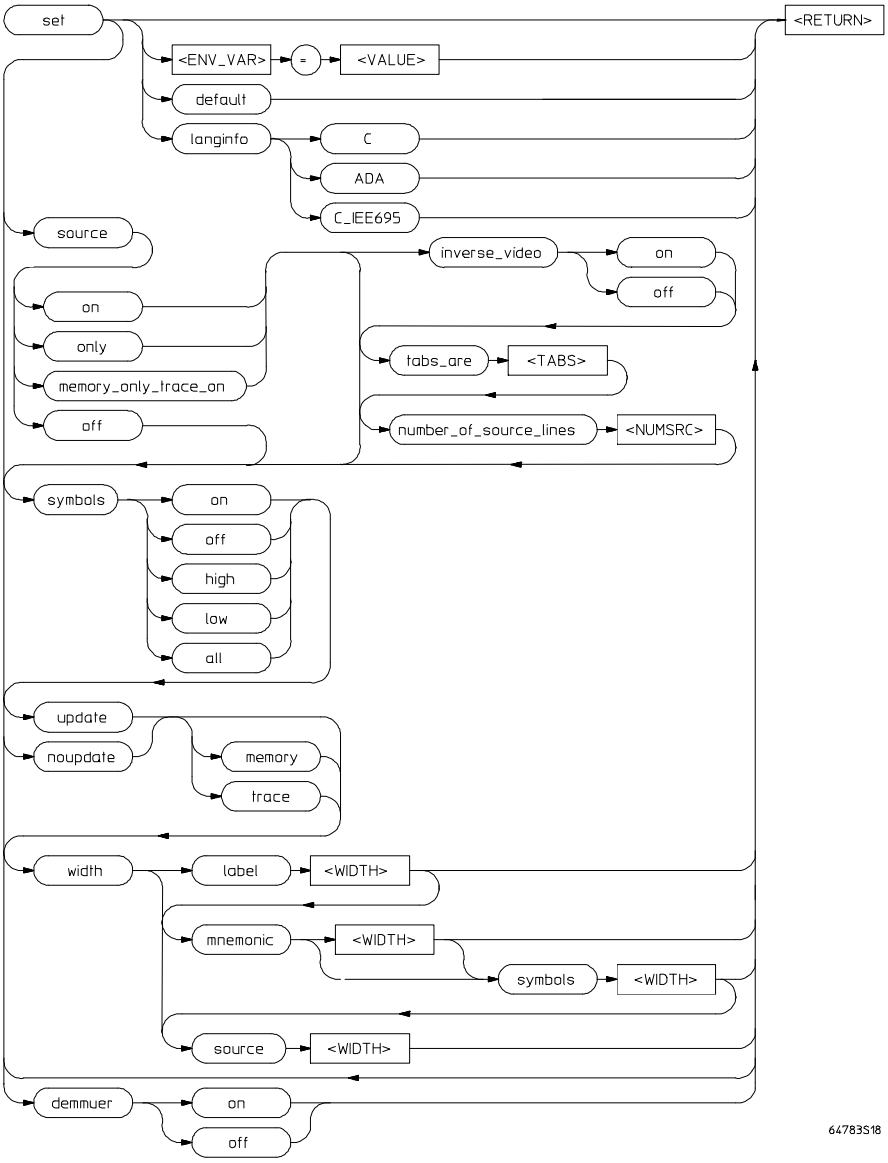
The following example uses symbols from an imaginary program that performs a series of tests in sequence. Occasionally test2 completes but does not start test3 (instead, it jumps directly to test 9). The following trace command would be used to capture a trace only when the program fails to step from test2 to test 3 so you could look at activity associated with this program failure:

```
display trace

trace find_sequence test1 then test2 restart test3
trigger about not range test3 thru test3 end
```

See Also

trace
QUALIFIER
WINDOW
help ***trace***

set


set
<RETURN>
<ENV_VAR>
=
<VALUE>
default
langinfo
C
ADA
C_IEE695
source
inverse_video
on
off
on
only
memory_only_trace_on
off
tabs_are
<TABS>
number_of_source_lines
<NUMSRC>
symbols
on
off
high
low
all
update
noupdate
memory
trace
width
label
<WIDTH>
mnemonic
<WIDTH>
symbols
<WIDTH>
source
<WIDTH>
demmuer
on
off
64783S18

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 modified 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.

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 simply using the command:

```
set default
```

default — This option restores all the set options to their default settings.

demmuer

- on — This option turns on the deMMUer. Addresses on the emulation bus will be reverse-translated (physical to logical) before being supplied to the analyzer. Reverse translations will be made according to the setup that was present in the MMU at the time you entered your last **load demmuer** command.
- off — This option turns off the deMMUer. Addresses on the emulation bus will be supplied directly to the analyzer without translation.

<ENV_VAR> — Specifies the name of an environment variable to be set within the HP 64000-UX environment or the system environment.

= — 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

Specifies the type of display to be updated or not updated.

noupdate

This option stops the display buffer in a window or terminal from updating when a new measurement completes. Without this option, displays that show memory contents are updated when a command executes that changes the values in memory (such as modify memory or load).

number_of_source_lines

This allows you to specify the number of source lines to be displayed above the program instructions or trace data to which they correlate. Displays may contain several blocks of source lines, with each block followed by program code or emulation-bus activity. Each block of source lines can begin with the next source line following the preceding block. If a particular block contains a lot of comment lines, the block can become very big. This option lets you specify a number to limit the size of the blocks of source lines. The default value is 5.

<NUMSRC>

This prompts you for the number of source lines to be displayed. Enter a value from 1 through 50.

source

memory_only_trace_on

This provides a way to default the memory and trace displays to a setting that HP believes is the nicest possible formats for memory and trace displays. Parameters such as "source on/only", number of source lines to show, display width, and turning symbols on are all governed by this one selection. With this selection, memory displays will show the maximum available source lines preceding each block of code, and trace lists will show five source lines preceding trace data.

off

This option prevents inclusion of source lines in the trace and memory mnemonic display lists.

on

This option displays source program lines preceding the processor instructions to which they correlate. This enables you to correlate processor instructions with source program code. The option is available in 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	Prevents symbol display.
on	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 assembler.
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. Enter values in the range of 2 through 15.

trace — Specifies the type of display to be updated or not updated.

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 or HP 64000-UX system 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 or mnemonic fields, or both 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. <CTRL>f and <CTRL>g may be used to shift the display left or right to display information which is off the screen.

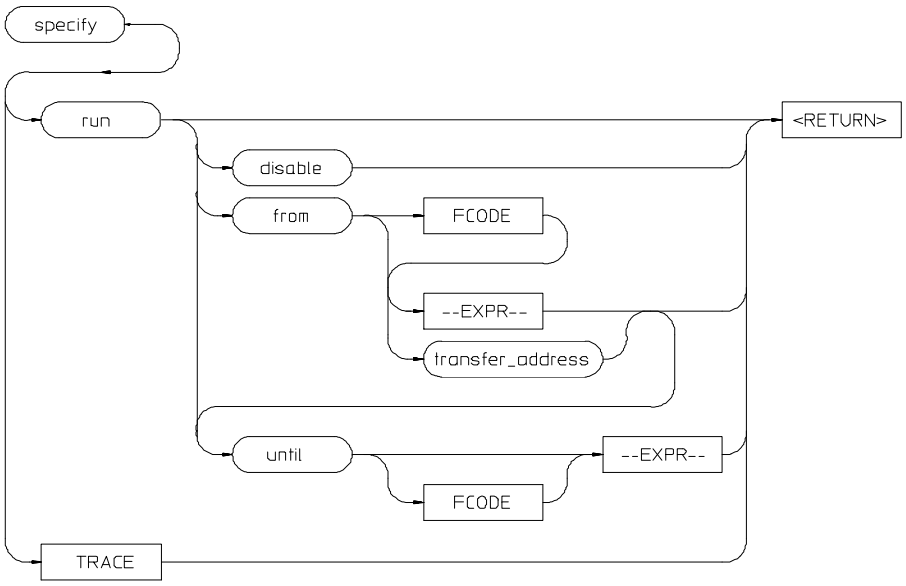
Examples

```
set noupdate

set source on inverse_video on tabs_are 2

set symbols on width label 30 mnemonic 20

set PRINTER = “lp -s”

set HP64KSYMBPATH=".file1:proc1
.file2:proc2:code_block_1"
```

See Also

display data
display memory
display software_breakpoints
display trace

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 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.

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, unless otherwise directed.

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 stop execution of your program when it reaches this address and enter the monitor.

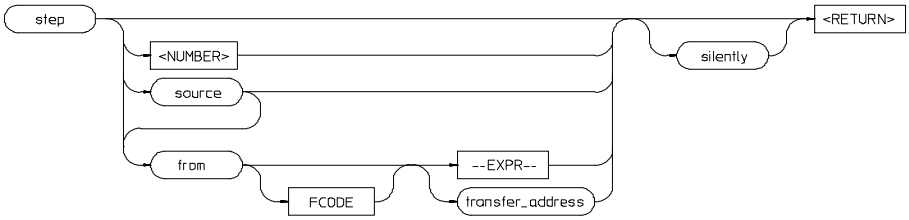
Examples

```
specify run from START

specify trace after address 1234H
```

See Also

cmb_execute
help ***specify***

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 through 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 indefinitely. To abort stepping, type <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 (see the discussion on opening and using other interface windows in Chapter 3, "Using the Emulator/Analyzer Interface").

If no value is entered for <NUMBER> times, only one **step** instruction is executed each time you press the carriage return key. Multiple instructions can be executed by holding down the carriage 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.

—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	This option updates the register step mnemonic only after stepping is complete. This will speed up instruction stepping. The default is to update the register step mnemonic after each assembly instruction (or source line) executes (if stepping is performed in the same window as the register display).
transfer_address	This represents the starting address of the program you loaded into emulation or target memory. The **transfer_address** is defined in the linker map.
source	This option performs stepping on source lines.

Examples

```
step

step  from  810H

step  20  from  0A0H

step 5 source

step 20 silently

step 4 from main
```

See Also

```
help  step
display registers
display memory mnemonic
set symbols
```

stop_trace

This command terminates the current trace and stops execution of the current measurement. The analyzer stops searching for trigger and trace states. If trace memory is empty (no states acquired), nothing will be displayed.

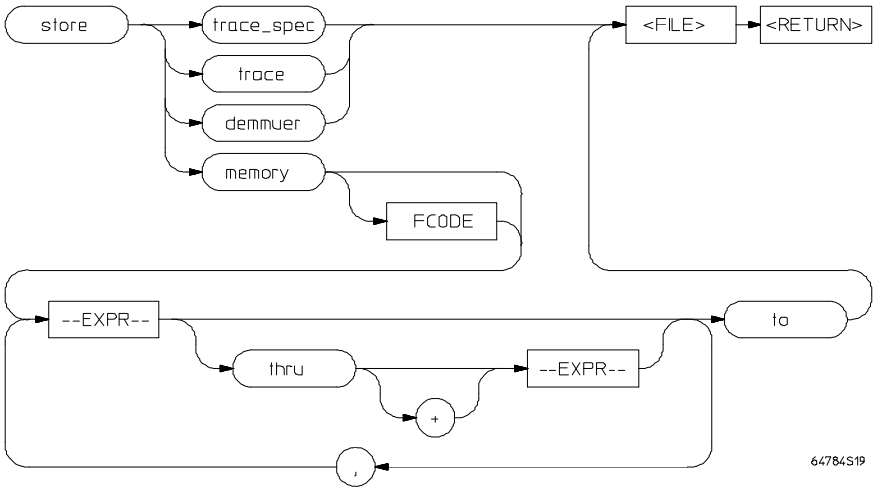
Example

```
stop_trace
```

See Also

```
help  stop_trace
trace
```

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. A new file is created 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.

demmuer — This causes the emulator to read the content of the MMU tables and store the appropriate demmuer setup in a file you name, with a **.ED** extension.

—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 file with a **.X** extension.
thru	This allows you to specify that ranges of memory be stored.
to	Use this in the **store memory** command to separate memory locations from the file identifier.
trace	This option causes the current trace data to be stored in the specified file with a **.TR** extension.
trace_spec	This option stores the current trace specification in the specified file with a **.TS** extension.
,	A comma separates memory expressions in the command line.

Examples

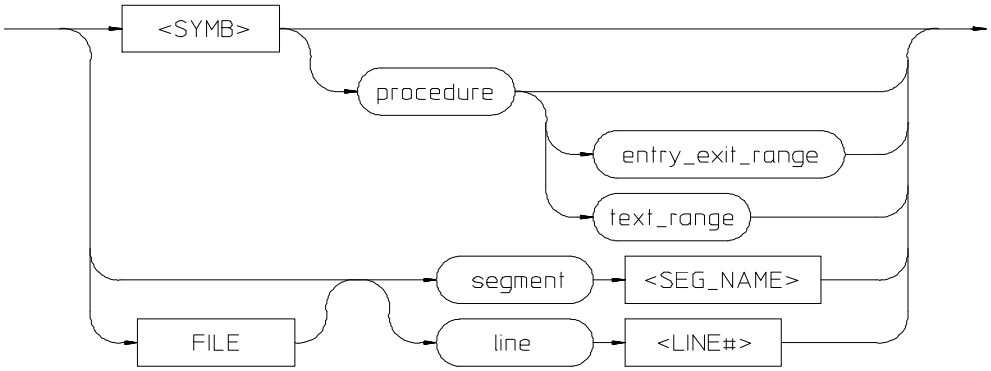
```
store memory 800H thru 20FFH to TEMP2

store memory EXEC thru DONE to \12.10

store trace TRACE

store trace_spec TRACE

store demmuer MMUTEST1
```

See Also

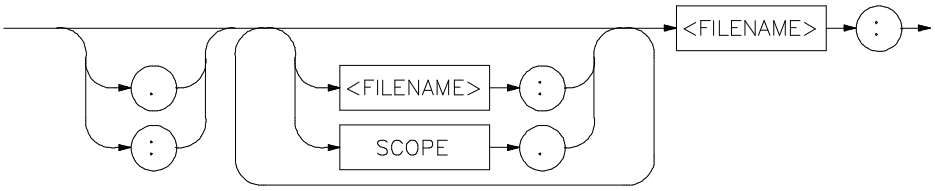
```
display memory
display trace
help store
load
```

—SYMB—

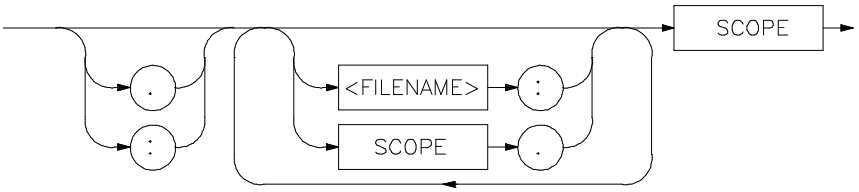
--SYMB--



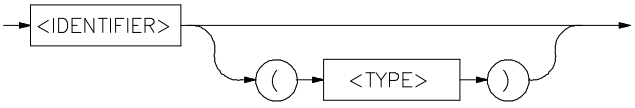
FILE



<SYMB>



SCOPE



This parameter is a symbolic reference to an address, address range, file, or other value. Symbols may be:

- Combinations of paths, filenames, and identifiers defining a scope, or referencing a particular identifier or location (including procedure entry and exit points).
- Combinations of paths, filenames, and line numbers referencing a particular source line.
- Combinations of paths, filenames, and segment identifiers identifying a particular PROG, DATA or COMN segment or a user-defined segment.

The Symbolic Retrieval Utilities (SRU) handle symbol scoping and referencing. These utilities build trees to identify unique symbol scopes.

If you use the SRU utilities to build a symbol database before entering the emulation environment, the measurements involving a particular symbol request will occur immediately. If you then change a module and re-enter the emulation environment without rebuilding the symbol database, the emulation software rebuilds the changed portions of the database in increments as necessary.

Further information regarding the SRU and symbol handling is available in the *SRU 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.

The **display memory mnemonic** command also can modify the current working symbol.

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.

entry_exit_range — The range of addresses beginning with the entry point and ending with the return instruction. The entry point is the address used by other files when they call this procedure.

<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> This 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.

text_range The range of addresses beginning with the lowest address occupied by any code in the procedure and ending with the highest address occupied by any code in the procedure. Normally, the text_range will be the same as the entry_exit_range; some compilers may rearrange code so that the return instruction (for example) is not at the highest address in the range occupied by code of the procedure.

(<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.
- fsegment This provides an alternate way to reference a file segment in a command (example: **myfile.c:PROG(fsegment**)). It is better to use the keyword **segment** (example: **myfile.c: segment PROG**). The "segment" keyword is preferred because it will do scanning for PROG, Prog, prog, and other expressions of the program segment in the example line. The **fsegment** keyword will only scan for the one expression (PROG).
- module These refer to module symbols. For most C compilers, these names derive from the source file name. 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.

prospecial — Prospecial symbols are symbols that were created by the HP SRU (such as entry, exit, and return). They are derived symbols, not intended for the product user. Typical prospecial symbols would be entry1, entry2, and entry3 in a procedure that has three possible entry points.

static — Static symbols, which include 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 delimit 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 C code example is provided to help illustrate how symbols are maintained by SRU and referenced in your emulation commands.

File /users/dave/control.c contains:

```
int *port_one;

main()
{
    int port_value;

    port_one = 255;
    port_value = 10;
    process_port (port_one, port_value);
} /* end main */
```

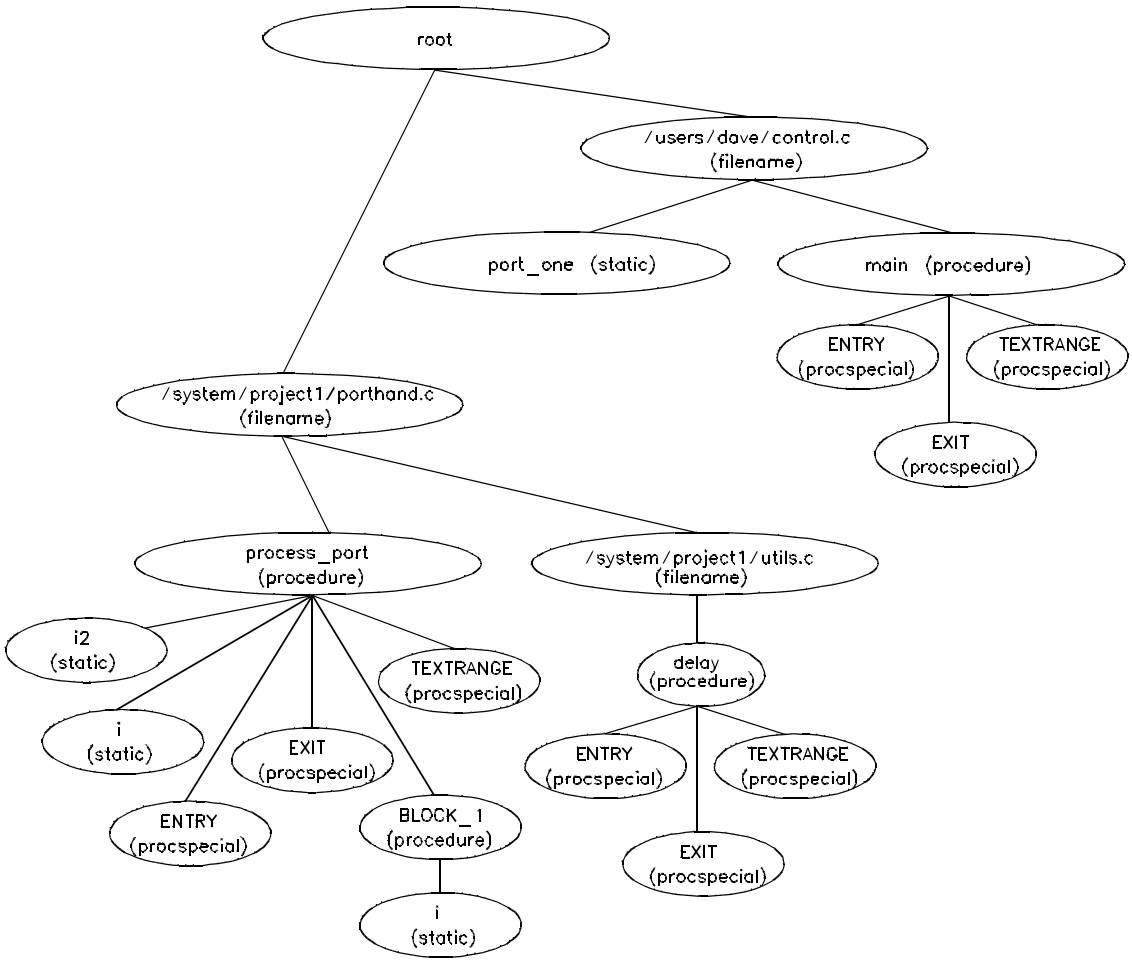
File /users/project1/porthand.c contains:

```
#include “utils.c”

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 /users/project1/utils.c contains:

```
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 shown in the following diagram, depending on the object module format (OMF) 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.

These are examples of referencing different symbols in the programs listed:

```
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)
```

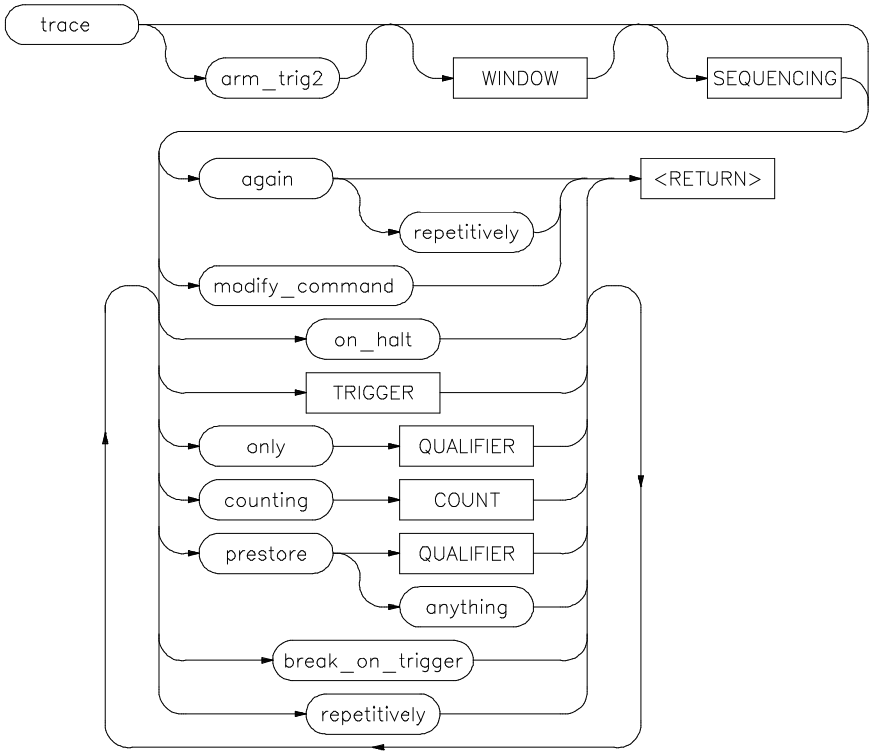
to reference the procedure address.

See Also

copy local_symbols_in
cws
display local_symbols_in
help ***symbols***
pws

Also refer to the *Symbolic Retrieval Utilities User's Guide* for additional information on symbols and information on building a symbol data base.

trace



The options shown can be executed once for each **trace** command. Refer to the COUNT, QUALIFIER, SEQUENCING, TRIGGER, and WINDOW 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. The analyzer will trace any state, counting time by default.

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 PAGE**, **PREV PAGE**, **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. When a **cmb_execute** command is given, which puts an EXECUTE signal on the Coordinated Measurement Bus, analyzers will begin tracing.

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 64700 Series emulators, or an HP 64700 Series emulator 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 6, "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.

COUNT — This specifies whether time or state occurrences, or nothing, will be counted during the trace. See the COUNT syntax diagram for details.

counting — This option specifies whether the analyzer will count time or occurrences of states during a trace, or whether the option is to be turned off.

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. You may also use it to further qualify complex trigger specifications. See the **WINDOW** syntax pages for details.

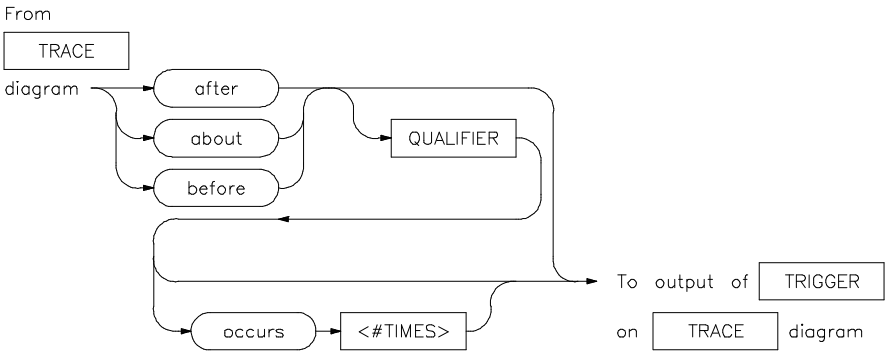
Examples

```
trace after 1000H

trace only address range 1000H thru 1004H

trace counting state address 1004H

trace after address 1000H occurs 2 only address
range 1000H thru 1004H counting time
break_on_trigger
```

See Also

copy trace
display trace
help ***trace***
load trace
load trace_spec
specify trace
store trace
store trace_spec

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.

about — This option captures trace data leading to and following the trigger qualifier. The trigger is centered in the trace listing.

after — Trace data is acquired after the trigger qualifier is found.

before — Trace data is acquired prior to the trigger qualifier.

occurs — This specifies a number of qualifier occurrences of a range or state on which the analyzer is to trigger.

QUALIFIER — This determines which of the traced states will be stored in trace memory.

<#TIMES> — This prompts you to enter a number of qualifier occurrences.

Examples

```
trace after MAIN

trace after 1000H then data 5
```

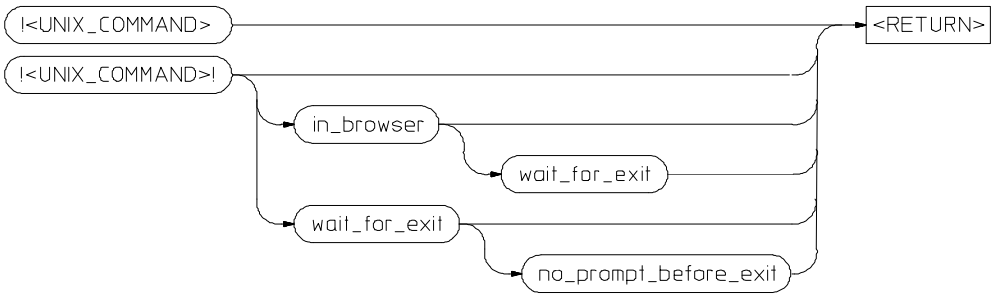
Also see the **trace** command examples.

See Also

```
help trace
trace
```

Also refer to Chapter 6, "Coordinated Measurements".

<UNIX_COMMAND>



This structure allows you to execute UNIX commands on the emulator/analyzer command line. The UNIX commands and command-line options are interpreted as noops in the emulator/analyzer interface.

in_browser — Places the resulting display in a scrollable box instead of an XTERM window.

wait_for_exit — Use this command to ensure completion of the associated UNIX command before starting the next command.

no_prompt_before_exit — When the associated UNIX command completes, the results display is shown and then removed from the display without need to press the RETURN key. Use this option to speed execution of your command when the results display is not important to you.

Examples

To see your present working directory:

```
!pwd
```

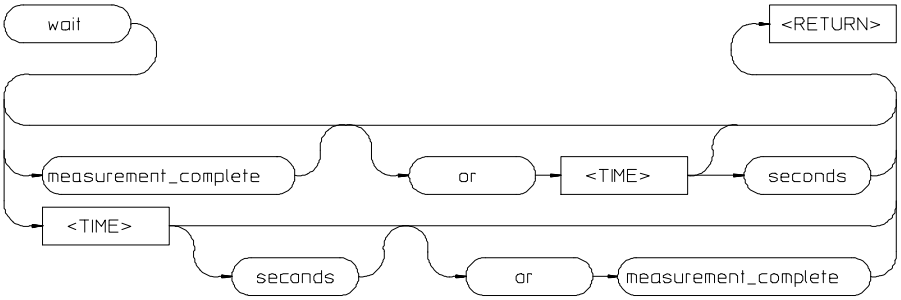
To see a directory listing in a browser instead of terminal window:

```
!ls! in_browser
```

To make and load an executable:

```
!make <file>! wait_for_exit; load <executable_file>
```

wait



This command allows you to pause the system. The **wait** command can be included in a command file, or used during normal operation at the main emulation level. Delays allow the emulation system and target processor 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.

If you issue a **wait** command without any other options, the system will pause until it receives a **<CTRL>c** signal.

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.

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.

measurement_ complete	This causes the system to pause until a pending measurement completes (for example, when 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. The wait will end when the first one of these two conditions occurs.
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.

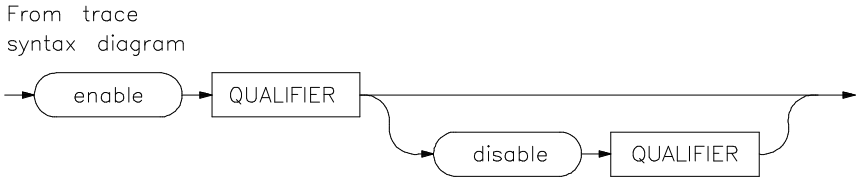
Examples

```
wait

wait 5; wait measurement_complete
```

See Also

```
help system_commands
help wait
```

WINDOW



WINDOW allows you to selectively toggle analyzer operation. When enabled, the analyzer will recognize sequence terms and 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.

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

The following example uses an imaginary program that writes messages. The window specification ensures that the trace memory will only store message writes (store write transactions only during the time when the Write_Msg routine is active), and not store other program activity.

```
display trace

trace enable Write_Msg start disable Write_Msg end only
status write
```

See Also

```
help trace
SEQUENCING
trace
QUALIFIER
```

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

<MMU_REGISTER> register cannot be read

Where <MMU_REGISTER> = TC, SRP, URP, ITT0 ITT1, DTT0, or DTT1.

Cause: An MMU register cannot be read, probably because the monitor is not functioning properly.

Action: Execute a Break command to see if the monitor is functioning.

Address translation error; non-resident page: <address> (Error 172)

Cause: The error occurred when the monitor attempted to access memory with the MMU enabled. You requested the monitor to display or modify memory, or you tried to exit the monitor; the memory access generated an access fault resulting from an MMU address translation failure. This error indicates that the address does not have a valid translation.

Action: Display the address translation tables for the <address> given in the message. You can display the MMU translations to see if the <address> is within one of the translated ranges. You can display translation tables for the address, and then you can view table details if one of the translation tables seems to be misdirecting the translation of the address.

Address translation error; supervisor-only page: <address> (Error 172)

Cause: The error occurred when the monitor attempted to access memory with the MMU enabled. You requested the monitor to display or modify memory, or you tried to exit the monitor and the memory access generated an access fault resulting from an MMU address translation failure. A user mode access was attempted to a page that is only accessible in supervisor mode.

Action: Try your command again, but be sure to specify access in the supervisor mode.

Address translation error; target bus error: <address> (Error 172)

Cause: The error occurred when the monitor attempted to access memory with the MMU enabled. You requested the monitor to display or modify memory, or you tried to exit the monitor; the memory access generated an access fault resulting from an MMU address translation failure. The target system terminated a tablewalk cycle with TEA (bus error).

Action: Verify that the SRP and URP registers point to the correct location in memory where your address translation tables reside. If this is target memory, you will need to determine why your target system asserts TEA.

Address translation error; write-protected page: <address> (Error 172)

Cause: The error occurred when the monitor attempted to access memory with the MMU enabled. You requested the monitor to display or modify memory, or you tried to exit the monitor and the memory access generated an access fault resulting from an MMU address translation failure.

This error indicates that write access was denied to a write-protected page. NOTE: Except for stacking on exit, any attempts to modify memory in write-protected pages using the monitor will succeed as long as the translation tables reside in RAM. The monitor will temporarily clear any write-protect flags in your translation tables in order to force the access to be completed. If the monitor is unable to clear the write-protect flags because the translation tables are in ROM, you will see this error.

Action: Check the content of the write-protected page to see if it has been changed by the attempted write transaction.

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.

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.

Cannot enable mmu/cache while background monitor is selected (Error 158)

Cause: You tried to enable either the MMU or the cache within the emulation configuration after selecting the background monitor. The background monitor requires the MMU and the cache to be disabled in order to operate properly.

Action: Use the foreground monitor if you want to enable either the MMU or the cache.

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 5, "Using the 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.

Continuing with default foreground monitor (Error 144)

Cause: You have downloaded a custom foreground monitor which was linked at an address other than the monitor address specified within the emulation configuration.

Action: Change the monitor address within the emulation configuration or link your custom monitor at the address specified in the configuration.

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 supported (Error 175)

Cause: The **cim** (copy 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 4, "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 5, "Using the Analyzer", for more information.

Coverage not supported (Error 175)

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

DeMMUer has not been loaded (Error 163)

Cause: You tried to enable the deMMUer before it had been loaded. The deMMUer can only be enabled after it has been loaded with a set of reverse translation information.

Action: Load the deMMUer from the present translation tables in memory or from a deMMUer file that you have previously saved.

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.

Disabled deMMUer

Cause: This is a status message. It indicates that the deMMUer has now been disabled. Physical memory addresses will be provided to the emulation-bus analyzer. The analyzer will not be able to accept commands containing source-file symbols, and it will not be able to show source-file symbols in its trace lists.

Disabled mmu/cache while background monitor is selected (Status 157)

Cause: This status message indicates that the MMU or the cache or both were enabled in the emulation configuration when you changed the monitor type to background. The background monitor requires the MMU and the cache to be disabled in order to operate properly. Both the MMU and the cache were disabled automatically when you changed to the background monitor.

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 <number of lines> lines (Status 162)

Cause: This status message indicates that more lines of MMU translations could have been displayed, but when you requested a display of MMU translations, you limited the number of lines to be displayed.

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.

Downloaded monitor spans multiple 4K byte block boundaries (Error 145)

Cause: You tried to load a custom foreground monitor, but the absolute file has address records that are outside the range of a single 4-Kbyte block.

Action: Modify your custom monitor so that its code and data fit into a single 4-Kbyte block; then assemble, link, and repeat the load operation.

Dual ported memory already in use (Error 142)

Cause: There is only one 4-Kbyte block of dual-port, emulation memory available for mapping and you tried to map another term using the dual-port attribute. If you select the foreground monitor, this block is used by the monitor and is not available for mapping.

Action: Reenter the map command and answer "No" to the Dual Port Memory attribute, or select a background monitor and reenter your map command.

Dual ported memory limited to 4K bytes (Error 141)

Cause: There are only 4 Kbytes of dual-port emulation memory on the emulator probe. You tried to map an emulation memory term whose address range spanned more than 4 Kbytes by answering "Yes" to the "Dual Port Memory" attribute selection.

Action: You can:

- Reenter the specification, answering "Yes" to the "Dual Port Memory" attribute question. Be sure to restrict the address range to 4 Kbytes.
- Reenter your specification, and use regular emulation memory. That is, answer "No" to the "Dual Port Memory" attribute question.

Enabled deMMUer

Cause: This is a status message. It indicates that the deMMUer has now been enabled to provide reverse translation information to the emulation-bus analyzer. The analyzer will be able to accept commands containing source-file symbols, and it will be able to show source-file symbols in its trace lists.

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 terminated hung bus cycle: <address> byte read (Status 170)

Cause: This status message will be displayed if the target system fails to provide TA or TEA bus cycle termination for a particular cycle and the emulator terminates the bus cycle in order to break from execution of the target program to execution within the monitor, or to complete execution of a monitor command (which accessed this memory address). This can happen on any access to target memory or interlocked emulation memory (when you answered "Yes" to the "Emulator Terminates Bus Cycles" attribute question).

The emulator will not terminate any hung bus cycles unless you explicitly say break or you execute a monitor command (ie: "display memory 7000"). The emulator will generate this status message each time it terminates a hung bus cycle. The emulator never attempts to terminate bus cycles in program space (opcode fetches) or for any addresses in the foreground monitor.

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 chapter titled "Interfaces of the Deep Analyzer" for correct formats for the **tgout** command.

Failed to disable step mode (Error 684)

Cause: System failure. 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.

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. When the MMU is enabled, the address displayed in this message will be physical, as denoted by the trailing "a" after the function code.

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.

Hardware breakpoints can only be used in target memory (Error 154)

Cause: You attempted to use the "force hardware" option to set a breakpoint at an address mapped as emulation memory. The "force hardware" option for breakpoints is not available for addresses in emulation memory; it is only available for breakpoints in target memory, typically for setting breakpoints in target ROM.

Action: Delete the "force hardware" option from your command and try to set the breakpoint again.

HP64783 M68040 firmware not compatible with emulation probe (Status 179)

Cause: The emulation control card is programmed with MC68040 firmware, but the firmware does not identify the probe as being the MC68040.

Action: Make sure that you are using an MC68040 probe, and then make sure the probe cables between the control card and the probe are connected correctly. Refer to Chapter 19, "Installation and Service", for proper cable connections.

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.

Interrupt stack is located in guarded memory: <address> (Error 151)

Cause: You issued a command to run the target program, but when the emulator attempted to write to one of your stacks, it detected that the stack address is in memory mapped as guarded.

The monitor exits to user program by executing an RTE instruction. Depending upon whether or not you set the M bit in the SR, the monitor will either place a format $0 stack frame on the interrupt stack or will place a format $1 (throwaway) stack frame on the interrupt stack and a format $0 stack frame on the master stack. Any access violations detected during these writes will abort the exit from the monitor.

Action: Use the **display registers** and **modify registers** commands to set the stack pointer to an even value that points at a memory region (emulation or target RAM) that can be used for stack operations before running your program. Or, you can modify the emulation configuration and respecify the memory map to RAM for the address range containing the interrupt stack, the master stack, or both.

Interrupt stack is located in ROM: <address> (Error 151)

Cause: You issued a command to run the target program, but when the emulator attempted to write to one of your stacks, it detected that the stack address is in memory mapped as ROM, and you enabled breaks on writes to ROM.

The monitor exits your target program by executing an RTE instruction. Depending upon whether or not you set the M bit in the SR, the monitor will either place a format $0 stack frame on the interrupt stack or will place a format $1 (throwaway) stack frame on the interrupt stack and a format $0 stack frame on the master stack. Any access violations detected during these writes will abort the exit from the monitor.

Action: Use the **display registers** and **modify registers** commands to set the stack pointer to an even value that points at a memory region (emulation or target RAM) that can be used for stack operations before running your program. Or, you can modify the emulation configuration and respecify the memory map to RAM for the address range containing the interrupt stack, the master stack, or both.

Interrupt stack pointer is odd or uninitialized (Error 151)

Cause: You are in the monitor and you tried to run, but the emulator detected that your stack pointer is invalid (it detected an odd value).

Action: Use the **display registers** and **modify registers** commands to set the stack pointer to an even value that points at a memory region (emulation or target RAM) that can be used for stack operations before running your program.

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.

Interrupt stack is not located in RAM: <address> (Error 151)

Cause: You issued a command to run the target program. When the emulator attempted to write to one of your stacks, it detected that the stack address is not located in memory which operates as RAM. When the monitor writes out a stack frame to your stack space, the monitor reads it back to verify that it was created correctly. Unless the emulator can verify that the stack frame is located in RAM and was created correctly, the monitor will abort the run.

The monitor exits the target program by executing an RTE instruction. Depending upon whether or not you set the M bit in the SR, the monitor will either place a format $0 stack frame on the interrupt stack or will place a format $1 (throwaway) stack frame on the interrupt stack and a format $0 stack frame on the master stack. Any access violations detected during these writes will abort the exit from the monitor.

Action: Use the **display registers** and **modify registers** commands to set the stack pointer to an even value that points at a memory region (emulation or target RAM) that can be used for stack operations before running your program. Or, you can modify the emulation configuration and respecify the memory map to RAM for the address range containing the interrupt stack, the master stack, or both.

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: The dual-port memory and "Emulator Terminates Bus Cycles" attributes are valid only for emulation ROM and emulation RAM memory types. You tried to assign one of these attributes to target memory.

Action: Refer to Chapter 3, "Using the Emulator/Analyzer Interface", for information on the memory type attributes.

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. For example, because the MC68040 emulator does not support an internal clock, you would see this message if you entered a command to specify an internal clock configuration item for your emulator.

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 8, "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 entry at line <NUM> in file: <FILE> (Error 10372)

Cause: The data in the named <FILE> being loaded into the deMMUer is not in the correct format.

Action: Edit the file <FILE> and correct the syntax error or create a new file with a Store DeMMUer to file 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 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 MC68040 emulator are Transfer Cache Inhibit, Emulator Terminates Bus Cycles, and Dual Port Memory.

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 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 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 5, "Using the 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.

Loaded deMMUer file: <FILE>

Cause: This is a status message. It indicates that the deMMUer was successfully loaded from a configuration file.

Loaded deMMUer from MMU translation tables

Cause: This is a status message. It indicates that the deMMUer was successfully loaded from the MMU translation tables in memory.

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.

Master stack is located in guarded memory: <address> (Error 151)

Cause: You issued a command to run the target program, but when the emulator attempted to write to one of your stacks, it detected that the stack address is in memory mapped as guarded.

The monitor exits to user program by executing an RTE instruction. Depending upon whether or not you set the M bit in the SR, the monitor will either place a format $0 stack frame on the interrupt stack or will place a format $1 (throwaway) stack frame on the interrupt stack and a format $0 stack frame on the master stack. Any access violations detected during these writes will abort the exit from the monitor.

Action: Use the **display registers** and **modify registers** commands to set the stack pointer to an even value that points at a memory region (emulation or target RAM) that can be used for stack operations before running your program. Or, you can modify the emulation configuration and respecify the memory map to RAM for the address range containing the interrupt stack, the master stack, or both.

Master stack is located in ROM: <address> (Error 151)

Cause: You issued a command to run the target program, but when the emulator attempted to write to one of your stacks, it detected that the stack address is in memory mapped as ROM, and you enabled breaks on writes to ROM.

The monitor exits your target program by executing an RTE instruction. Depending upon whether or not you set the M bit in the SR, the monitor will either place a format $0 stack frame on the interrupt stack or will place a format $1 (throwaway) stack frame on the interrupt stack and a format $0 stack frame on the master stack. Any access violations detected during these writes will abort the exit from the monitor.

Action: Use the **display registers** and **modify registers** commands to set the stack pointer to an even value that points at a memory region (emulation or target RAM) that can be used for stack operations before running your program. Or, you can modify the emulation configuration and respecify the memory map to RAM for the address range containing the interrupt stack, the master stack, or both.

Master stack is not located in RAM: <address> (Error 151)

Cause: You issued a command to run the target program. When the emulator attempted to write to one of your stacks, it detected that the stack address is not located in memory which operates as RAM. When the monitor writes out a stack frame to your stack space, the monitor reads it back to verify that it was created correctly. Unless the emulator can verify that the stack frame is located in RAM and was created correctly, the monitor will abort the run.

The monitor exits the target program by executing an RTE instruction. Depending upon whether or not you set the M bit in the SR, the monitor will either place a format $0 stack frame on the interrupt stack or will place a format $1 (throwaway) stack frame on the interrupt stack and a format $0 stack frame on the master stack. Any access violations detected during these writes will abort the exit from the monitor.

Action: Use the **display registers** and **modify registers** commands to set the stack pointer to an even value that points at a memory region (emulation or target RAM) that can be used for stack operations before running your program. Or, you can modify the emulation configuration and respecify the memory map to RAM for the address range containing the interrupt stack, the master stack, or both.

Master stack pointer is odd or uninitialized (Error 151)

Cause: You are in the monitor and you tried to run, but the emulator detected that your stack pointer is invalid (it detected an odd value).

Action: Use the **display registers** and **modify registers** commands to set the stack pointer to an even value that points at a memory region (emulation or target RAM) that can be used for stack operations before running your program.

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.

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.

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 Commands", in this manual for further information on required syntax.

MMU is not enabled via configuration (Error 160)

Cause: You tried to display MMU translations or load the deMMUer but the MMU is disabled within the emulation configuration.

Action: If you wish to use the MMU, enable it in the emulation configuration before attempting to display its translations or load the deMMUer.

MMU is not enabled via translation control register (Error 160)

Cause: You tried to display the MMU translations or load the deMMUer. While the MMU is enabled within the emulation configuration, the enable bit is not set in the translation control register.

Action: Either enable the MMU in your target system by modifying the TC register, or specify an enabled value for the TC register on the command line when invoking the MMU or deMMUer commands.

Monitor address is not set to <addr> for downloaded monitor (Error 144)

Cause: You have downloaded a custom foreground monitor which was linked at an address other than the monitor address specified within the emulation configuration.

Action: Change the monitor address within the emulation configuration or link your custom monitor at the address specified in the configuration.

Monitor operation interrupted by target system (Error 173)

Cause: Your attempt to execute a monitor command was aborted when the target system preempted the monitor and did not return control. When the foreground monitor is running and is in its idle state, the monitor can be interrupted by the target system to service target system requirements. If the target system interrupts the monitor and fails to return control to the monitor after it has finished, this error is generated. The emulator does not attempt to regain control when after the monitor has been preempted.

Action: The only way to regain control of your emulation system is to reset the emulation processor. If you do not want the monitor to be preemptable by target system interrupts, you can increase the monitor interrupt priority level. Refer to Chapter 8, "Configuring the Emulator."

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 MC68040 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 MMU translations display exists

Cause: There has been no **Display MMU Translations** command prior to attempting to **Copy MMU Translations** to a file.

Action: Perform a **Display MMU Translations** command first. Then try your **Copy MMU Translations** command again.

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.

No monitor configured (Error 174)

Cause: You configured monitor "none" and you tried to break into the monitor or execute a command that requires use of the monitor.

Action: Either change the configuration to use a monitor, or do not try to issue a command that requires the monitor.

No translation for alternate function code address spaces (Error 161)

Cause: You tried to display an MMU translation for an address specified with alternate function codes 0, 3, 4, or 7.

Action: Don't use alternate function codes 0, 3, 4, or 7 when attempting to display an MMU translation for an address. The MMU does not translate addresses in alternate function code space.

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 hardware breakpoints (Error 154)

Cause: You either tried to set a breakpoint in target ROM or use the force hardware option to set a breakpoint in target RAM, and all eight hardware breakpoint resources are already in use.

Action: Review your present set of breakpoints to see if you can delete one or more of the hardware breakpoints that are presently set. No more than eight hardware breakpoints can be set at any one time (one per aligned long word). Only one hardware resource is used if two hardware breakpoints are set in the same long word.

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 odd or uninitialized (Error 150)

Cause: You tried to run the processor from the current PC, but the value of the current PC is odd.

Action: Modify the PC to an even value. The processor expects even word alignment of opcodes.

Program counter is located in guarded memory (Error 150)

Cause: You tried to run but the emulator detected that the program counter is located in guarded memory. This error will only be generated if the MMU is disabled; otherwise, you will see an asynchronous error indicating access to guarded memory occurred when the emulator attempted to run the target program.

Action: Make sure the program counter is set to an address in RAM or ROM before you attempt to run your program.

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 Commands", in this manual for a list of register names.

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.

Request cannot be satisfied with remaining map resources (Error 147)

Cause: Although you have not exceeded the maximum number of map terms that can be specified in the memory map, you have run into a hardware resource limitation in the emulator that arises when target memory is mapped including the transfer cache inhibit attribute.

There are eight hardware resources on the emulation probe for mapping emulation memory and driving the TCI signal for target memory ranges. When two emulation memory modules are installed, the emulator requires seven of these resources to map all of the emulation memory. Target memory ranges require either zero or one resource, depending on whether or not use of the transfer cache inhibit attribute matches its use in the "Default Memory Type" term. For example, if "Default Memory Type" is mapped to target RAM and the Transfer Cache Inhibit attribute is OFF, one hardware resource is required to add a map term for target memory that requires Transfer Cache Inhibit to be ON. Consuming additional hardware resources for mapping target memory will reduce the amount of emulation memory available for mapping. Once all eight hardware resources have been consumed, mappable emulation memory will be reduced to zero and you will get this message.

Action: Try to minimize the number of hardware resources used for mapping target memory by mapping the "Default Memory Type" term to target memory both with the Transfer Cache Inhibit attribute ON and OFF. Find out which specification for Default Memory Type uses the least number of hardware resources.

Request failed; bus grant (Error 171)

Cause: An attempt was made to execute a monitor command, but an external target system device has monopolized the bus and the monitor is no longer responding.

Action: Wait until the processor has regained bus control, and then retry the operation or don't let external devices monopolize the bus for extended periods of time.

Request failed; halted (Error 171)

Cause: During a monitor command, one or more target exceptions caused the processor to stop running bus cycles.

Action: Use the emulation-bus analyzer to determine what exceptions caused the problem and try to work around them.

Request failed; no bus cycles (Error 171)

Cause: During a monitor command, some problem caused the processor to stop running bus cycles.

Action: Use the emulation-bus analyzer to determine what caused the problem and try to work around them. If you are using the demo board, make sure the reset flying lead from the probe is connected to the demo board.

Request failed; no target power (Error 171)

Cause: You do not have proper power applied to your target system or demo board.

Action: Check the connection from your emulation probe to the target system or demo board. If using the demo board, be sure you have connected the external power cable correctly.

Request failed; slow clock (Error 171)

Cause: The target system is providing target power but no clock signal.

Action: Make sure the clock oscillator is installed correctly.

Request failed; target reset (Error 171)

Cause: During a monitor command, the target system asserted (and continues to assert) the reset signal; the monitor is no longer responding.

Action: Prevent your target system from asserting the reset signal when you are using monitor commands.

Request failed; unexpected exception: <vector number> (Error 171)

Cause: The monitor was executing a command and some exception occurred that it did not expect. During monitor command execution, the monitor traps all exceptions by using its own stack and vector table. The monitor provides exception handlers for some exceptions, such as access fault, so that it can either recover or issue a detailed error message. The monitor had no exception handler for the exception number shown in this message.

Action: Reset the emulator and try your command again.

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.

Stored deMMUer file: <FILE>

Cause: This is a status message. It indicates that a deMMUer configuration file was successfully stored.

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 bus error: <address> (Error 172)

Cause: The monitor attempted to access target system memory or memory that you specified must be terminated by the target system, and the target system terminated the bus cycle with TEA.

Action: Retry your command. If the error occurs again and if it is during an attempted access to emulation memory, you can answer "Yes" to the configuration question "Emulator Terminates Bus Cycles" for the emulation memory. If the error occurs again on access to target system memory, inspect your target system to understand why it is sending the TEA for the specified address.

Target failed to terminate bus cycle: <address> long read (Error 170)

Cause: You attempted to break or reset into the monitor and the target system failed to terminate a bus cycle with TA or TEA. Normally, the emulator will force bus cycle termination for the target system in order to break into the monitor. However, the emulator refused to terminate the bus cycle because the address was in program space or it was within the address range of the foreground monitor.

Action: Reset the emulator and target system. If the address is within emulation memory, answer "Yes" to the question "Emulator Terminates Bus Cycles" if the target system does not provide cycle terminations within this address range.

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.

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 access deMMUer while analysis trace is in process (Error 163)

Cause: You tried to issue a command that requires access to the deMMUer while the analyzer was running a trace. You cannot load, enable or disable the deMMUer while an analysis trace is in process.

Action: Wait for the trace to complete or stop the trace before changing the state of the deMMUer.

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 4, "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.

Unable to run HP64783 performance verification tests (Error 178)

Cause: You entered the **pv** command, but the emulator was unable to start performance verification because the firmware did not identify the probe as being the MC68040.

Action: Make sure the correct emulator probe is connected and that all cables are secured. Make sure that the demo board is connected to the emulator probe, the power cable is connected between the HP 64700 card cage and the demo board, and the reset flying lead is connected between the emulation probe and the demo board.

Unable to run HP64783 tests without target power (Error 178)

Cause: The demo board does not have proper power connected to it.

Action: Check the connections of the external power cable and the reset flying lead to the demo board.

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. If this message was received after you enabled the MMU, read "Execution Breakpoint Problems" in Chapter 10, "Using Memory Management".

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. This can happen if the MMU relocates a page containing a breakpoint before that breakpoint is executed. In this case, the emulator will have no record of the breakpoint at the relocated 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 177)

Cause: This error occurred when you attemped to disassemble a trace and the analyzer firmware was found to be out of date.

Action: Refer to Chapter 20, "Installing/Updating Emulator Firmware". You must update the firmware to the version number specified in the message, or newer firmware version number. Your analyzer is not able to disassemble its trace memory with its present firmware.

Update HP64700 system firmware to A.04.00 or newer (Error 176)

Cause: This error occurred because your system firmware is out of date.

Action: Refer to Chapter 20, "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.

Vector table modified for single stepping (Status 155)

Cause: This status message indicates that you issued the emulator command to single step. The emulator detected that the trace vector was not properly set for stepping so the emulator temporarily modified one or more exception vectors in your vector table. The original values are restored by the emulator after the step completes. This message is only issued one time if you do not change the address or value of the trace vector.

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. When the MMU is enabled, the address displayed in this message will be physical, as denoted by the trailing "a" after the function code.

13

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 when the operating system is HP-UX or /usr/openwin/lib/X11/app-defaults/HP64_Softkey when the operating system is 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 is read (typically contains resource specifications for a specific remote host).

5 Resource specifications included in the command line with the **-xrm** option.

6 System scheme files in directory /usr/hp64000/lib/X11/HP64_schemes.

7 System-wide custom scheme files located in directory $XAPPLRESDIR/HP64_schemes.

8 User-defined scheme files located in directory $HOME/.HP64_schemes (note the dot in the directory name).

Scheme files group resource specifications for different displays, computing environments, and languages.

This chapter shows you how to:

- Modify the Graphical User Interface resources.
- Use customized scheme files.
- Set up custom action keys.
- Set initial recall buffer values.
- Set up demos or tutorials.

Refer to Chapter 17, "X Resources and the Graphical Interface", for more detailed information.

To modify the Graphical User Interface resources

You can customize the appearance of an X Windows application by modifying its X resources. The following tables describe some of the commonly modified application resources.

Application Resources for Schemes		
Resource	**Values**	**Description**
HP64_Softkey.platformScheme	HP-UX SunOS (custom)	Names the subdirectory for platform specific schemes. This resource should be set to the platform on which the X server is running (and displaying the Graphical User Interface) if it is different from the platform where the application is running.
HP64_Softkey.colorScheme	BW Color (custom)	Names the color scheme file.
HP64_Softkey.sizeScheme	Small Large (custom)	Names the size scheme file which defines the fonts and the spacing used.
HP64_Softkey.labelScheme	Label $LANG (custom)	Names to use for labels and pushbutton text. The default uses the $LANG environment variable if it is set and if a scheme file named Softkey.$LANG exists in one of the directories searched for scheme files; otherwise, the default is Label.
HP64_Softkey.inputScheme	Input (custom)	Specifies mouse and keyboard operation.

Commonly Modified Application Resources		
Resource	**Values**	**Description**
HP64_Softkey.lines	24 (min. 18)	Specifies the number of lines in the main display area.
HP64_Softkey.columns	100 (min. 80)	Specifies the number of columns, in characters, in the main display area.
HP64_Softkey.enableCmdline	True False	Specifies whether the command line area is displayed when you initially enter the Graphical User Interface.
*editFile	(example) vi %s	Specifies the command used to edit files.
*editFileLine	(example) vi +%d %s	Specifies the command used to edit a file at a certain line number.
*<proc>*actionKeysSub.keyDefs	(paired list of strings)	Specifies the text that should appear on the action key pushbuttons 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** popup 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.

To modify the Graphical User Interface resources

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 recreated 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, "Using the Emulator/Analyzer Interface".

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 from 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 the name of your custom scheme file is different from the name of the default scheme file, 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 have different names than the default scheme files, you must modify these resource settings so that your customized scheme files are used instead of the default scheme files.

For example, to use the "$HOME/.HP64_schemes/HP-UX/Softkey.MyColor" color scheme file you would set the "HP64_Softkey.colorScheme" resource to "MyColor":

```
HP64_Softkey.colorScheme: MyColor
```

Refer to the previous "To customize Graphical User Interface resources" section for more detailed information on modifying resources.

To set up custom action keys

- Modify the "actionKeysSub.keyDefs" resource.

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

A pair of parentheses (with no spaces, that is "()") can be used in the command definition to indicate that text from the entry buffer should replace the parentheses when the command is executed.

Action keys that use the entry buffer should always include the entry buffer symbol, "( )", in the action key label as a visual cue to remind you to place information in the entry buffer before clicking the action key.

Shell commands can be executed by using an exclamation point prefix. A second exclamation point ends the command string and allows additional options on the command line.

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

Finally, an empty action ("") means to repeat the previous operation, whether it came from a pulldown, a dialog, a popup, or another action key.

Examples

To set up custom action keys when the graphical interface is used with the 68040 emulator, modify the "*m68040*actionKeysSub.keyDefs" resource:

```
*m68040*actionKeysSub.keyDefs: \
   "Make"               "cd /users/project2/68040; !make! in_browser" \
   "Load Pgm"           "load configuration config.EA; load program2" \
   "Run Pgm"            "run from reset" \
   "Trace after ( )"    "trace after (); display trace" \
   "Step Source"        "set source on; display memory mnemonic; step source" \
   "Again"              ""
```

Refer to "To modify Graphical User Interface resources" earlier in this chapter 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 popup recall buffers present in the Graphical User Interface. The resources for these popup recall buffers are listed in the following table.

  The window manager resource "*transientDecoration" controls the borders around dialog box windows. The most natural setting for this resource is "title."

Popup Recall Buffer Resources

Recall Popup	Resources	Description
File→Context→Directory ...	*dirSelect.textColumns *dirSelect.listVisibleItemCount *dirSelectSub.entries	The default number of text columns in the popup is 50. The default number of visible lines in the popup is 12. The "entries" resource is defined as a list of strings (see the following example). Up to 40 unique values are saved in each of the recall buffers (as specified by the resource settings "*XcRecall.maxDepth: 40" and "*XcRecall.onlyUnique: True").
File→Context→Symbols ...	*symSelect.textColumns *symSelect.listVisibleItemCount *symSelectSub.entries	
Trace→Trace Spec ...	*modtrace.textColumns *modtrace.listVisibleItemCount *modtraceSub.entries	
Entry Buffer ():	*recall.textColumns *recall.listVisibleItemCount *recallSub.entries	
Command Line command recall	*recallCmd.textColumns *recallCmd.listVisibleItemCount *recallCmdSub.entries	
Command Line pod/simio recall	*recallKbd.textColumns *recallKbd.listVisibleItemCount *recallKbdSub.entries	

Examples

To set the initial values for the directory selection dialog box when the Graphical User Interface is used with 68040 emulators, modify the "*m68040*dirSelectSub.entries" resource:

```
*m68040*dirSelectSub.entries: \
   "$HOME" \
   ".." \
   "/users/project1" \
   "/users/project2/68020"
```

Refer to "To modify the Graphical User Interface resources" earlier in this chapter 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 demo topic list popup.
*demoPopup.listVisibleItemCount	10	Specifies the length, in lines, of the demo topic list popup.
*demoTopic	About demos	Specifies the default topic in the demo popup selection buffer.

To set up demos or tutorials

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 tutorial topic list popup.
*tutorialPopup.listVisibleItemCount	10	Specifies the length, in lines, of the tutorial topic list popup.
*tutorialTopic	About tutorials	Specifies the default topic in the tutorial popup 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)" pushbutton (in the topic popup 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 popup 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/64783A).

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/64783A).

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 "To Modify Graphical User Interface resources" earlier in this chapter for more detailed information on modifying resources.

14

The SPARCsystem Graphical User Interface and Softkey Interface

Using the emulator/analyzer interface on the SPARCsystem

The SPARCsystem Emulator/Analyzer Interface

This chapter contains the following important information about use of the emulator on the SPARCsystem:

- Cross referencing HP product numbers used on HP-UX and SunOS versions of the 68040 emulator.
- A description of how the emulator/analyzer uses the keys on your SPARCsystem keyboard.

HP-UX/SunOS product number cross reference

The HP-UX and SunOS versions of the 68040 Emulator Interface operate the same way. Product numbers and other terminology is different. The following table will help you translate between HP-UX and SunOS product numbers:

Workstation Software [1]	
HP 64783A/B	HP emulator for Motorola 68040 microprocessor
HP B3090B	Graphical User Interface and Softkey Interface for use with HP 64783A/B
HP B1463B	C cross compiler
HP B1465B	cross assembler/linker
HP B1467B	debugger/simulator
HP B1477B	debugger/emulator
HP B1487A	Software Performance Analyzer
HP B1418A	HP Branch Validator
[1]Support for HP 9000 and SunOS can be obtained by ordering the following options to the above product numbers:	
Option AAV	SunOS™ version, media and documentation
Option AAX	HP 9000, 300/400 versions, media and documentation
Option AAY	HP 9000, 700 version, media and documentation

Using your SPARCsystem keyboard

The following table describes the use of the keys on your SPARCstation keyboard. The "roll" functions are for an hp64term window in OpenWindows only.

To get function	Press this key		
delete char		Del	
insert char		Ins	(For SunView and xterm windows only. Does not work in OpenWindows windows.)
next page		PgDn	
prev page		PgUp	
roll up	Shift	↑	(for xterm only)
roll down	Shift	↓	(for xterm only)
roll left	Shift	←	(for xterm only)
roll right	Shift	→	(for xterm only)
up arrow		↑	
down arrow		↓	
left arrow		←	
right arrow		→	
home		Home	
anti-home		End	
carriage return		Enter	

To get function	Press this key		
tab		Tab	
backtab		Undo	
cleartoeol	Control	E	
roll left	Control	F	
roll right	Control	G	
clearline	Control	U	
FW_recall	Control	B	
BK_recall	Control	R	
eof	Control	D	
refresh	Control	L	
other	Control	X	
sigint	Control	C	
sigquit	Control	|	end release
kill !!!	Control	Z	(**Do not use!** Disabled in xterm window created by hp64term.)
softkey 1		F1	
softkey 2		F2	

To get function	Press this key
softkey 3	F3
softkey 4	F4
softkey 5	F5
softkey 6	F6
softkey 7	F7
softkey 8	F8

Keyboard template

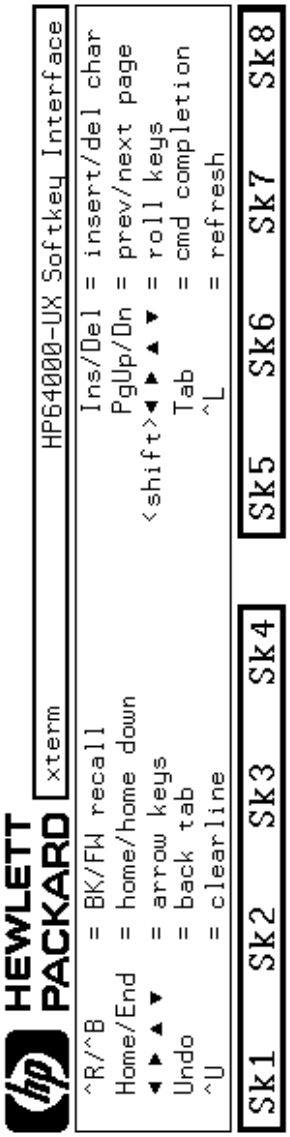
Cut out the template on this page and place it on your keyboard for quick reference.

Place the Sk1 label over the F1 key.

If you exported KEYMAP=sun.2-9 or KEYMAP=xterm.2-9, then place the Sk1 label over the F2 key.

The roll keys only work in xterm windows.

^R means to hold down the control key while pressing R.



15

Microtec Language Tools Used With MC68040 Emulators

Using the emulator/analyzer with Microtec language tools

Microtec Language Tools used with the Emulator/Analyzer

This chapter contains the following important information about use of Microtec language tools with the 68040 emulator/analyzer:

- A comparison of the names of Microtec and HP executables.
- Microtec commands.
- Assembler defaults.
- Linker defaults.
- Librarian defaults.

Using Microtec Language Tools

You can use the Microtec Research, Inc. language tools with emul700 and **db68k**. The language tools available from Microtec are the **mcc68k** C compiler and the **asm68k** assembler.

The following table compares the names of the Microtec and HP executables:

Executables for the MC68040:

Program	Microtec name	HP AxLS name	HP product number
assembler	asm68k	as68040	B3641
linker	lnk68k	ld68040	B3641
librarian	lib68k	ar68040	B3641
compiler	mcc68k	cc68040	B3640

The emulator and Softkey Interface are designed to work with the HP Advanced Cross Language System. Microtec's language tools are quite similar to the HP language tools. The input syntax and code generated by the HP and Microtec assemblers, linkers, and librarians are identical with few exceptions.

Microtec uses a license manager to enable execution of their products. Be sure to read about this in the Microtec manuals.

To use the Microtec commands

- To assemble a file to generate an HP-OMF (**.X**) absolute file, use the following commands:

```
asm68k -h -o filename.o -f d,case filename.s
lnk68k  -H filename.L  -o filename.X -c filename.k filename.o
```

- To compile a C program to generate an HP-OMF object file, use the following commands:

```
mcc68k -hgc -Wa,-f"d"  -o filename.o filename.c
lnk68k -m -fd -H filename.L -c filename.k filename.o > filename.MAP
```

- To assemble a file to generate an HP-MRI IEEE-695 (**.x**) absolute file, use the following commands:

```
asm68k -o filename.o -f d,case filename.s
lnk68k -fi -o filename.x -c filename.k filename.o
```

- To compile a C program to generate an HP-MRI IEEE-695 (**.x**) absolute file, use the following commands:

```
mcc68k  -Wl,-m,-fd  -g -I/usr/hp64000/include/68xxx  -efilename.k  -o
filename.X \
 filename.c
```

Assembler defaults

You should be aware of these differences between **asm68k** and **as68k**:

Command-line syntax

The differences are minor. See the on-line man pages for a description of the command-line options.

Case sensitivity

as68k is case sensitive by default, **asm68k** is not. Use the command line flag "-fcase" to make **asm68k** case sensitive.

Symbols in HP-MRI IEEE-695 files

The HP assembler places local symbols in the output object file by default, **asm68k** does not. Use the command line flag "-fd' with **asm68k** to generate local symbols.

The HP assembler places global symbols put in the debug part by default. There is no way to do this with Microtec's **asm68k**. This information is needed by emul700/SRU to correctly scope symbols. Thus you will find that some symbols may be incorrectly scoped when using the emulator with the Microtec assembler.

Linker defaults

You should be aware of these differences between **lnk68k** and **ld68k**:

Output file format

ld68k produces HP-MRI IEEE-695 by default. **lnk68k** produces Motorola S-Records by default. To generate an HP-MRI IEEE-695 (**.x**) format absolute file, use the **-H** command line option or **-fi** flag.

Local symbols

ld68k provides local symbols in absolute file by default, but **lnk68k** does not. The command line flag **-fi** and option **-H** also set the **d** flag, which will cause **lnk68k** to generate local symbols.

Support files

ld68k and **lnk68k** have different default locations and environment variables used to locate linker command files and libraries.

Librarian defaults

ar68k uses **.a** as the default library suffix. **lib68k** uses **.lib** as the default library suffix.

The Microtec MCC68K compiler

mcc68k is very different from **cc68k**. Study the Microtec documentation if you need specific information about **mcc68k**.

To get **db68k** to work with **mcc68k** output, you need to add the following to your .profile:

```
HP64_DEBUG_PATH=<path to source files>
export HP64_DEBUG_PATH
```

If this is not done, the debugger cannot find the high-level “C” source files.

16

Specifications and Characteristics

Processor Compatibility

The HP 64783A/B is compatible with the Motorola MC68040, MC68EC040, and MC68LC040 processors, and with any processors that meet all specifications of the MC68040, MC68EC040, and MC68LC040 processors.

Electrical

Maximum clock speed

The maximum external clock speed of the HP 64783A is 33 MHz, and of the HP 64783B is 40 MHz. The emulator runs without wait states at clock speeds up to 25 MHz. Above 25 MHz, one wait state is required in all bus cycles and between burst transfers.

Motorola JTAG

HP 64783A/B does not support Motorola JTAG. Therefore, no specifications are given for Motorola JTAG in this manual.

HP 64783A/B Maximum Ratings

Characteristic	Symbol	Value	Unit
Supply Voltage	V_{CC}	–0.3 to +5.5	V
Input Voltage	V_{in}	–0.5 to +5.5	V
Maximum Operating Ambient Temperature	T_A	45	^oC
Minimum Operating Ambient Temperature	T_A	0	^oC
Storage Temperature Range	T_{stg}	–40 to +70	^oC

HP 64783A/B Electrical Specifications

HP 64783A/B — DC ELECTRICAL SPECIFICATIONS (V_{CC}=5.0 Vdc ±5%)				
Characteristic	**Symbol**	**Min**	**Max**	**Unit**
Input High Voltage	V_{IH}	2	V_{CC}	V
Input Low Voltage	V_{IL}	GND	0.8	V
Undershoot		—	0.5	V
Input Leakage Current @ 0.5/2.4 V $\overline{AVEC}$, BCLK, $\overline{BG}$, $\overline{CDIS}$, $\overline{MDIS}$, $\overline{IPLx}$, PCLK, $\overline{RSTI}$, SCx, $\overline{TBI}$, TLNx, $\overline{TCI}$, TCK, $\overline{TEA}$	I_{IL} I_{IH}	–250 —	— 25	µA
Hi-Z (Off-State) Leakage Current @ 0.5/2.4 V An, $\overline{CIOUT}$, Dn, $\overline{LOCK}$, $\overline{LOCKE}$, SIZx, TDO, $\overline{TMx}$, $\overline{TLNx}$, $\overline{TTx}$, $\overline{UPAx}$ $\overline{BB}$, R/$\overline{W}$, $\overline{TIP}$, $\overline{TS}$ $\overline{TA}$	I_{TSI}	 –50 –100 –200	 50 100 200	µA
Output High Voltage I_{OH} = –32 mA: An, Dn, SIZx, TTx, UPAx, $\overline{LOCK}$, $\overline{LOCKE}$, TLNx, $\overline{CIOUT}$, TMx, PSTx, $\overline{RSTO}$, $\overline{BR}$, $\overline{MI}$, $\overline{BG}$,reset flying lead I_{OH} = –3.2 mA: R/$\overline{W}$, $\overline{TS}$, $\overline{TIP}$, $\overline{BB}$, $\overline{TA}$, $\overline{IPEND}$	V_{OH}	 2.0 2.4	 — —	V
Output Low Voltage I_{OL} = 64 mA An, Dn, SIZx, TTx, UPAx, $\overline{LOCK}$, $\overline{LOCKE}$, TLNx, $\overline{CIOUT}$, TMx, PSTx, $\overline{RSTO}$, $\overline{BR}$, $\overline{MI}$, $\overline{BG}$, reset flying lead I_{OL} = 24 mA R/$\overline{W}$, $\overline{TS}$, $\overline{TIP}$, $\overline{BB}$, $\overline{TA}$, $\overline{IPEND}$	V_{OL}	 — —	 0.55 0.5	V
Capacitance V_{in}=0 V, f=1 MHz	C_{in}	—	25	pF

HP 64783A/B — DC ELECTRICAL SPECIFICATIONS (V_{CC}=5.0 Vdc ±5%)				
Characteristic	**Symbol**	**Min**	**Max**	**Unit**
Supply Current f = 25 MHz f = 33 MHz	I_{CC}	 — —	 1.4 1.8	 A A

Notes for HP 64783A/B Electrical Specifications:

BCLK and PCLK have additional input current and capacitance loading because of RC terminations. Refer to their equivalent circuit diagrams for details. The numbers given in the HP 64783A/B Electrical Specifications table do not include the RC terminations.

HP 64783A/B Clock AC Timing Specifications

Num	Characteristic	25 MHz		33 MHz		40 MHz		Unit
		Min	Max	Min	Max	Min	Max	
	Frequency of Operation	16.67	25	16.67	33	20	40	MHz
1	PCLK Cycle Time	20	30	15	30	12.5	25	ns
2	PCLK Rise Time		1.7		1.7	—	1.5	ns
3	PCLK Fall Time		1.6		1.6	—	1.5	ns
4	PCLK Duty Cycle Measured at 1.5 V	47.50	52.50	46.67	53.33	46.0	54.00	%
4a[1]	PCLK Pulse Width High Meas. at 1.5 V	9.50	10.50	7	8	5.75	6.75	ns
4b[1]	PCLK Pulse Width Low Measured at 1.5 V	9.50	10.50	7	8	5.75	6.75	ns
5	BCLK Cycle Time	40	60	30	60	25	50	ns
6,7	BCLK Rise and Fall Time	—	4	—	3	—	3	ns
8	BCLK Duty Cycle Measured at 1.5 V	40	60	40	60	40	60	%
8a[1]	BCLK Pulse Width High Measured at 1.5 V	16	24	12	18	10	15	ns
8b[1]	BCLK Pulse Width Low Measured at 1.5 V	16	24	12	18	10	15	ns
9	PCLK, BCLK Frequency Stability	—	1000	—	1000	—	1000	ppm
10	PCLK to BCLK Skew	—	n/a	—	n/a	—	n/a	ns

Notes for Clock AC Timing Specifications:

1 Specification value at maximum frequency of operation.

HP 64783A/B Output AC Timing Specifications

Num	Characteristic	25 MHz[1]		33 MHz[1]		40 MHz[1]		Unit
		Min	Max	Min	Max	Min	Max	
11	BCLK to Address $\overline{\text{CIOUT}}$, $\overline{\text{LOCK}}$, $\overline{\text{LOCKE}}$, R/$\overline{\text{W}}$, SIZx, TLNx, TMx, TTx, UPAx Valid	9	25	6.5	22.5	5.25	21	ns
12	BCLK to Output Invalid (Output Hold)	9	—	6.5	—	5.25	—	ns
13	BCLK to $\overline{\text{TS}}$ Valid	9	25	6.5	22.5	5.25	21	ns
14	BCLK to $\overline{\text{TIP}}$ Valid	9	25	6.5	22.5	5.25	22	ns
18	BCLK to Data Out Valid	9	27	6.5	24.5	5.25	23	ns
19	BCLK to Data Out Invalid (Output Hold)	9	—	6.5	—	5.25	—	ns
20	BCLK to Output Low Impedance	3	—	3	—	3	—	ns
21	BCLK to Data-Out High Impedance	9	32	6.5	27	5.25	24.5	ns
26[2]	BCLK to Multiplexed Address Valid	n/a	n/a	n/a	n/a	n/a	n/a	ns
27[2]	BCLK to Multiplexed Address Driven	n/a	—	n/a	—	n/a	—	ns
28[2]	BCLK to Multiplexed Address High Impedance	n/a	n/a	n/a	n/a	n/a	n/a	ns
29[2]	BCLK to Multiplexed Data Driven	n/a	—	n/a	—	n/a	—	ns
30[2]	BCLK to Multiplexed Data Valid	n/a	n/a	n/a	n/a	n/a	n/a	ns
38	BCLK to Address, $\overline{\text{CIOUT}}$, $\overline{\text{LOCK}}$, $\overline{\text{LOCKE}}$, R/$\overline{\text{W}}$, SIZx, $\overline{\text{TS}}$, TLNx, TMx, TTx, UPAx High Impedance	9	31	6.5	26	5.25	23.5	ns
39	BCLK to $\overline{\text{BB}}$, $\overline{\text{TA}}$, $\overline{\text{TIP}}$ High Impedance	19	31	14	26	11.5	23.5	ns

Num	Characteristic	25 MHz[1]		33 MHz[1]		40 MHz[1]		Unit
		Min	Max	Min	Max	Min	Max	
40	BCLK to $\overline{BR}$, $\overline{BB}$ Valid	9	25	6.5	22.5	5.25	21	ns
43	BCLK to $\overline{MI}$ Valid	9	25	6.5	22.5	5.25	21	ns
48	BCLK to $\overline{TA}$ Valid	9	25	6.5	22.5	5.25	21	ns
50	BCLK to $\overline{IPEND}$, PSTx, $\overline{RSTO}$ Valid	9	25	6.5	22.5	5.25	21	ns

Notes:

1 Output timing is given for output drivers specified in the DC specs (Refer to the table of HP 64783A/B Electrical Specifications). Large/small buffer mode select has no effect.

2 Address multiplex mode is not supported.

HP 64783A/B Input AC Timing Specifications

Num	Characteristic	25 MHz		33 MHz		40 MHz		Unit
		Min	Max	Min	Max	Min	Max	
15	Data-In Valid to BCLK (Setup)	9	—	9	—	8	—	ns
16	BCLK to Data-In Invalid (Hold)	4	—	4	—	3	—	ns
17	BCLK to Data-In High Impedance (Read Followed by Write)	—	49	—	36.5	—	30.25	ns
22a	$\overline{TA}$ Valid to BCLK (Setup)	15	—	15	—	13	—	ns
22b	$\overline{TEA}$ Valid to BCLK (Setup)	15	—	15	—	14	—	ns
22c	$\overline{TCI}$ Valid to BCLK (Setup)	15	—	15	—	14	—	ns
22d	$\overline{TBI}$ Valid to BCLK (Setup)	15	—	15	—	14	—	ns
23	BCLK to $\overline{TA}$, $\overline{TEA}$, $\overline{TCI}$, $\overline{TBI}$ Invalid (Hold)	2	—	2	—	2	—	ns
24	$\overline{AVEC}$ Valid to BCLK (Setup)	10	—	10	—	10	—	ns
25	BCLK to $\overline{AVEC}$ Invalid (Hold)	2	—	2	—	2	—	ns
31[1]	DLE Width High	n/a	—	n/a	—	n/a	—	ns
32[1]	Data-In Valid to DLE (Setup)	n/a	—	n/a	—	n/a	—	ns
33[1]	DLE to Data-In Invalid (Hold)	n/a	—	n/a	—	n/a	—	ns
34[1]	BCLK to DLE Hold	n/a	—	n/a	—	n/a	—	ns
35[1]	DLE High to BCLK	n/a	—	n/a	—	n/a	—	ns

Num	Characteristic	25 MHz		33 MHz		40 MHz		Unit
		Min	Max	Min	Max	Min	Max	
36[1]	Data-In Valid to BCLK (DLE Mode Setup)	n/a	—	n/a	—	n/a	—	ns
37[1]	BCLK to Data-In Invalid (DLE Mode Hold)	n/a	—	n/a	—	n/a	—	ns
41a	$\overline{BB}$ Valid to BCLK (Setup)	12	—	12	—	12	—	ns
41b	$\overline{BG}$ Valid to BCLK (Setup)	12	—	12	—	12	—	ns
41c	$\overline{CDIS}$, $\overline{MDIS}$ Valid to BCLK (Setup)	13	—	13	—	13	—	ns
41d	$\overline{IPLx}$ Valid to BCLK (Setup)	8	—	8	—	8	—	ns
42	BCLK to $\overline{BB}$, $\overline{BG}$, $\overline{CDIS}$, $\overline{IPLx}$, $\overline{MDIS}$ Invalid (Hold)	2	—	2	—	2	—	ns
44a	Address Valid to BCLK (Setup)	12	—	12	—	12	—	ns
44b	SIZx Valid to BCLK (Setup)	13	—	13	—	13	—	ns
44c	TTx Valid to BCLK (Setup)	13	—	13	—	13	—	ns
44d	R/$\overline{W}$ Valid to BCLK (Setup)	10	—	10	—	10	—	ns
44e	SCx Valid to BCLK (Setup)	16	—	16	—	13	—	ns
45	BCLK to Address, SIZx, TTx, R/$\overline{W}$, SCx Invalid (Hold)	2	—	2	—	2	—	ns
46	$\overline{TS}$ Valid to BCLK (Setup)	14	—	14	—	12	—	ns
47	BCLK to $\overline{TS}$ Invalid (Hold)	2	—	2	—	2	—	ns
49	BCLK to $\overline{BB}$ High Impedance (MC68040 Assumes Bus Mastership)	—	9	—	9	—	9	ns

Num	Characteristic	25 MHz		33 MHz		40 MHz		Unit
		Min	Max	Min	Max	Min	Max	
51	$\overline{\text{RSTI}}$ Valid to BCLK	9	—	9	—	9	—	ns
52	BCLK to $\overline{\text{RSTI}}$ Invalid	2	—	2	—	2	—	ns
53[2]	Mode Select Setup to $\overline{\text{RSTI}}$ Negated	n/a	—	n/a	—	n/a	—	ns
54[2]	$\overline{\text{RSTI}}$ Negated to Mode Selects Invalid	n/a	—	n/a	—	n/a	—	ns

Notes:

1 Data Latch mode is not supported.
2 Mode selects are not used.

Physical

Emulator Dimensions

173 mm height x 325 mm width x 389 mm depth (6.8 in. x 12.8 in. x 15.3 in.)

Emulator Weight

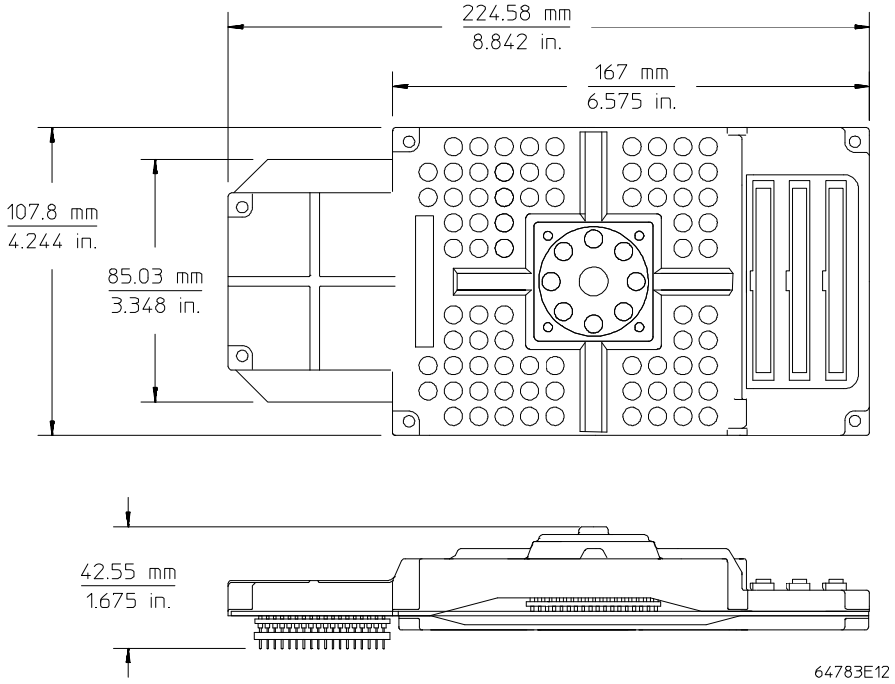
HP 64783A/B, 8.2 kg (18 lb). Any component used in suspending the emulator must be rated for 30 kg (65 lb) capacity.

Probe alone: 0.3 kg (10 oz).

Cable Length

Emulation Control Card to Probe, approximately 914 mm (36 inches).

Probe dimensions



Environmental

Temperature

Operating, 0° to +40° C (+32° to +104° F); nonoperating, -40° C to +60° C (-40° F to +140° F).

Altitude

Operating/nonoperating 4600 m (15 000 ft).

Relative Humidity

15% to 95%.

BNC, labeled TRIGGER IN/OUT

Output Drive

Logic high level with 50-ohm load >= 2.0 V. Logic low level with 50-ohm load <= 0.4 V.

Input

74HCT132 with 135 ohms to ground in parallel. Maximum input: 5 V above Vcc; 5 V below ground.

Communications

Host Port

25-pin female type “D” subminiature connector.

RS-232-C DCE or DTE to 38.4 kbaud.

RS-422 DCE only to 460.8 kbaud.

CMB Port

9-pin female type “D” subminiature connector.

Part 4

Concept Guide

Concepts Guide

In This Part

Part 4 of this book provides conceptual information on the X resources and the Graphical User Interface.

17

X Resources and the Graphical User Interface

X Resources and the Graphical User Interface

This chapter contains more detailed information about X resources and scheme files that control the appearance and operation of the Graphical User Interface. This chapter:

- Describes the X Window concepts surrounding resource specification.
- Describes the Graphical User Interface's implementation of scheme files.

X Resource Specifications

An X resource specification is a resource name and a value. The resource name identifies the element whose appearance or behavior is to be defined, and the value specifies how the element should look or behave. For example, consider the following resource specification:

```
Application.form.row.done.background: red
```

The resource name is "Application.form.row.done.background:" and the value is "red".

Resource Names Follow Widget Hierarchy

A *widget* is an OSF/Motif graphic device from which X applications are built. For example, pushbuttons and menu bars are Motif widgets. Applications are built using a hierarchy of widgets, and the application's X resource names follow this hierarchy. For example:

```
Application.form.row.done.background: red
```

In the resource name above, the top-level widget is named after the application. One of the top-level widget's children is a form widget, one of the form widget's children is a row-column manager widget, and one of the row-column manager widget's children is a pushbutton widget. Resource names show a path in the widget hierarchy.

Each widget in the hierarchy is a member of a widget class, and the particular instance of the widget is named by the application programmer.

Class Names or Instance Names Can Be Used

When specifying resource names, you can use either instance names or class names. For example, a "Done" pushbutton may have an instance name of "done" and a class name of "XmPushButton". To set the background color for a hypothetical "Done" pushbutton, you can use:

```
Application.form.row.done.background: red
```

Or, you can use:

```
Application.form.row.XmPushButton.background: red
```

Applications also have class and instance names. For example, an application may have an instance name of "applic1" and a class name of "Application". To set the background color for a hypothetical "Done" pushbutton only in the "applic1" application, you can use:

```
applic1.form.row.done.background: red
```

Note that instance names are more specific than class names. That is, class names may apply to many instances of the widget.

The class and instance names for the widgets in the Graphical User Interface can be displayed by choosing **Help→X Resource Names** and clicking on the "All names" pushbutton.

Wildcards Can Be Used

A wildcard may be used to match a resource specification to many different widgets at once. For example, to set the background color of all pushbuttons, you can use:

```
Application*XmPushButton.background: red
```

Note that resource names with wildcards are more general than those without wildcards.

Specific Names Override General Names

A more specific resource specification will override a more general one when both apply to a particular widget or application.

The names for the application and the main window widget in HP64_Softkey applications have been chosen so that you may specify custom resource values that apply in particular situations:

1 Apply to ALL HP64_Softkey applications:

HP64_Softkey*<resource>: <value>

2 Apply to specific types of HP64_Softkey applications:

emul*<resource>: <value> (for the emulator)
perf*<resource>: <value> (for the performance analyzer)

3 Apply to all HP64_Softkey applications, but only when they are connected to a particular type of microprocessor:

m68040<resource>: <value>

4 Apply to a specific HP64_Softkey application connected to a specific processor:

perf.m68040*<resource>: <value> (for the 68040 perf. analyzer)
emul.m68040*<resource>: <value> (for the 68040 emulator)

If all four examples above are used for a particular resource, #3 will override #2 for all applications connected to a 68040 emulator, and #4 will override #2, but only for the specifically mentioned type of microprocessor.

When modifying resources, your resource paths must either match, or be more specific than, those found in the application defaults file.

How X Resource Specifications are Loaded

When the Graphical User Interface starts up, it loads resource specifications from a set of configuration files located in system directories and user-specific locations.

Application Default Resource Specifications

Default resource specifications for an application are placed in a system directory:

HP-UX	/usr/lib/X11/app-defaults
SunOS	/usr/openwin/lib/X11/app-defaults

The name of the Graphical User Interface application defaults file is HP64_Softkey (same as the application class name). This file is well-commented and contains information about each of the X resources you can modify. You can easily view this file by choosing **Help→Topic** and selecting the "X Resources: App Default File" topic. Do not modify the application defaults file; any changes to this file will affect the appearance and behavior of the application for all users.

User-Defined Resource Specifications

User-defined resources (for any X application) are located in the X server's RESOURCE_MANAGER property or in the user's $HOME/.Xdefaults file.

Load Order

Resource specifications are loaded from the following places in the following order:

1 The application defaults file. For example, /usr/lib/X11/app-defaults/HP64_Softkey when the operating system is HP-UX or /usr/openwin/lib/X11/app-defaults/HP64_Softkey when the operating system is SunOS.

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 is read (typically contains resource specifications for a specific remote host).

5 Resource specifications included in the command line with the **-xrm** option.

When specifications with identical resource names appear in different places, the latter specification overrides the former.

Scheme Files

Several of the Graphical User Interface's X resources identify *scheme files* that contain additional X resource specifications. Scheme files group resource specifications for different displays, computing environments, and languages.

Resources for Graphical User Interface Schemes

There are five X resources that identify scheme files:

HP64_Softkey.labelScheme:

Names the scheme file to use for labels and pushbutton text. Values can be: Label, $LANG, or a custom scheme file name. The default uses the $LANG environment variable if it is set and if a scheme file named Softkey.$LANG exists in one of the directories searched for scheme files; otherwise, the default is Label.

HP64_Softkey.platformScheme:

Names the subdirectory for the platform specific color, size, and input scheme files. This resource should be set to the platform on which the X server is running (and displaying the Graphical User Interface) if it is different from the platform where the application is running. Values can be: HP-UX, SunOS, pc-xview, or a custom platform scheme directory name.

HP64_Softkey.colorScheme:

Names the color scheme file. Values can be Color, BW, or a custom scheme file name.

HP64_Softkey.sizeScheme:

Names the size scheme file which defines the fonts and the spacing used. Values can be Large, Small, or a custom scheme file name.

HP64_Softkey.inputScheme:

Names the input scheme file which specifies mouse and keyboard operation. Values can be Input, or a custom scheme file name.

The actual scheme file names take the form, "Softkey.<value>".

Scheme File Names

There are six scheme files provided with the Graphical User Interface. Their names and brief descriptions of the resources they contain follow.

Softkey.Label	Defines the labels for the fixed text in the interface. Such things as menu item labels and similar text are in this file. If the $LANG environment variable is set, the scheme file "Softkey.$LANG" is loaded if it exists; otherwise, the file "Softkey.Label" is loaded.
Softkey.BW	Defines the *color scheme* for black and white displays. This file is chosen if the display cannot produce at least 16 colors.
Softkey.Color	Defines the *color scheme* for color displays. This file is chosen if the display can produce 16 or more colors.
Softkey.Large	Defines the *size scheme* (that is, the window dimensions and fonts) for high resolution displays (1000 pixels or more vertically).
Softkey.Small	Defines the *size scheme* (that is, the window dimensions and fonts) for low resolution displays (less than 1000 pixels vertically).
Softkey.Input	Defines the *input scheme* (that is, the pushbutton and key bindings for the mouse and keyboard).

Load Order for Scheme Files

Scheme files are searched for in the following directories and in the following order:

1. System scheme files in directory /usr/hp64000/lib/X11/HP64_schemes.
2. System-wide custom scheme files located in directory $XAPPLRESDIR/HP64_schemes.
3. User-defined scheme files located in directory $HOME/.HP64_schemes (note the dot in the directory name).

Custom Scheme Files

You can modify scheme files by copying them to the directory for user-defined schemes and changing the resource specifications in the file. For example, if you wish to modify the color scheme, and your platform is HP-UX, you can copy the /usr/hp64000/lib/X11/HP64_schemes/HP-UX/Softkey.Color file to $HOME/.HP64_schemes/HP-UX/Softkey.Color and modify its resource specifications.

You can create custom scheme files by modifying the X resource for the particular scheme and by placing the custom scheme file in the directory for user-defined schemes. For example, if the following resource specifications are made:

```
HP64_Softkey.platformScheme:  HP-UX
HP64_Softkey.colorScheme:     MyColor
```

The custom scheme file would be:

```
$HOME/.HP64_schemes/HP-UX/Softkey.MyColor
```

Part 5

Installation and Service Guide

Installation and Service Guide

In This Part

Part 5 of this book shows you how to:

- Connect the emulator into an MC68040 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
- Use the progflash program to ensure software compatibility.

18

Connecting the Emulator to a Target System

Things you need to know to successfully connect the emulator to a target system and overcome problems you may encounter.

Plugging The Emulator Into A Target System

The following paragraphs help you understand the emulator. Equivalent circuits are shown, followed by a list of devices that you may need to use to overcome mechanical and electrical constraints in your target system.

Understanding an emulator

An emulator is a tool intended for debugging software, and the interactions between software and hardware. Although emulators can help in debugging certain hardware problems, catastrophic problems often require use of other tools, such as a timing analyzers with preprocessors, or oscilloscopes. To effectively use an emulator, you need to understand its capabilities and limitations, and how it interacts with your target system. This chapter discusses limitations and interactions of an emulator, as they relate to your target system.

An emulator is designed to be electrically and functionally equivalent to the processor it emulates, as much as possible. Most MC68040 signals are electrically isolated from their counterparts on the target system connection. This is done for both electrical and functional reasons. Equivalent circuits of each processor signal are shown later in this chapter. The impacts of these circuits are calculated and presented in the emulator specifications listed in Chapter 16, "Specifications and Characteristics".

In the ideal case, you would use the emulator specifications listed in this manual when designing your target system, instead of using the processor specifications. In the typical case, your target system has already been designed and prototyped. A target system that is designed around MC68040 worst-case specifications will typically work with the emulator. If certain circuits in your target system do not allow for variations in the MC68040 specifications, compare the relevant emulator specifications to evaluate their impact on your target system. By keeping the differences between emulator specifications and processor specifications in mind while you design your target system, you can save hours of debugging time when you plug the emulator into your target system.

The MC68040 emulator does not switch between large and small buffer modes like the MC68040 processor does. The emulator internally uses the large buffer mode to get optimum timing performance. Since these large drivers can cause problems

for systems designed to work with small buffer mode, the emulator buffers all signals from the processor to the target connector. Most of the signals are buffered in ABT logic family parts. These parts are chosen to provide high speed and high current capability while keeping slew rates to an acceptable level for small buffer mode systems. Some control signals are buffered in PALs which have significantly less drive capability than the processor in large mode.

Examine the DC specifications of the emulator to evaluate their differences from processor specifications. Again, you can refer to the equivalent circuit diagrams in this chapter for exact details. Because the emulator does not behave exactly like the processor, you may need to examine signal quality and take appropriate steps to compensate for differences.

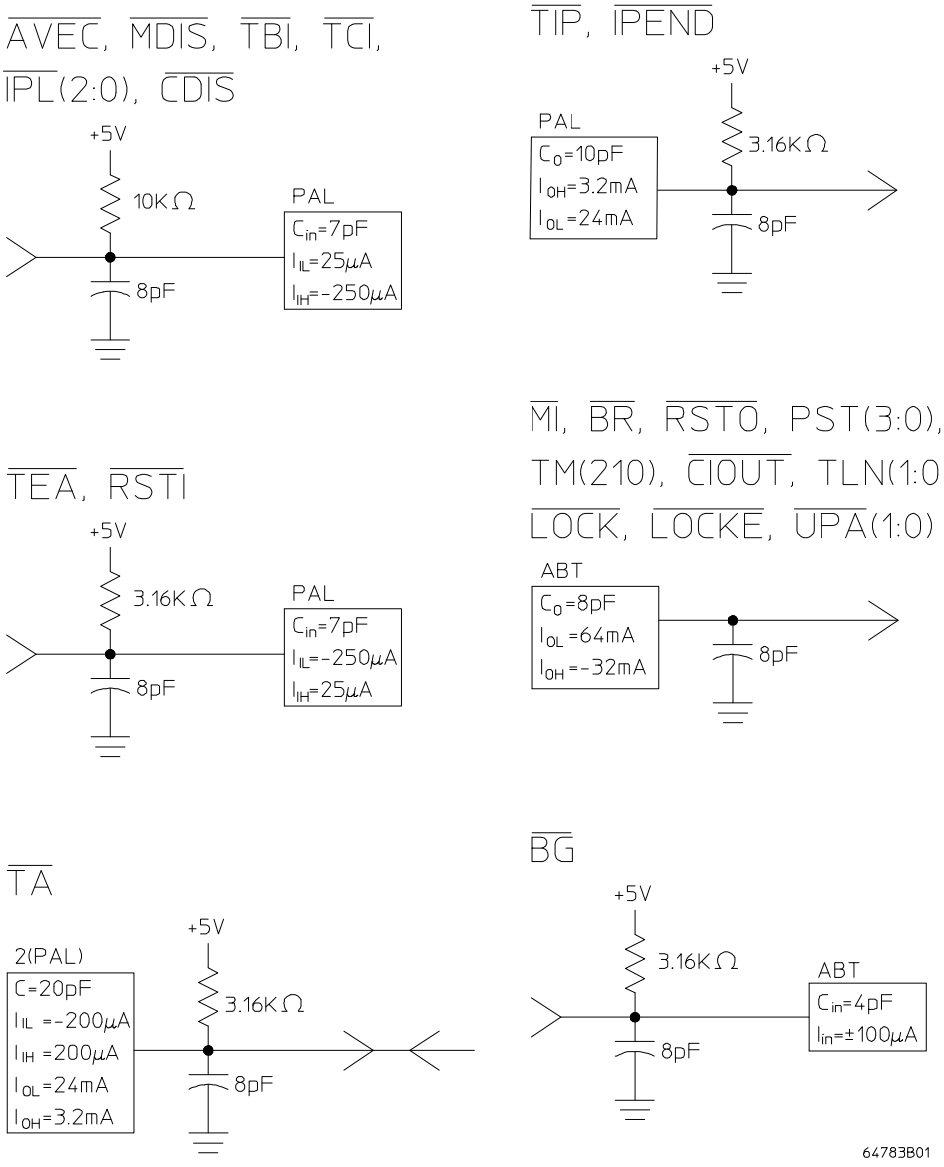
The BCLK clock is the most important signal to the emulator because all system timing is derived from this signal. The BCLK clock signal must have clean edges; the duty cycle of this clock is not particularly important. The emulator regenerates an internal BCLK from this signal with a 50% duty cycle. All timing is referenced from the rising edge of BCLK. The PCLK clock is also internally regenerated; therefore, the emulator is not sensitive to this signal.

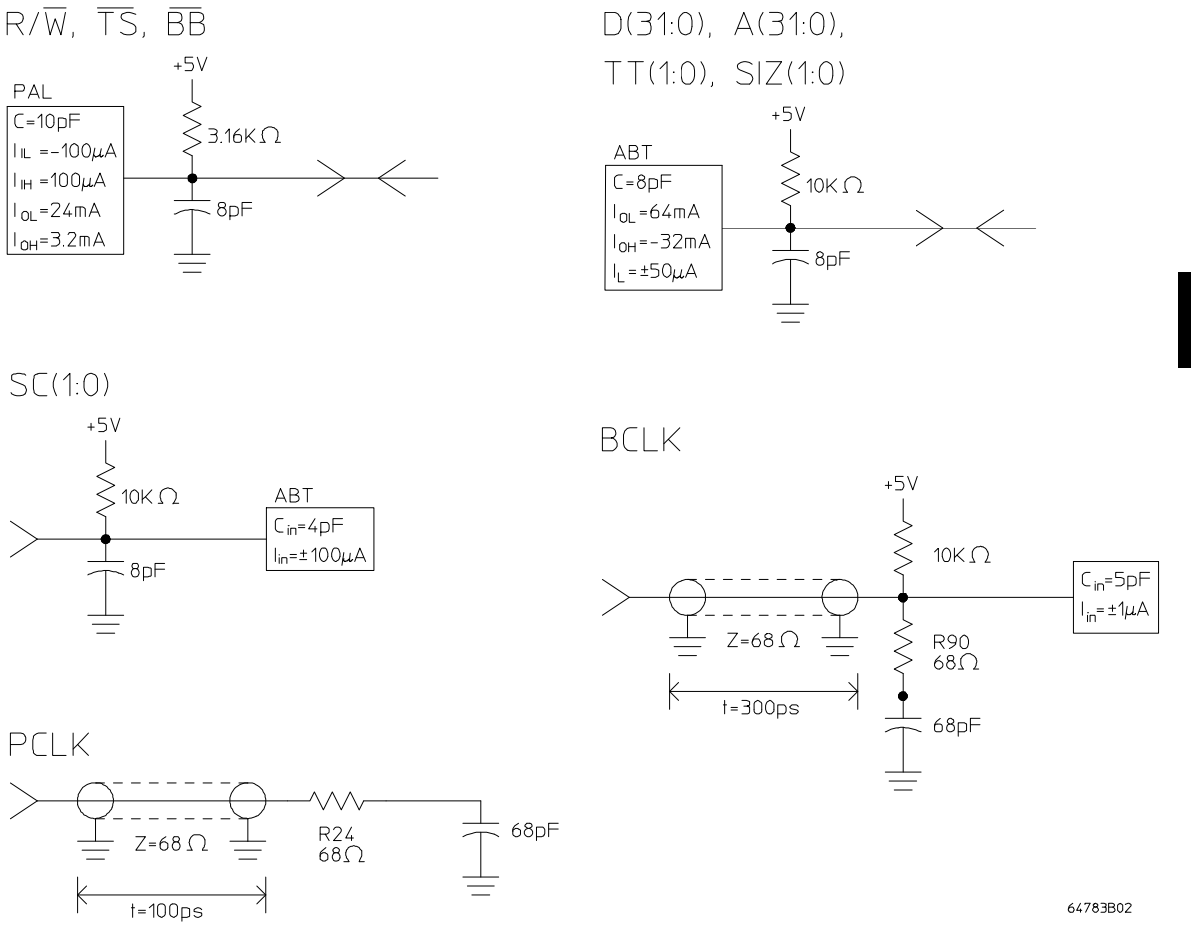
Both the BCLK and PCLK signals are terminated on the emulator. The terminations are placed on these signals, even though the emulator causes only a short electrical stub, so that accessories such as the flexible cable can be used to connect the emulator probe to your target system. The terminations on these signals can interact with terminations on your target system. Refer to the equivalent circuits in this chapter and adjust terminations in your target system for best results.

The emulator uses power from the target system to operate the emulation processor and some pullup resistors. Target power is sensed to make sure the emulator does not drive the target system until it is powered up. In addition, the power detection circuit delays release of processor reset for 50 ms after power is in specification to allow the clock circuits to synchronize. Because of the protections designed into the emulator, always power on the emulator before the target system and power off the emulator after the target system.

Equivalent circuits

The equivalent circuits shown on this page and the next help you understand connection requirements between the emulator probe and your target system.




R/W, TS, BB
+5V
PAL
C=10pF
IIL =-100µA
IIH =100µA
IOL=24mA
IOH=3.2mA
3.16KΩ
8pF
D(31:0), A(31:0),
TT(1:0), SIZ(1:0)
+5V
ABT
C=8pF
IOL=64mA
IOH=-32mA
IL =±50µA
10KΩ
8pF
SC(1:0)
+5V
10KΩ
ABT
Cin=4pF
Iin=±100µA
8pF
BCLK
+5V
10KΩ
Cin=5pF
Iin=±1µA
Z=68Ω
R90
68Ω
t=300ps
68pF
PCLK
Z=68Ω
R24
68Ω
68pF
t=100ps
64783B02

Obtaining the terminal interface

The troubleshooting procedures in this chapter depend heavily on interpretation of command-line prompts that are only seen at the low level terminal interface of this emulator. Therefore, the commands you are told to enter are shown in the terminal-interface form, and the displays you are told to look for are shown in this chapter as they appear in the terminal interface.

To perform the procedures in this chapter, exit out of your emulator/analyzer and invoke it through its terminal interface. Type the following command:

$**telnet** <hostname>

Where <hostname> is the name of the emulator. You could use the Internet Protocol (IP) address (or internet address) in place of the emulator name, if desired. For example:

$**telnet 15.35.226.210**

You should see messages similar to:

```
Trying...
Connected to 15.35.226.210
Escape character is '^]'
```

After you connect to the emulator, you should see a prompt similar to:

R>

This command-line prompt indicates the emulator is in the reset state.

Make sure you begin the emulation session with the emulator in its default state. Initialize the emulator by entering the **init** command:

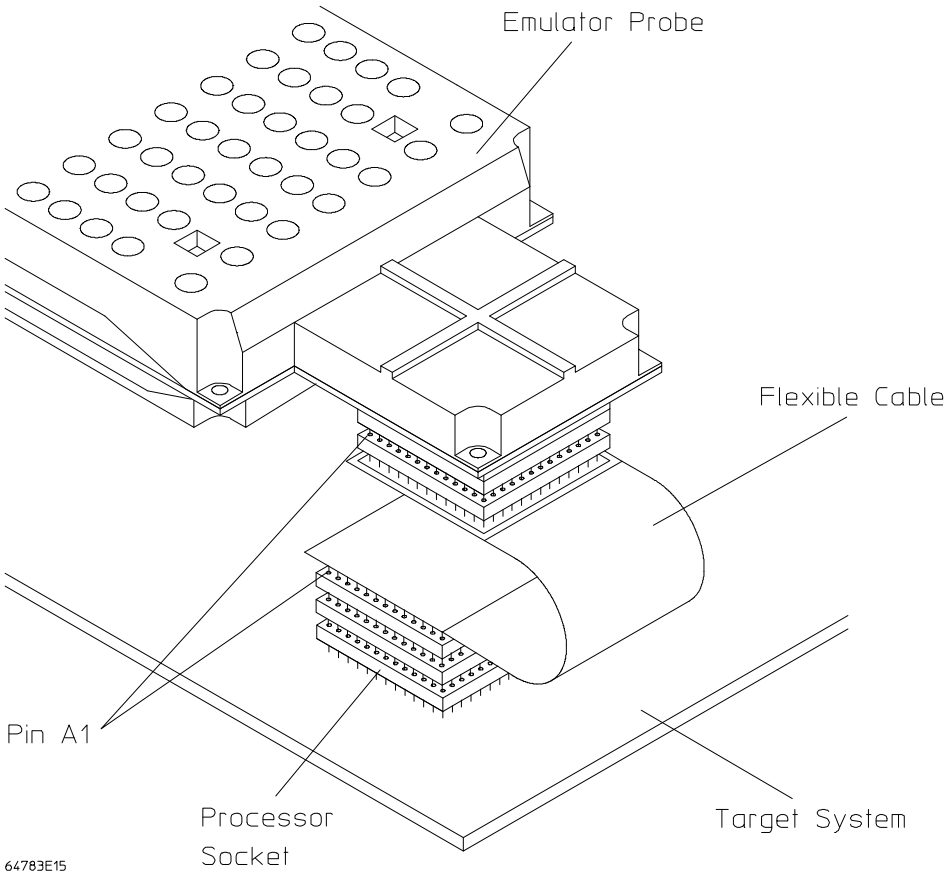
R>init

```
$ Limited initialization completed
```

Connecting the emulator to the target system

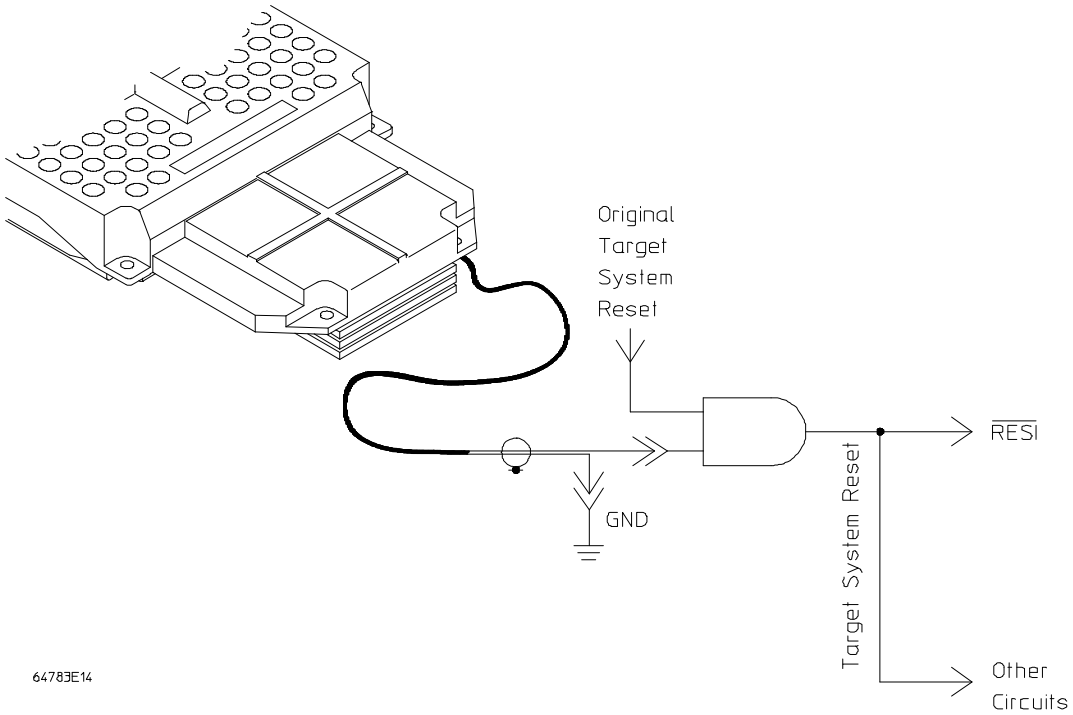
Plugging the emulator into a target system can be difficult because of mechanical constraints. If the mechanical constraints cannot be removed so that the emulator can be plugged directly into the target socket, there are several accessories available to help with the connection. These accessories are:

- Stacking pin protectors.
- PGA rotators, available from Emulation Technology.
- PGA to PGA Flexible Adapter (see below), HP Part Number E3429A.



Unfortunately, these accessories have an electrical impact on your target system. The specifications given for the emulator do not include the impact of these accessories. In addition to delays, the accessories can cause problems with signal quality. Only use these accessories as a last resort.

An optional Reset Flying Lead is provided with the emulator. It can be used to reset the target system when the **rst** command is used. The signal is driven low when the emulator is in it's reset state ("R>" prompt on screen). In addition, the signal will pulse low when a **r rst** or **rst -m** command is issued, if the emulator is not already in the reset state. The signal carried by the Reset Flying Lead is intended to be used to initialize circuitry in your target system that would normally be reset along with the processor (see below).



The example circuit shows the emulator reset signal being ANDed with a target system reset signal to generate a new target system reset signal. This new signal will reset the processor and other circuits on the target system when either the emulator asserts reset, or the target system generates reset.

Verifying Operation Of The Emulator In Your Target System

When connecting an emulator into a new target system, the step-by-step approach described in the remainder of this chapter will help you get your system running most quickly. This is a logical procedure that starts out with the most simple requirements and moves toward compete functionality, allowing for verification of installation at each step of the way. This not only helps debug problems if they arise, but builds confidence that the emulator is functioning correctly in your target system.

To begin, run the performance verification procedure described in Chapter 19, "Installation and Service".

Some additional equipment may be required to make measurements of MC68040 signals. It will help to have an oscilloscope and high speed timing analyzer to use during these procedures. A 250-MHz timing analyzer may be fast enough, but faster is better. The oscilloscope should have a single-shot bandwidth greater than 500 MHz. You may also need to cross trigger these instruments from the emulator. If there are no trigger inputs to the timing analyzer, you can probably use a timing channel. The BNC trigger output of the 64700 emulation card cage provides a rising edge TTL signal.

When making measurements, remember that signals need to be probed at the right place for the measurement being made. The emulator specifications are referenced to the target socket connector on the probe. This is where measurements should be made to verify compliance with the specifications. When probing setup and hold times to circuits in the target system, make the appropriate measurements at the circuits. This will keep connection accessories from impacting the true measurements. Always use ground leads to get the most accurate measurements possible.

Running the emulator configured like the processor

This step uses no emulation monitor, or emulation memory, and does not attempt to control any of the processor signals. For this test, the only emulation feature that is operating is the emulation-bus analyzer. The emulation-bus analyzer is passive, like a preprocessor. The main purpose of this step is to determine whether the loading and timing changes of the emulator impact your target system.

If your target system can run a program without the emulator, do this procedure. Otherwise, go to step 2.

1 Turn on power to the emulator.

2 Check the emulator prompt by pressing the carriage return key.

The prompt should be "p>". A prompt of "->" indicates a software compatibility problem. Correct problems indicated in error messages (seen in the emulator error log) or check the software version using the **ver** command for more information.

3 Configure the emulator by entering the following commands:

cf mon=none
cf cache=en
cf mmu=en
cf ti=en
cf wait=<en,dis>, as appropriate for your target system

4 Set up the emulation-bus analyzer to capture all MC68040 system cycles.

tck -u
tg any
tsto any
tp c
t

5 Execute your program with the command: **r rst**.

This tells the emulator to deassert reset so that the emulator does not interfere with the target system powerup reset.

6 Power on the target system.

7 Verify correct operation.

The target system should run just as if the processor was being used. If your target system performs any I/O, check it to see of your system performs it correctly. If your target system appears to work correctly, allow it to reach its stable operating temperature and test it again.

If the target system appears to work correctly, go to the paragraph titled, "Installing the Background Monitor", later in this chapter. Otherwise, verify operation of the target system as described next.

To verify operation of the target system

Get the prompt by pressing the carriage return key, or use the command **es** to get more information about the emulator status. If the system is working the prompt will normally be "U>", but there are a few situations where the system will be working properly and the prompt will be something different. If the bus is taken away from the MC68040 often or for long periods of time, the emulator can display the "g>" prompt or alternate between "g>" and "U>". If the MC68040 is running code in its internal cache for long periods of time, the emulator may display the "b>" prompt. The emulator may alternate between any of these prompts during normal operation.

All other prompts usually indicate a problem. Even the "g>" or "b>" prompts can indicate a problem. To understand problems indicated by the prompts, you need to know whether bus cycles were executed, how many bus cycles were executed, what type of bus cycles were executed, and whether the target system is still executing bus cycles. You can tell the difference between these conditions by checking the trace status to see if any bus cycles were captured. The analyzer may have states in its internal pipeline that will not be reported until the trace is halted.

```
b>th;ts
  Emulation trace halted
  --- Emulation Trace Status ---
  User trace halted                  <- trace status
  Arm ignored
  Trigger not in memory
  Arm to trigger ?
  States 0 (0) ?..?                  <- number of states captured
  Sequence term 2
  Occurrence left 1
b>
```

If the trace status indicates that the trace was halted, look at the number of states collected to decide how many bus cycles were executed. If the status indicates that the user trace was completed, a large number of states were executed. If this is the case, it may help to take another trace to see if bus cycles are still being executed. Again, view the trace status to determine if bus cycles are executing.

If the "p>" prompt remains after target powerup, check:

- mechanical installation of the probe.
- blown fuses.
- target system power supply voltage.

If the prompt is "c>", mechanical installation may be causing the problem, but the most likely cause is a problem with the clock. Check clock quality. Look at the voltage levels, edges, and duty cycle. If you suspect the clock, compare it to the target system clock without the emulator. If there is a significant difference, you may need to adjust the target system terminations to account for the emulator's termination.

If the prompt is "r>", either the target system never released reset, or the target system reset itself because of some program error condition. If no bus cycles were captured by the analyzer, the target system never released reset. You need to find out which conditions must occur to release reset, and then investigate these conditions to determine why reset isn't being released.

An example of a failure to release reset might be a multicard system where the master card starts the slave cards after verifying that they are installed in the system by reading checksums from their ROMs. If a checksum is not read correctly, reset to the associated slave card is not released. If the emulator interfered with the reading of the checksum, then reset would not be released.

One thing to keep in mind is that the emulator does not trace alternate bus master cycles while it is reset.

If any bus cycles were executed before the reset occurred, then something caused the target system to reassert the reset condition. Usually, this is caused by some type of fault which is detected by the system. This may result from access to a certain address range or because of a watchdog timeout. Refer to "Interpreting the Trace List", later in this chapter, to help you understand what caused the reset.

If the prompt is "b>", and there are no cycles in the trace list, the processor never attempted to run any bus cycles even if other indications show it should have. This could indicate problems with power, clock, or signal transitions, especially the reset signal. Check power supply voltage levels. Make sure the power up is monotonic. Check clock quality. Check that the reset signal meets its required assertion time after power up and clock stabilization. Check signal quality on the reset signal, especially the signal transitions.

If some cycles were captured in the trace list, but no cycles are occuring now, check for setup and hold violations on the processor strobes. All MC68040 signals, except the interrupt lines and reset signal, are synchronous to the clock and have to be valid for all rising edges of BCLK. Check timing inputs to the emulator, such as TA, TEA, and TBI , for setup and hold violations. The "b>" prompt is not a normal condition for the processor when you find no functional reason. It usually indicates that the processor has malfunctioned.

One possible cause of a "b>" prompt is the processor missing the end-of-cycle indication during the cycle of an alternate bus master. The processor monitors the TS signal during alternate bus master activity to see if it needs to intervene in the cycle (snooping). If the processor sees a TS signal but misses the corresponding TA signal, the processor may hang, waiting for this bus cycle to complete, even though the bus was granted to the MC68040 and released.

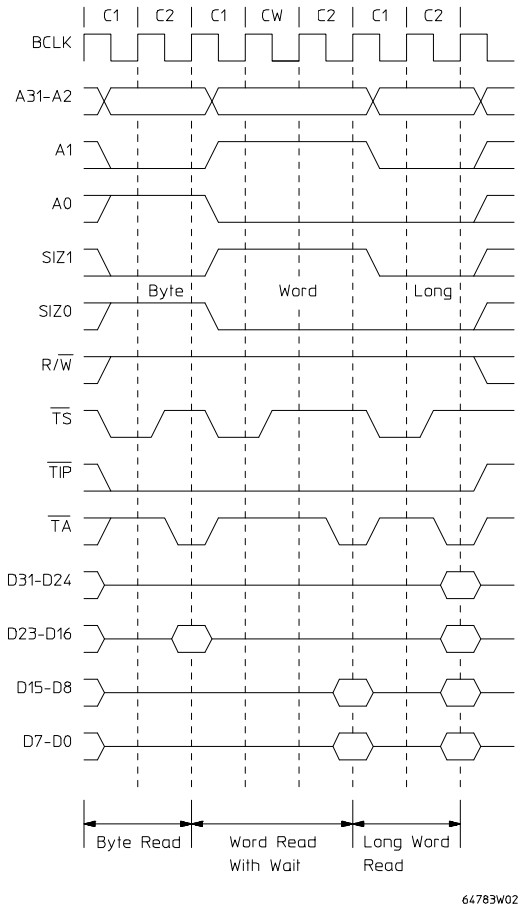
If bus cycles are occuring, then the "b>" prompt only indicates that bus cycles are infrequent. A type of system that would exhibit this behavior would be an interrupt-driven system. When done processing an interrupt, the system could execute a STOP instruction to wait for the next interrupt. If the interrupts were infrequent a "b>" prompt would be displayed.

If the prompt is "w>", the emulator has stopped in the middle of a bus cycle. Get the emulation status; it will tell you the address and the type of cycle.

```
w>es
  M68040--CPU in wait state; 00badad00@sd long read
w>
```

To troubleshoot the above problem, you need to know if the target system provides bus termination for the address. If the answer is no, then the target program must have run incorrectly. The emulation-bus analyzer will have to be used to investigate further. If the answer is yes, then the reason the bus cycle did not complete must be determined, as described next.

There are many reasons why bus cycle interaction between a target system and an emulator may fail. Usually the cause is that the target system missed the start-of-cycle indication from the emulator, or that the emulator missed the cycle-termination indication from the target system. For a better idea of what is going on, refer to the MC68040 bus cycle diagram, below:



A basic MC68040 bus cycle starts with the transfer start signal, TS. The TS signal pulses low for about one clock cycle. Another signal, transfer in progress TIP stays low throughout the cycle, but is not necessarily deasserted between cycles. The end of the cycle occurs when the processor samples a transfer acknowledge TA or a transfer error acknowledge TEA or both on the rising edge of the clock. Because of the nature of these signals, most systems are synchronous to the clock. The typical system will sample TS on the rising clock edge and then generate a TA signal an intregal number of clocks later. Wait states are added to a cycle by delaying when the TA is asserted.

If the emulator is configured for wait states (BCLK >25 MHz), then a compatibility problem with the emulator may be stalling the processor.

```
w>cf
  cf cache=en
  cf mmu=en
  cf mon=none
  cf rrt=dis
  cf ti=en
  cf wait=en                    <- configuration for wait states
w>
```

The emulator requires at least one wait state in all bus cycles when it is configured as above. The emulator does not add this wait state, but will not accept a TA from the target system until after a wait state has been added. If TA is asserted by the target system during the wait state period and is then deasserted before the emulator allows termination, the bus cycle will never complete.

This particular example can be easily duplicated on the demo board by configuring for wait states and interlocking memory to the demo board.

cf wait=en
map 0..0ff eram lock
r rst

If there is no functional reason why the bus cycle would not complete, check the timing relationships between the various bus cycle control signals. Probably the first measurement you will want to make is to see if the setup time of TA to BCLK is within the emulator specification.

If there are no cycles in the trace list, then the processor stopped during the first bus cycle. In this case, it is pretty easy to set up the trace using TS as the trigger because the cycle of interest is the first cycle. If there are only a few cycles in the trace list, the same technique can be used if the oscilloscope or timing analyzer has enough depth.

If there are many cycles in the trace list before the processor stalled, use a different method of triggering. There are a number of different approaches that can be used. The most direct method is to trigger on a condition of TIP low and TA high for a period of time greater than the length of a memory cycle. Another method is to determine if the system always stops at the same address. This address can then be used as the trigger. One drawback to this method is that you may have to probe a large number of signals to get a unique address.

A better way would be to use the emulation-bus analyzer to generate a trigger. Unfortunately, because the cycle never finishes, the emulation-bus analyzer will not capture this address, so something preceding this event must be used as the trigger. Examine the trace list to find a unique event to use as the trigger. Once you have specified the trigger, you need to configure the emulator to drive the trigger out. The real trick to crosstriggering is to correlate the trigger event to the captured data. In this type of measurement, the correlation is easy because the signals of interest stop transitioning shortly after the trigger occurs.

tg addr=00badad00
tp c
tgout trig2
bnct -r trig2
t

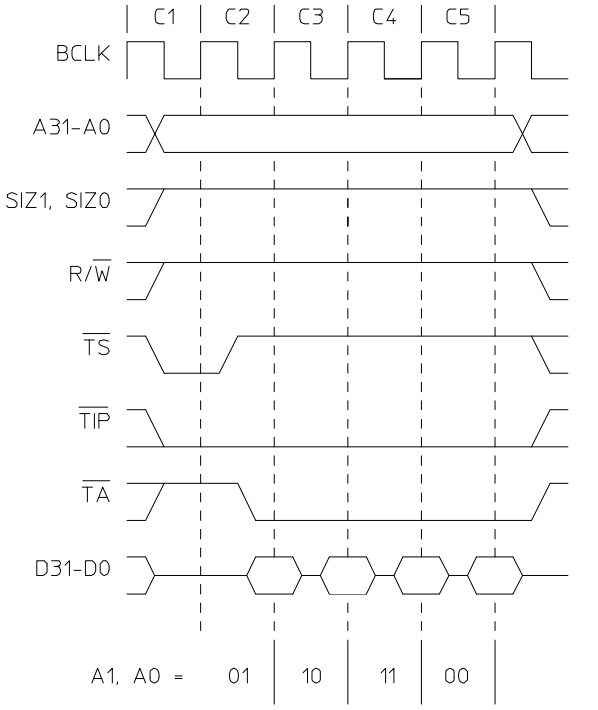
Once you have a trace of the offending cycle, verify that TA is present for a valid rising clock edge, taking into account a wait state if running faster than 25 MHz. If TA looks reasonably correct, verify the setup and hold specifications. If TA occurs but on an invalid clock edge, you may need to make modifications to the target system to ensure that there is at least one wait state in target cycles. If TA is not asserted at all, it could be an indication that the target system missed the TS. Set up your oscilloscope or logic analyzer to make a measurement on your cycle start circuitry to determine why the target system did not respond to the cycle.

If the cycle where processing stops is part of a burst cycle, as indicated by the line access type in the status display, there are several things to check.

```
w>es
  M68040--CPU in wait state; 000000000@sd line read
w>
```

A burst cycle is shown below. The main characteristic of a burst cycle is that there are four data transfers as part of one cycle. The processor puts out an address and asserts TS only once during the cycle. A burst request is indicated by the SIZx signals. The target memory system can inhibit the burst cycle by asserting the TBI signal. If the cycle is inhibited, the timing becomes just like a normal cycle. If the cycle is not inhibited, once TS has been asserted, the process starts sampling TA for

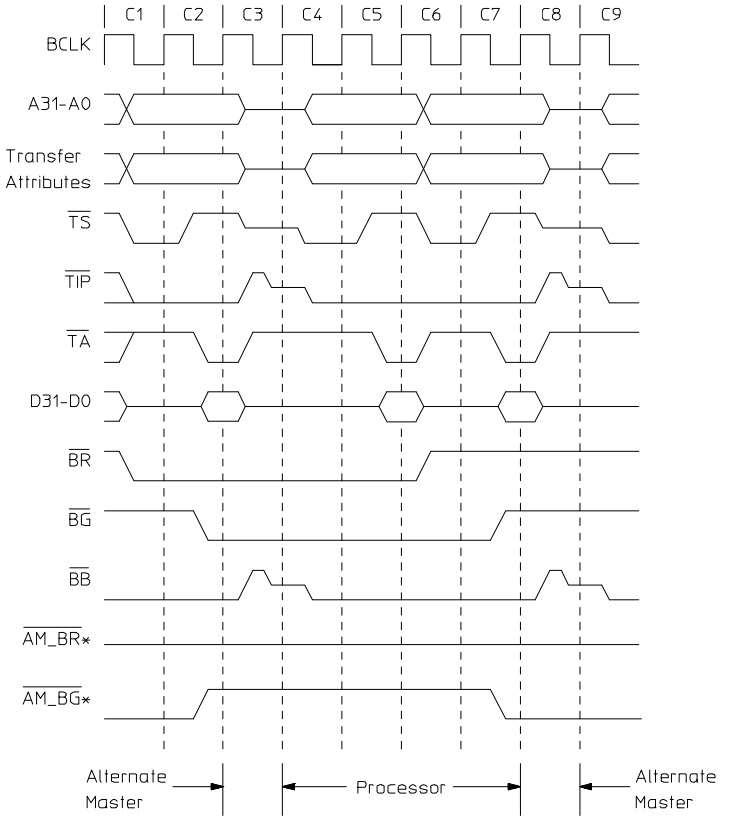
each data transfer. The cycle is not over until the fourth $\overline{TA}$ is received. When the emulator has wait states enabled, a wait state is required between each of the data transfers in the burst cycle. Evaluating the timing is the same as for a normal cycle.



Note: The selected device increments the value of A3 and A2.

64783W03

If the prompt is "g>" and there are no cycles in the trace list, the target system never gave the bus to the processor. Check the bus arbitration signals for proper functionality and timing. Refer to the bus arbitration diagram below. Remember that the analyzer does not trace alternate bus master cycles while the emulator is reset, but it does once the emulator is running.



* AM indicates the alternate bus master.

64783W01

When trying to determine why the bus is not being granted to the processor, you will need to determine why either the bus arbitration circuitry or an alternate bus master is not behaving correctly. The processor is not the bus master; therefore, it requests the bus with BR and waits for the target system to grant the bus with BG. The processor then waits for the BB line to be deasserted, indicating an idle bus, before taking control of the bus. The processor will not request the bus until after the reset line has been deasserted.

If the bus is requested by the processor, but it is not being granted check the bus arbitration signals BB, BG, and BR. If the bus is granted, but never becomes idle, the alternate bus master may be stuck in the middle of a cycle. Check the cycle strobes TS, TA, and TEA. These strobes do not have to be asserted during alternate master accesses, but if TS is shown to the processor, then TA needs to be shown to end the cycle. While the processor is reset, the only item of concern is signal quality.

If some cycles are shown in the trace list, but no cycles are occuring now, the processor executed some cycles before getting stuck in a DMA cycle. Examine the bus arbitration signals and cycle strobes around where the target system gets stuck. Use the same techniques to set up a trigger as were described for measuring a bus cycle that stops before it is complete.

If there are bus cycles occuring, then the "g>" prompt indicates that a high percentage of the bus activity is by alternate bus masters.

Interpreting the trace list

There are some cases where a problem caused by an errant bus cycle does not show up until many cycles later. The emulation-bus analyzer must be used to track back through the sequence of events to the faulty bus cycle. Data problems will often behave like this, but there may be other causes.

If the "h>" prompt is shown, indicating a double bus fault, and if there are only two states in the tracelist, this indicates a problem with the fetching of the initial vectors.

```
h>tl
   Line   addr,H    68040 Mnemonic
  -----   --------  ----------------------------------------
      0   00000000   $00000000     sdata long read
      1   00000004   $000BADAD     sdata long read
      2
h>
```

The first two cycles in the trace list are the initial stack pointer and the initial program counter. The initial program counter must be even or the processor will immediately double bus fault. You should verify that the data captured by the analyzer is what is expected.

If the data for the vectors is wrong, a trace should be set up to check for access problems during the fetch of the initial vectors. If the data is completely incorrect, suspect an address or strobe timing problem. If only a few bits are wrong or if the data in the trace is correct, suspect a data timing problem.

If there are a lot of cycles in the tracelist, you need to start from the end and work backwords to understand what caused the double bus fault. If the trace was completed before the processor stopped, modify the trace specification to "trigger on nothing" so that the last bus cycles that were run can be captured. Wait until the emulator status shows a double bus fault, and then halt the trace.

tg never

reset the target system

es
th
tl -20

```
h>tl
   Line   addr,H    68040 Mnemonic
  -----   --------  ----------------------------------------
    -16   00000008   $4AFC0000     sprog long read     <- illegal inst
    -15   0000000c   $000BADAD     sprog long read
    -14   00000010   $000BADAD     sprog long read
    -13   00000014   $00000000     sprog long read
    -12   00000018   $00000000     sprog long read
    -11   000000ee   $----0010     sdata word write    <- illegal inst stack
    -10   000000ea   $----0000     sdata word write
     -9   000000ec   $0008----     sdata word write
     -8   00000010   $000BADAD     sdata long read     <- odd vector
     -7   000000e8   $2700----     sdata word write
     -6   000000e4   $000BADAC     sdata long write
     -5   000000e2   $----200C     sdata word write    <- address error stack
     -4   000000de   $----0000     sdata word write
     -3   000000e0   $0008----     sdata word write
     -2   0000000c   $000BADAD     sdata long read     <- odd vector
     -1   000000dc   $2700----     sdata word write
h>
```

A double bus fault occurs when the processor encounters an exception that prevents processing of a previous exception. An example of a double bus fault is shown above. This original exception occured because the target system tried to execute an illegal instruction. During processing of the illegal instruction exception, the processor encountered another exception.

This exception was an address error caused because the vector supplied for the illegal instruction handler was odd. The double bus fault occured when the vector supplied for the address error handler was also odd. Other things that can cause a double bus fault are bus errors that occur during exception stacking or vector fetch. Keep in mind that bus errors can happen because the the target system asserts TEA or because of an access violation caused by the MMU.

Once you have found the cause of the double bus fault, you need to determine the root cause of the problem. In some cases, the exception is a normal part of execution, but the subsequent faults indicate a problem. In some cases, the first fault indicates a problem directly, such as when the program has already malfunctioned, and the fault is caused by an unintentional accesses.

At this point, the problem is to find the faulty bus cycle that eventually caused a recognizable problem. The same situation exists if the processor stops execution at an address that should not have been executed, or if a program is simply running code incorrectly.

There are really only two ways to go about determining what is wrong. One is to try to trace back the terminal error condition to a faulty bus cycle. The other is to start at the beginning of the trace, or at some other known point, and work forward, comparing the trace to the execution that was expected while looking for the point where execution first becomes unexpected. A listing of the program or a tracelist captured by a preprocessor could be used for this comparison.

When you find a suspected bus cycle, set up a trigger on it so that you can make a timing measurement on the cycle. When looking for clues or shortcuts to the problem, keep in mind that a system is usually made up of many different types of memory devices: ROM, EEPROM, SRAM, DRAM, and peripheral ports. Each of these devices may have different timing characteristics. Also, keep in mind that unique characteristics of a bus cycle, such as size, transfer type, number of wait states, and bursting may result in unique timing requirements.

Fixing timing problems

When a timing problem is identified, you must decide how to fix it. First, examine the signal to make sure that signal quality is not affecting the timing. Look for AC or DC drive problems or reflections caused by transmission line problems. If you can find no other solution to the problem, you may have to lower the clock speed.

If the timing problem only occurs during data accesses, another possible solution is to add wait states to the memory access. This assumes that the problem is with the amount of time it takes to access the memories in the system and is not a problem with a setup time to a synchronous circuit. A good indicator of this type of problem is when the data setup time to the emulator is being missed. One point of caution: the emulator, when configured with wait states (**cf wait=en**), does not add a wait state to target accesses. The target system is responsible for adding the wait state.

Another possible solution to data access problems is to use faster memories while using the emulator.

Installing the emulator in a target system without known good software

If you do not have a program in ROM on your target system that you can run to electrically test the emulator, you will need to create a test environment. The initial step of this is to use the emulator's dual-port memory to install a simple program that will run from reset. To do this, proceed as follows:

1 Turn on emulator power.

2 Check the prompt by pressing the carriage return key.
The prompt should be "p>". A "->" prompt indicates a software compatibility problem. Correct problems indicated in error messages or check the version "ver" for more information.

3 Configure the emulator by entering the following commands:

cf mon=none
cf cache=en
cf mmu=en
cf ti=en
cf wait=<en,dis>, as appropriate for the target system

4 Map dual-port memory with the following command:

map 0..0fff eram dp,lock

This maps a block of emulation memory starting at address 0 so that the reset vectors will be accessed from this block. The block is configured to be interlocked to the target system strobes because all systems must have some memory that responds at address 0 to operate.

5 Load a program with the following commands:

mo -ax -dl
m 0=0f00,100
mo -dw
m 100=60fe

This sets up the reset vectors ISP=0f00 and IPC=100. It then loads the most simple program imaginable: jump to self.

6 Setup a trace to capture all MC68040 cycles, as follows:

tck -u
tg any
tsto any
tp c
t

7 Execute **r rst**.

This tells the emulator to deassert reset so that the emulator does not interfere with the target system powerup reset.

8 Power on the target system.

9 Verify correct operation.

The target system should run the same as when the target processor was being used. The first indication of whether or not your target system is working is to see if your program performs any I/O that can verify correct system operation. If your target system appears to work initially, allow it to reach normal operating temperature before concluding that target system operation is as it should be.

If the target system appears to work properly, go ahead to the paragraph titled "Installing a Monitor". If you suspect problems, return to "Verifying System Operation" in the previous paragraphs. Keep in mind that the emulator must receive strobes from the target system for emulation memory accesses to complete. Also, because these cycles are from internal emulation memory, the data on the target system will not be the same as what the processor sees. If you think that there are problems with emulation memory data, check the clock speed configuration; the emulator is designed to give correct data at all speeds of operation.

Installing Emulator Features

Once the emulator is transparently running in the target system, it is time to start adding other emulator features. Dividing the installation of features into two tasks is the easiest way to debug problems. The monitor is the facility that provides the majority of the emulator's features, but some features like the reset circuitry do not require the monitor. The first feature to be installed does not depend on the monitor.

Evaluating the reset facilities

Now is a good time to use the emulator to find out how the emulator reset interacts with your target system. The first question to answer is whether or not the emulator reset command is adequate to reset your target system. Perform the following steps:

1 Run your target program by following the procedure in the previous steps.

2 Reset the emulation processor and run your program using the emulator commands:

r rst

Note that the "r rst" command pulses the processor reset line.

3 Verify correct operation.

If your program does not run correctly after performing the above procedure, your target system has other circuitry besides the processor that must be reset. The emulator only resets the emulation processor when it responds to a reset command. Other circuitry on your target system does not get reset. The following sequence determines if an additional reset circuit is required.

4 Run your target program following the procedure in the previous steps.

5 Reset the emulation processor and run your target program using these emulator commands:

rst

Reset the target system using whatever facility is available.

r rst

6 Verify correct operation of the target system.

An example of a target system that requires an additional reset circuit is one that normally has RAM starting at address 0, but for the first two bus cycles after reset, maps ROM to this area instead to provide the inital vectors. If this remapping does not occur, the system will attempt to fetch these vectors out of RAM, which will fail.

For systems that require additional circuitry to be initialized by reset, a reset output from the emulation probe (called reset flying lead) is provided. This reset flying lead can be connected into your target circuitry to eliminate the need for an additional step to reset circuitry in your target system. This allows the whole reset procedure to be controlled by the emulator, automatically.

One additional thing to keep in mind is that your target system can initiate a reset without the knowledge of the emulator. A reset that is initiated by your target system will reset the emulator. If the emulator was running your target program at the time of the reset, then when your system releases reset, the emulator will run as if an **r rst** command had been issued. If the emulator was executing in the monitor at the time of the reset, it will return to the monitor when the reset is released.

Another resetting method that may provide more convenience than the first method requires use of the monitor. This method works well for target systems such as those in the example above. This method resets the emulator into the monitor instead of running the target system program immediately. Once in the monitor, the initial stack pointer and initial PC can be loaded into the appropriate registers, and then a run of the target program can be initiated. This method will be illustrated in the next section.

Installing the background monitor

The emulator allows you to choose between use of a background and foreground monitor, but the choice is really predetermined by which of the MC68040 features you will be using.

The background monitor does not support use of the MMU, the caches, or DMA. Therefore, the background monitor is only useful in the most simple systems, or to provided a mechanism for testing target hardware, or to further evaluate the integration of the emulator with your target system.

The background monitor does not show cycles to your target system. It accomplishes this by blocking the TS and TIP signals. Therefore, the background monitor is transparent to your target system. Even though the background monitor does not show its cycles to the target system, the initial vector fetch cycles are shown to the target system and interlocked with the target system strobes. Cycles not shown to the target system are called background cycles. All other cycles are called foreground cycles.

Resetting into the background monitor

There are three ways to initially get into the background monitor. The first of these ways is to enter the monitor from reset. Perform the following command sequence to enter the monitor:

1. Reset the emulator and the target system if necessary using any reset procedure you determined to work adequately.

2. Configure the emulator by entering the following commands:

   cf mon=bg
   cf monkaa=none
   cf cache=dis
   cf mmu=dis
   cf ti=en
   cf wait=<en,dis>, as appropriate for the target system

3 Set up a trace to capture all MC68040 cycles, including background monitor cycles, by entering the following commands:

```
tck -ub
tsto any
tg any
t
```

4 Execute the command: **rst -m**. This tells the emulator to release reset, but enter the monitor.

5 Verify that the emulator is in the monitor.

The prompt should be "M>", indicating that operation is in the monitor. There is not much that can go wrong up to this point because everything required has been previously verified.

If you see the following error messages, something went wrong during the initial vector fetches from the target system. Check these cycles for problems.

```
!STATUS  170! Emulator terminated hung bus cycle: 000000000@sd long read
!STATUS  170! Emulator terminated hung bus cycle: 000000004@sd long read
```

If you see a "g>" prompt, the background monitor is not compatible with this type of target system. Go to the paragraph titled "Installing the Foreground Monitor".

If you get the "?>" prompt or something other than the "M>" prompt, this indicates something went wrong with monitor operation. This may indicate problems with the clock or reset signals. Because the emulator provides all control signals for the background monitor, typically problems are with signals that can prevent the processor from running bus cycles.

Dealing with keep-alive circuitry while using the background monitor

Another thing to watch for when using the background monitor is the triggering of a target system keep-alive circuit because monitor bus cycles are hidden. Depending on how a keep-alive circuit operates, the monitor may cause a problem. The symptoms for different keep alive circuits may not show up in the same way.

Keep-alive circuits that monitor accesses on the bus or require a certain address to be accessed probably will fail when you use the background monitor. Keep-alive circuits that make sure bus cycles complete will not fail. If the keep-alive circuit generates a bus error or an interrupt, the monitor will not be affected immediately. If the keep-alive circuit asserts reset instead, monitor operation will be affected immediately, although there may be no apparant symptoms if reset is only asserted temporarily because the monitor will be reentered as soon as reset is deasserted.

If you suspect a problem with a keep-alive circuit, there is a configuration option that can make the background monitor periodically cause a read access to a particular address. If you do need a particular address to be read for the keep-alive function, make sure the address you give will respond with memory strobes when accessed.

cf monkaa=0deadad0

Retry the reset into monitor with this configuration enabled. If there is any sort of problem with the keep-alive access, it will probably show up as a wait state at the keep-alive address. If this happens, check the timing on that particular cycle. The keep-alive address may respond with a bus error without adversly affecting monitor operation.

Testing memory accesses with the background monitor

Once the background monitor looks like it is running properly, you can use it to test accesses to different ranges of memory in your target system. This may be an easier way to diagnose problems than by running a program that accesses each memory range. It is also easy to check accesses of different sizes using the monitor.

mo -ax -dl
m 0badad=12345678

When accesses to your target memory do not execute exactly right, the monitor attempts to diagnose these problems and resolve them so the monitor program does not malfunction. However, the monitor does not read back write cycles to check the integrity of the data written. When testing memory accesses, the data should be checked to make sure that it is correct.

M>m 0badad
0000badad ffdf00ff

If your target memory does not respond to a bus cycle, the monitor will force termination of the cycle and report this error message:

!STATUS 170! Emulator terminated hung bus cycle: 0000badad@sd word read
!ERROR 700! Target memory access failed

Or, if the target system responds with a bus error for this memory access, the monitor will report that information:

!ERROR 170! Target bus error: 0000badad@sd
!ERROR 700! Target memory access failed

Running a program from the background monitor

Once you are satisfied that the monitor is working and that memory in your target system can be accessed correctly, you can use the monitor to run your target program. Proceed as follows:

1 Reset into the monitor.

2 Load a program, if necessary.

3 Initialize the initial stack pointer and initial program counter.

reg isp=<initial ISP>
reg pc=<target program starting address>

If these values are not known, they can be found by taking a trace of the program running from reset, as was done in the previous sections.

4 Take a trace of the program running, using the following commands:

tg addr=<long aligned target program starting address>
t

The trigger address must be long aligned because the MC68040 always fetches instructions as long words from long-word boundaries.

5 Run the program with the command:

r

6 Verify correct operation of the program.

Assuming that the program ran without the monitor, the stack is most likely the cause of any problems you see. The monitor runs the program by creating a stack in foreground memory at the location indicated by the initial stack pointer. The monitor then initiates an RTE, which starts the target program running. The following trace list is an example showing correct operation:

```
 Line   addr,H    68040 Mnemonic
-----   --------  ----------------------------------------
   -4   000000f0   $00------ mon sdata byte read
   -3   000009b4   $4E714E71 mon sprog long read
   -2   000000ec   $000a007C     sdata long read     <-unstack
   -1   000000e8   $27000000     sdata long read     <-unstack
    0   00000008   $000060FE     sprog long read      <-target program
    1   0000000c   $000BADAD     sprog long read
```

If the monitor detects problems with the stack pointer (the stack pointer must be even), or if the monitor has a problem accessing the stack memory, an error message is issued. Additionally, the monitor checks to make sure that the stack has been written correctly before exiting. Problems are indicated by the error messages listed below.

From this point on, most of the problems will be discussed from a functional point of view instead of a parametric point of view. If any of the functional problems discussed below identify a problem that looks parametric, use the debugging techniques of the previous procedures to isolate the problem.

!ERROR 151! Interrupt stack pointer is odd or uninitialized
!ERROR 610! Unable to run

This message indicates that the stack pointer is invalid. Only word-aligned stack pointers are allowed with the emulator. If this error is seen, the run will not be attempted.

!ERROR 170! Target bus error: 0000000e8@sd
!ERROR 610! Unable to run

This message indicates a bus error occured during the stack write. This behavior could be caused by putting the stack in a memory range that responded with bus error for all accesses, or bus error on write accesses. Or, it could be caused by putting the stack where nothing responds, and the bus error is the result of a timeout. Keep in mind that the stack grows down from the initial stack pointer.

!STATUS 170! Emulator terminated hung bus cycle: 0000000e8@sd long write
!ERROR 610! Unable to run

This message indicates that the stack is in an address range that did not respond with a memory strobe. Make sure that the stack is placed in valid memory.

!ERROR 151! Interrupt stack is not located in RAM: 0000000e8@sd
!ERROR 610! Unable to run

This message indicates that the stack memory was not writeable. Check to make sure that the stack is placed in RAM.

If the target program appears to start at the wrong address, or if there is some other problem, the stack can be decoded to see if the correct information is present there. The stack above is interpreted as follows: The initial stack pointer is defined to point to the next available stack location. Therefore the exit stack starts four words below the initial stack pointer.

```
ISP-8 -> Status register =     2700
ISP-6 -> Program Counter = 0000000a
ISP-2 -> Vector Offset   =     007C
```

The monitor is always exited using the FOUR WORD STACK frame, and the monitor always uses 07C as the vector offset. When running a program from the monitor after entering from reset, the powerup status word of 2700 is used. Therefore, the only difference you will see in this stack frame will be because of different initial program counter values.

The procedure of setting the initial stack pointer and initial program counter can be automated by using the initial vectors configuration question to define these values.

cf rv=<initial ISP>,<initial PC>

Once this configuration has been set up, the following reset sequence may be useful on systems that remap memory to provide reset vectors similar to the example in the "Evaluating the Reset Facility" section.

rst -m
r

Breaking into the background monitor

The next thing to try with the background monitor is to see if you can break into it from your target program. The emulator uses a nonmaskable interrupt (interrupt 7) to break into the monitor. The interrupt is generated in such a way as to not interfere with any interrupts pending in your target system. The resulting interrupt acknowledge cycle is not shown to the target system. The associated stacking is in foreground memory at the location determined by the interrupt stack pointer. If the target system program is running in Master mode, there will also be stacking on the master stack.

A vector fetch occurs sometime during or after stacking; it is also shown to the target system. The emulator provides the data for this vector fetch to correctly run the background monitor. After stacking and the vector fetch are completed, the emulator transitions into the background monitor. The background monitor may access foreground memory during its operation.

While the emulator is in the background monitor, no target interrupts are serviced. The interrupt signals from the target system are ignored while in the background monitor. The emulator will not respond to these signals in any way while in the monitor. If the signals are still present when the monitor is exited, they will be serviced according to normal interrupt priorities.

Entry into the background monitor can be traced by using the following trigger specification:

```
tck -ub
tp c
tg stat=11xxxxxxxx1x111xy
t
b
```

```
 Line   addr,H    68040 Mnemonic
-----   --------  ----------------------------------------
   -2   00000008   $60FE0000     sprog long read
   -1   0000000c   $000BADAD     sprog long read
    0   ffffffff   $------FF mon int7 ack             <-acknowledge
    1   000000ee   $----007C     sdata word write      <-stack format
    2   000000ea   $----0000     sdata word write      <-stack PC high
    3   000000ec   $0008----     sdata word write      <-stack PC low
    4   0000007c   $0000069C     sdata long read       <-vector fetch
    5   000000e8   $2700----     sdata word write      <-stack SR
    6   00000698   $0012FFFF mon sprog long read       <-monitor
    7   0000069c   $11FC001F mon sprog long read
```

If you have problems trying to break into the monitor, the most likely causes are the values of the stack pointers, or the vector base register does not point to valid memory. Any bus errors that occur during monitor entry will cause the break to fail. If any stacking or vector fetch cycles are not terminated, the monitor will terminate them by force. If this happens, the PC and SR may be displayed incorrectly by the monitor. The same problem can result from stack memory that is not writeable. Neither condition will inhibit entry into the monitor, but the target state will be corrupted.

Exiting the background monitor

If the procedures described in the preceding paragraphs gave satisfactory results, you should be able to resume execution of the target program. You may want to take a trace of the monitor exit procedure to verify that it is completed correctly.

r

If the target system and emulator do not work correctly after exiting the background monitor, the problem may be because your target system is real-time sensitive. If interrupts that needed to be serviced to keep the target system running were delayed by the monitor, things such as data overrun could cause problems in the target system. If you suspect such a problem, use the foreground monitor.

Software breakpoint entry into the background monitor

The background monitor can also be entered via a software breakpoint. The emulator will respond to any software breakpoint instruction in the code if breakpoints are enabled, regardless of whether the breakpoint was inserted by the emulator or not. Breakpoints are enabled by the following command.

bc -e bp

Set breakpoints only on the initial word of an instruction; otherwise, they will not be executed, and might alter an instruction, unintentionally. The emulator can place a breakpoint using one of two methods. By default, the emulator will attempt to modify memory to insert a breakpoint instruction at the address specified. If the memory at the address specified is ROM or cannot be modified for some other reason, special hardware resources on the emulator will interject a breakpoint instruction when that address is fetched.

b
bp <instruction address>

If you suspect a problem occurred during the setting of the breakpoint, you can use the analyzer to watch the breakpoint being set. The easiest way to do this is to store-qualify your trace on the address where you are setting the breakpoint. The trace list will only contain a cycle or two, but you can see what happened when the emulator accessed this address.

tg any
tsto addr=<instruction address>
b
bp <instruction address>

```
 Line   addr,H    68040 Mnemonic
-----   --------  ----------------------------------------
    0   00000008   $FFFF----     sdata word read
    1   00000008   $FFFF----     sdata word read
    2   00000008   $FFFF----     sdata word read
    3   00000008   $484F----     sdata word write  <- breakpoint write
    4   00000008   $FFFF----     sdata word read  <- verify
    5   00000008   $FFFF----     sdata word read
    6
```

When a software breakpoint instruction is executed, the processor initiates a breakpoint-acknowledge cycle. This cycle signals the start of an entry into the monitor. From this point on, stacking and the vector fetch procede the same as for a break entry. Unlike the interrupt-acknowledge cycle, the breakpoint-acknowledge cycle is shown to the target system.

tsto any
tg stat=11xxxxxxxx1x000xy
t
r 8

```
 Line   addr,H    68040 Mnemonic
-----   --------  ----------------------------------------
   -4   00000008   $484F0000     sprog long read      <-bkpt fetch
   -3   0000000c   $000BADAD     sprog long read
   -2   00000010   $000BADAD     sprog long read
   -1   00000014   $00000000     sprog long read
    0   00000000   $41------     bkpt ack (buserror)  <-acknowledge
    1   000000ee   $----0010     sdata word write     <-stack format
    2   000000ea   $----0000     sdata word write     <-stack PC high
    3   000000ec   $0008----     sdata word write     <-stack PC low
    4   00000010   $00000690     sdata long read      <-vector fetch
    5   000000e8   $2700----     sdata word write     <-stack SR
    6   00000690   $11FC0004 mon sprog long read      <-monitor
    7   00000694   $01186000 mon sprog long read
```

The only unique portion of a breakpoint entry is the breakpoint-acknowledge cycle so any problems that you see will probably be related to this cycle. Because the emulator internally responds to this cycle, it is not necessary for the target system to respond to it. If the target system does respond to this cycle with any wait states, the emulator may become out of sync with the target system because the emulator terminates this cycle immediately. If this were to cause a problem, it would show up on the cycle immediately following the breakpoint-acknowledge cycle.

Stepping with the background monitor

The last feature of the background monitor which needs to be evaluated is the single-stepping facility. The emulator uses a combination of the processor trace facility and a nonmaskable interrupt to reenter the monitor after executing exactly one instruction.

b
tsto any
tg stat=11xxxxxxxx1x111xy
t
s

```
000000008@s  -                BRA.B     $00000008
PC = 000000008@s
```

When a step command is issued, the emulator sets the trace bits in the SR and then performs a normal monitor exit. The emulator then forces a break to return to the monitor. A typical trace of a single step is shown below:

```
 Line   addr,H    68040 Mnemonic
-----   --------  ----------------------------------------
  -17   000009b0   $4E714E71 mon sprog long read
  -16   000000f0   $00------ mon sdata byte read
  -15   000009b4   $4E714E71 mon sprog long read
  -14   000000ec   $0008007C     sdata long read     <- unstack
  -13   000000e8   $A7000000     sdata long read     <- unstack
  -12   00000008   $60FE0000     sprog long read     <- stepped inst
  -11   0000000c   $000BADAD     sprog long read
  -10   00000008   $60FE0000     sprog long read
   -9   0000000c   $000BADAD     sprog long read
   -8   000000ec   $00000008     sdata long write    <- trace stack addr
   -7   000000ea   $----2024     sdata word write    <- trace stack format
   -6   000000e6   $----0000     sdata word write    <- trace stack PC up
   -5   000000e8   $0008----     sdata word write    <- trace stack PC low
   -4   00000024   $00000000     sdata long read     <- trace vector fetch
   -3   000000e4   $A700----     sdata word write    <- trace stack SR
   -2   00000000   $000000F0     sprog long read     <- trace prefetch
   -1   00000004   $00000008     sprog long read     <- trace prefetch
    0   ffffffff   $------FF mon int7 ack            <- break acknowledge
    1   000000e2   $----007C     sdata word write    <- break stack format
    2   000000de   $----0000     sdata word write    <- break stack PC up
    3   000000e0   $0000----     sdata word write    <- break stack PC low
    4   0000007c   $0000069C     sdata long read     <- break vector fetch
    5   000000dc   $2700----     sdata word write    <- break stack SR
    6   00000698   $0012FFFF mon sprog long read     <- monitor
    7   0000069c   $11FC001F mon sprog long read
```

At the end of the execution of the first target program instruction, the processor takes a trace exception. Stacking for this trace exception commences and at some point, the trace vector is fetched. Once stacking for the trace is complete, the processor prefetches from the address of the trace handler, but these instructions are

never executed because the processor immediately starts interrupt processing. The interrupt processing proceeds the same as in a normal break.

Before exiting for a step, the monitor checks to make sure that the trace vector is valid and that it points to accessable memory. If the vector is not even, or if the memory it points to responds with a bus error or hangs, the emulator temporarily modifies the trace vector to point to the start of the vector table. Because the instructions of the trace handler will not be executed, the content of the address locations is not important.

If the emulator modifies the trace vector, the following status message is given:

!STATUS 155! Vector table modified for single stepping

If the emulator finds it must modify the trace vector for single stepping to complete, but the modification attempt fails, an error message similar to the following is displayed:

!ERROR 170! Target bus error: 0ff800024@sd
!ERROR 156! Unable to modify trace vector to ff800000h for single stepping
!ERROR 680! Stepping failed

If this error occurs, the vector table must be modified so that the trace vector contains an address that points to accessable memory. If the vectors are in ROM, perhaps the memory can be copied into emulation memory where you can modify it.

One way to watch what the emulator is doing during a step, is to set up the analyzer to trace only foreground cycles and to store everything. This lets you watch the emulator check and possibly modify the trace exeception vector. Use the following commands:

tck -u
tsto any
tg any
t
s

The emulator may experience problems when stepping over instructions that modify the VBR. This is because the check of the trace exception vector is made using the old VBR value, but the actual stacking will use the new value of the VBR. If the new VBR value changes the trace exception vector to something that would require modification, then stepping can fail.

!ERROR 680! Stepping failed

When stepping over instructions that cause the processor to take exceptions, the trace list can look very different. Most exceptions preempt the trace exception until after their exception handler runs. Other exceptions (like TRAP, CHK and CHK2) create their stack frame and then take the trace exception. Any exceptions cause the step trace list to look different. In all cases, the monitor is still entered through the interrupt 7 exception.

For all exceptions except TRAP, CHK, and CHK2, the trace stack frame will be missing when the monitor is entered. Instead of using the trace stack frame, the exception stack frame will be used. The emulator detects that and issues an error message that says stepping failed. This error message does not actually indicate a problem with emulator stepping; it just indicates that an exception was hit. The emulator is stopped at the starting address of the exception handler, and stepping can be resumed.

The TRAP, CHK, and CHK2 exceptions will have an additional stack frame when the monitor is entered. The exception stack frame will precede the normal trace and interrupt stack frames. These exceptions do not cause the monitor to issue an error message so multiple steps will not stop on this type of exception.

Installing the foreground monitor

The foreground monitor supports all features of the emulator, but imposes on your target system more than the background monitor. The foreground monitor occupies a 4-Kbyte block in your target memory space. The emulator provides memory for this 4-Kbyte block, but the target system cannot use this address range for anything. The cycles strobes TS and TIP are shown to the target system during foreground monitor cycles. The monitor needs to be placed in an address range where it will not interfere with target system operation.

If the monitor is placed in an address range where the target system responds with a TA, interlock the monitor to the target strobes. The target system must not respond with TEA for this address range. If the monitor is placed in an address range where the target system does not respond with any strobes, do not interlock the monitor. If in doubt, interlock the foreground monitor to the target system. It will be obvious if this is the wrong thing to do because the monitor will stop operating immediately.

If the MMU is being used, the monitor must be placed in an address range that is translated logical=physical, and is writeable for supervisor program and data. If the memory management scheme is dynamic, the monitor page must be resident at all times. In addition, any pages required for stacking or vector fetches must also be resident.

If there is not a suitable address range in which to put the monitor, the system protection schemes may need to be modified to create a place for the monitor. This may be as simple as adding an entry to the MMU tables, or it may require modifying a hardware protection scheme to allow placement of the monitor.

Besides adding special requirements to the placement of the monitor, the MMU impacts many operations of the emulator and processor. When the MMU is on, the emulator can access both physical and logical memory. The emulator also provides commands to examine the MMU tables.

With the MMU on, there are new problems added to the task of connecting the emulator probe into a target system. Besides making sure that the restrictions noted above are complied with, interpreting the trace list becomes more difficult. You also need to keep in mind the distinctions between logical and physical memory accesses when accessing memory. Finally, you need to find out whether you need to load your program before the MMU is running or while it is running.

The foreground monitor, in contrast to the background monitor, allows servicing of interrupts. When the foreground monitor is not busy performing some action, interrupts are allowed. The interrupt routine must return control to the monitor within a reasonable period of time or the monitor may timeout if it attempts to do something. The level of interrupt that can be recognized by the monitor can be controlled through a configuration question:

cf monint=0

Resetting into the foreground monitor

If you have successfully established operation of the background monitor, or if you have decided that you cannot use the background monitor because you need certain MC68040 features, then it is time to evaluate the foreground monitor. The first thing to do is to enter the foreground monitor from reset. Perform the following command sequence to enter the monitor.

1. Reset the emulator, and the target system if necessary, using whatever reset procedure you determined to work.

2. Configure the emulator, as follows:

   cf mon=fg
   cf monaddr=addr as appropriate for the target system
   cf monlock=<en,dis> as appropriate for the address mapping
   cf monint=0
   cf cache=en
   cf mmu=en
   cf ti=en
   cf wait=<en,dis> as appropriate for the target system

3. Set up a trace to capture all MC68040 cycles. Background cycles do not need to be traced to see foreground monitor operation.

   tg any
   tsto any
   tck -u
   t

4. Execute the command: **rst -m**

This tells the emulator to release reset, but enter the monitor.

5 Verify that the emulator is in the monitor.

The prompt should be "M>", indicating correct operation in the monitor. There is not much that can go wrong up to this point since everything required has been previously verified.

If you get the following error messages, a failure occurred during the initial vector fetches from the target system. Check these cycles for problems.

!STATUS 170! Emulator terminated hung bus cycle: 000000000@sd long read
!STATUS 170! Emulator terminated hung bus cycle: 000000004@sd long read

If you get a "w>" prompt for a monitor address, you may have incorrectly interlocked the monitor to the target system. If the monitor was correctly interlocked, check to see if there is a timing problem with the target terminations for the monitor address range.

If you get the "b>" prompt or something other than the "M>" prompt, suspect a failure in monitor operation. These prompts may indicate problems with the clock or reset signals. If the monitor is interlocked, it may also indicate that the target system responded with a bus error for a monitor access.

Dealing with keep-alive circuitry by using the custom foreground monitor

As with the background monitor, you may have problems with keep-alive circuitry located in the target system. Because the foreground monitor cycles are shown to the target system, bus cycle activity monitors should not be a problem. Also, because interrupts can be serviced within a reasonable period of time, any keep-alive circuits that depend on interrupts should not be a problem.

Keep-alive circuits that require a certain address to be accessed probably will fail when you are using the foreground monitor. The keep-alive problem will most likely show up immediately when using the foreground monitor. If the monitor is interlocked, it will be affected immediately if a keep-alive circuit causes a bus error. If a keep-alive circuit generates an interrupt or a reset, it should also be immediately obvious. If reset is only temporarily asserted, it may not be so obvious because the emulator will return to the monitor when it is released.

If you suspect a problem with a keep-alive circuit, try using the custom foreground monitor. This monitor can be customized to take the required actions to satisfy a keep-alive circuit. See Chapter 8, "Configuring the Emulator", for information on using the custom foreground monitor. Retry your reset into the monitor with the customized foreground monitor.

If keep-alive circuits cannot be accommodated by using the available emulator features, you may need to disable them for emulation.

Testing memory access with the foreground monitor

Once the foreground monitor looks like it is running properly, you can use it to test accesses to different ranges of memory in your target system. This may be an easier way to diagnose problems than by running a program that accesses each memory range. It is also easy to check accesses of different sizes using the monitor.

mo -ax -dl
m 0badad=12345678

When accesses to your target memory are not performed exactly right, the monitor attempts to diagnose these problems and resolve them so the monitor program does not malfunction. However, the monitor does not read back write cycles to check the integrity of the data written. When testing memory accesses, check the data to make sure it is correct.

```
M>m 0badad
  0000badad  ffdf00ff
```

If your target memory does not respond to a bus cycle, the monitor will force termination of the cycle and report this error message.

!STATUS 170! Emulator terminated hung bus cycle: 0000badad@sd word read
!ERROR 700! Target memory access failed

Or, if the target system responds with a bus error for this memory access, the monitor will report that information.

!ERROR 170! Target bus error: 0000badad@sd
!ERROR 700! Target memory access failed

Running a program from the foreground monitor

Once you are satisfied that the monitor is working and that memory in your target system can be accessed correctly, you can use the monitor to run your target program. Use the following procedure:

1 Reset into the monitor.

2 Load a program, if necessary.

3 Initialize the initial stack pointer and initial program counter.

reg isp=<initial ISP>
reg pc=<starting address of target program>

If you do not know these values, you can find them by taking a trace of the program running from reset as done in the previous sections.

4 Take a trace of the program as it is running, using the following commands:

tg addr=<long aligned starting address of target program>
t

The trigger address must be long aligned because the MC68040 always fetches instructions as long words from long-word boundaries.

5 Run the program with the command:

r

6 Verify correct operation of the program.

Assuming that the program ran without the monitor, the stack is most likely the cause of any problems that you see. The monitor runs the program by creating a stack in memory at the location indicated by the initial stack pointer. The monitor then initiates an RTE, which starts the target program running. The following trace list shows an example of correct operation:

```
 Line   addr,H    68040 Mnemonic
-----   --------  ----------------------------------------
   -4   000010f0   $00------ log sdata byte read
   -3   00001e74   $4E714E71 log sprog long read
   -2   0000f0ec   $000a007C log sdata long read   <-unstack
   -1   0000f0e8   $27000000 log sdata long read   <-unstack
    0   00000008   $000060FE log sprog long read   <-target program
    1   0000000c   $000BADAD log sprog long read
```

If the monitor detects problems with the stack pointer (the stack pointer must be even), or if the monitor has a problem accessing the stack memory, an error message is issued. Additionally, the monitor checks to make sure that the stack has been written correctly before exiting. Problems are indicated by the following error messages:

!ERROR 151! Interrupt stack pointer is odd or uninitialized
!ERROR 610! Unable to run

This message indicates that the stack pointer is invalid. Only word aligned stack pointers are allowed with the emulator. The run is not attempted.

!ERROR 170! Target bus error: 00000f0e8@sd
!ERROR 610! Unable to run

This message indicates a bus error occurred during the stack write. This behavior can be caused if the stack is in a memory range that responds with bus error for all accesses or for write accesses. Or, this behavior can be caused by putting the stack where the target system fails to respond immediately; the bus error is the result of a timeout. Keep in mind that the stack grows down from the initial stack pointer.

!STATUS 170! Emulator terminated hung bus cycle: 00000f0e8@sd long write
!ERROR 610! Unable to run

This indicates that the stack is in an address range that did not respond with a memory strobe. Make sure that the stack is placed in valid memory.

!ERROR 151! Interrupt stack is not located in RAM: 00000f0e8@sd
!ERROR 610! Unable to run

This indicates that the stack memory was not writeable. Check to make sure that the stack is placed in RAM.

If the target program appears to start at the wrong address, or if there is some other problem, the stack can be decoded to see if the correct information is present. The stack above is interpreted as follows: The initial stack pointer is defined to point to

the next available stack location. Therefore, the exit stack starts four words below the initial stack pointer.

```
ISP-8 - Status register =     2700
ISP-6 - Program Counter = 0000000a
ISP-2 - Vector Offset   =     007C
```

The monitor is always exited using the FOUR WORD STACK frame, and the monitor always uses 07C as the vector offset. When running a program from the monitor after entering from reset, the powerup status word of 2700 is used. Therefore, the only difference you will see in this stack frame will be because of different initial program counters.

The procedure for setting the initial stack pointer and initial program counter can be automated by using the initial vectors configuration question to define these values.

cf rv=<initial ISP>,<target program starting address>

Once this configuration has been set up, the following reset sequence may be useful on systems that remap memory to provide reset vectors.

rst -m
r

Breaking into the foreground monitor

The next thing to try with the foreground monitor is to see if you can break into it from your target program. The emulator uses a nonmaskable interrupt (interrupt 7) to break into the monitor. The interrupt is generated in such a way as to not interfere with any interrupts pending in your target system. The resulting interrupt acknowledge cycle is not shown to the target system. The associated stacking is in foreground memory at the location determined by the interrupt stack pointer. If the target system program is running in Master mode, there will also be stacking on the master stack.

A vector fetch occurs sometime during or after stacking. The emulator provides the data for this vector fetch to correctly run the foreground monitor. While the emulator is transitioning into the foreground monitor, interrupts are temporarily blocked. Once in the monitor the interrupt mask level is lowered to the greater of the "monint" configuration setting or the target program mask level.

Entry into the foreground monitor can be traced by using the following trigger specification. The interrupt acknowledge signal is not shown to the target system and is also not shown to the analyzer unless background cycles are being traced.

tck -ub
tp c
tg stat=11xxxxxxxx1x111xy
t
b

```
 Line   addr,H    68040 Mnemonic
-----   --------  ----------------------------------------
   -2   00000008   $60FE0000 phy sprog long read
   -1   0000000c   $00000000 phy sprog long read
    0   ffffffff   $------FF mon int7 ack                <- acknowledge
    1   00000200   $0000040B mmu twalk data long read     <- twalk stack
    2   00000400   $0000060B mmu twalk data long read
    3   0000063c   $0000F01B mmu twalk data long read
    4   0000f0ee   $----007C phy sdata word write         <- stack format
    5   0000f0ea   $----0000 phy sdata word write         <- stack PC high
    6   0000f0ec   $0008---- phy sdata word write         <-stack PC low
    7   00000200   $0000040B mmu twalk data long read     <- twalk vector
    8   00000400   $0000060B mmu twalk data long read
    9   00000600   $0000009F mmu twalk data long read
   10   0000007c   $000016C2 phy sdata long read          <- vector fetch
   11   0000f0e8   $2700---- phy sdata word write         <- stack SR
   12   00000200   $0000040B     twalk prog long read     <- twalk monitor
   13   00000400   $0000060B     twalk prog long read
   14   00000600   $0000101b     twalk prog long read
   15   000016c0   $4E732F0D phy sprog long read          <- monitor
   16   000016c4   $4BFAFB10 phy sprog long read
```

If you have problems trying to break into the monitor, the most likely causes are that the stack pointers or vector base register do not point to valid memory. Any exceptions during monitor entry will cause the break to fail. Access errors during stacking or vector fetches are the most common causes of failures. The target system can respond with a bus error, or if the MMU is running, the MMU can signal an access error. The MMU will signal an error if a translation is not available, if a bus error occurs during translation lookup, or if a write-protection error occurs.

The break will also fail if accesses to the monitor cause an exception. This includes bus errors and access errors signaled by the MMU. It is possible for the monitor to execute correctly until the MMU is enabled, and then have problems. Keep in mind that the monitor must be translated logical=physical and located in address space that is not write-protected.

If any stacking or vector-fetch cycles are not terminated, the monitor will terminate them by force. If this happens, the PC and SR may be displayed incorrectly by the monitor. The same problem can result from stack memory that is not writeable. Neither condition will prevent entry into the monitor, but you will not be able to resume execution in the target program.

Exiting the foreground monitor

If the tests of the preceding paragraphs operate correctly, you should be able to resume execution of the target program. You may want to take a trace of the monitor exit to verify that everything is working correctly. Use the run command:

r

Software breakpoint entry into the foreground monitor

The foreground monitor can also be entered via a software breakpoint. The emulator will respond to any software breakpoint instruction in the code if breakpoints are enabled, regardless of whether the breakpoint was inserted by the emulator or not. Breakpoints are enabled by the following command:

bc -e bp

Only set breakpoints on the initial word of an instruction; otherwise, they will not be executed, and they may alter an instruction, unintentionally. The emulator can place a breakpoint using two methods. By default, the emulator will attempt to modify memory to insert a breakpoint instruction at the address specified. If the memory at the address specified is ROM or cannot be modified for some other reason, special hardware resources on the emulator will interject a breakpoint instruction when the associated address is fetched. You can tell if a hardware resource was required to support a breakpoint by viewing memory at the breakpoint address. If the BKPT instruction has replaced the normal instruction at that address, a software breakpoint was used. If the normal instruction is still in the

breakpoint address, the emulator is using one of its eight hardware resources to implement the breakpoint.

b
bp <program instruction>

If you suspect some kind of problem with the setting of the breakpoint, use the analyzer to watch the setting of the breakpoint. The easiest way to do this is to store-qualify the trace on the address where you are setting the breakpoint. The trace list will only contain a cycle or two, but you can see what happened when the emulator accessed the breakpoint address.

If the MMU is running, you will need to store-qualify the actual physical address being accessed. The address given in the "bp" command must always be treated as a logical address. To find the corresponding physical address, use the MMU translation command. Also, keep in mind that the MMU may cause problems when setting the breakpoint.

mmu -t <logical breakpoint address>

tg any
tsto addr=<physical breakpoint address>
b
bp <logical breakpoint address>

```
 Line   addr,H    68040 Mnemonic
-----   --------  ----------------------------------------
    0   00000008   $FFFF---- phy sdata word read
    1   00000008   $FFFF---- phy sdata word read
    2   00000008   $FFFF---- phy sdata word read
    3   00000008   $484F---- phy sdata word write  <- breakpoint write
    4   00000008   $FFFF---- phy sdata word read  <- verify
    5   00000008   $FFFF---- phy sdata word read
    6
```

When a software breakpoint instruction is executed, the processor initiates a breakpoint-acknowledge cycle. This cycle signals the start of an entry into the monitor. From this point on, stacking and the vector fetch proceed the same as for a break entry. Unlike the interrupt-acknowledge cycle, the breakpoint-acknowledge cycle is shown to the target system.

tck -u
tsto any
tg stat=11xxxxxxxx1x000xy
t
r 8

```
 Line   addr,H    68040 Mnemonic
-----   --------  ----------------------------------------
   -4   00000008   $484F0000 log sprog long read          <- bkpt fetch
   -3   0000000c   $00000000 log sprog long read
   -2   00000010   $01000000 log sprog long read
   -1   00000014   $00000000 log sprog long read
    0   00000000   $41------     bkpt ack (buserror)      <- acknowledge
    1   00000200   $0000040B mmu twalk data long read     <- twalk stack
    2   00000400   $0000060B mmu twalk data long read
    3   0000063c   $0000F01B mmu twalk data long read
    4   0000f0ee   $----0010 phy sdata word write         <- stack format
    5   0000f0ea   $----0000 phy sdata word write         <- stack PC high
    6   0000f0ec   $0008---- phy sdata word write         <- stack PC low
    7   00000200   $0000040B mmu twalk data long read     <- twalk vector
    8   00000400   $0000060B mmu twalk data long read
    9   00000600   $0000009F mmu twalk data long read
   10   00000010   $000016A2 phy sdata long read          <- vector fetch
   11   0000f0e8   $2700---- phy sdata word write         <- stack SR
   12   00000200   $0000040B     twalk prog long read     <- twalk monitor
   13   00000400   $0000060B     twalk prog long read
   14   00000600   $0000101b     twalk prog long read
   15   000016a0   $007E2F0D phy sprog long read          <- monitor
   16   000016a4   $4BFAFA73 phy sprog long read
```

The only unique part of a breakpoint entry is the breakpoint-acknowledge cycle so any problems will probably be related to this cycle. Because the emulator internally responds to this cycle, it is not necessary for the target system to respond to it. If the target system responds to this cycle with any wait states, the emulator may become out of sync with the target system because the emulator terminates this cycle immediately. If this causes a problem, it will show up on the cycle immediately following the breakpoint-acknowledge cycle.

Stepping with the foreground monitor

The last feature of the foreground monitor that needs to be evaluated is the single-stepping facility. The emulator uses the processor trace facility to reenter the monitor after executing exactly one instruction, unless an exception occurs.

b
tsto any
tg stat=11xxxxxxxx1x000xy
t
s

```
000000008@s  -                BRA.B     $00000008
PC = 000000008@s
```

When a step command is issued, the emulator sets the trace bits in the SR, and then performs a normal monitor exit. The emulator modifies the trace vector to transfer control to the monitor. A typical trace of a single step is shown below:

```
 Line   addr,H    68040 Mnemonic
-----   --------  ----------------------------------------
  -42   000010f0   $00------ log sdata byte read
  -41   00001e74   $4E714E71 log sprog long read
  -40   00000200   $0000040B mmu twalk data long read     <- twalk stack
  -39   00000400   $0000060B mmu twalk data long read
  -38   0000063c   $0000F01B mmu twalk data long read
  -37   0000f0ec   $0008007C log sdata long read          <- unstack
  -36   0000f0e8   $A7000000 log sdata long read          <- unstack
  -35   00000200   $0000040B     twalk prog long read     <- twalk monitor
  -34   00000400   $0000060B     twalk prog long read
  -33   00000600   $0000009f     twalk prog long read
  -32   00000008   $60FE0000 log sprog long read          <- stepped inst
  -31   0000000c   $00000000 log sprog long read
  -30   00000008   $60FE0000 log sprog long read
  -29   0000000c   $00000000 log sprog long read
  -28   0000f0ec   $00000008 log sdata long write         <- stack address
  -27   0000f0ea   $----2024 log sdata word write         <- stack format
  -26   0000f0e6   $----0000 log sdata word write         <- stack PC high
  -25   0000f0e8   $0008---- log sdata word write         <- stack PC low
  -24   00000200   $0000040B mmu twalk data long read     <- twalk vector
  -23   00000400   $0000060B mmu twalk data long read
  -22   00000600   $0000009F mmu twalk data long read
  -21   00000024   $00001680 log sdata long read          <- vector fetch
  -20   0000f0e4   $A700---- log sdata word write         <- stack SR
  -19   00001680   $2F0D4BFA log sprog long read          <- monitor
  -18   00001684   $FB523ABC log sprog long read
  -17   00001688   $20246000 log sprog long read
  -16   0000168c   $00924BFA log sprog long read
  -15   0000f0e0   $00000000 log sdata long write
```

```
-14   00001718   $2F256000 log sprog long read
-13   000011d6   $----2024 log sdata word write
-12   0000171c   $0022083A log sprog long read
-11   00001720   $0002F9D6 log sprog long read
-10   00001724   $67184BFA log sprog long read
 -9   00001728   $F9F108D5 log sprog long read
 -8   000010f8   $F5------ log sdata byte read
 -7   0000172c   $0003484F log sprog long read          <- monitor bkpt
 -6   00001730   $4E7AD002 log sprog long read
 -5   00001734   $4A8D6A06 log sprog long read
 -4   00001738   $F4784BFA log sprog long read
 -3   00001119   $--03---- log sdata byte read
 -2   0000173c   $F8C40C3A log sprog long read
 -1   00001119   $--0B---- log sdata byte write
  0   00000000   $41------     bkpt ack (buserror)      <- acknowledge
  1   0000f0de   $----0010 log sdata word write
  2   0000f0da   $----0000 log sdata word write
  3   0000f0dc   $172E---- log sdata word write
  4   00000010   $000016A2 log sdata long read
  5   0000f0d8   $2704---- log sdata word write
  6   000016a0   $007E2F0D log sprog long read
  7   000016a4   $4BFAFA73 log sprog long read
```

At the end of the execution of the first target program instruction, the processor takes a trace exception. Stacking for this trace exception commences and at some point, the modified trace vector is fetched. The monitor internally uses a breakpoint instruction, but it is not part of the entry sequence.

If an error occurs during modification of the trace vector, an error message similar to the following is displayed.

!ERROR 170! Target bus error: 0ff800024@sd
!ERROR 156! Unable to modify trace vector to 000001680 for single stepping
!ERROR 680! Stepping failed

If the emulator does not reenter the monitor after stepping, as indicated by the following error message, there can be a number of explanations. If the emulator steps an instuction that modifies the VBR, the step will fail because the modified trace vector will not be used to reenter the monitor. To get around this problem, the trace vector in the target program can be modified to point to the monitor entry point <monaddress + 0680>.

!ERROR 680! Stepping failed

Stepping will behave differently when executing instructions that cause the processor to take exceptions. Most exceptions preempt the trace exception until after their exception handler runs. Other exceptions (like TRAP, CHK, and CHK2) create their stack frame and then take the trace exception.

For all exceptions except TRAP, CHK, and CHK2, the exception handler will execute before the trace exception is taken to return to the monitor. Exception handlers that are instruction emulators are responsible for emulating the trace behavior as well. If they do not emulate this behavior, stepping may fail because the trace exception will never happen.

The TRAP, CHK, and CHK2 exception handlers do not run before the trace exception is taken. They will have an additional stack frame when the monitor is entered. The exception stack frame will precede the normal trace stack frame.

Installing emulation memory

The last feature of the emulator that you need to integrate is the emulation memory. Emulation memory is intended to overlay ROM in the target system. This allows changes to target programs to be quickly loaded into a system. Emulation memory is not dual ported as is the case with the monitor memories. To display and modify emulation memory, you must use the monitor.

If emulation memory is placed over existing target memory, interlock it to the target memory strobes. This ensures that the target memory control circuits remain in sync with the emulator. If there are no strobes that respond in the address range where emulation memory is placed, then do not interlock. When interlocked, both the TA and TEA signals are sampled.

```
!ERROR  170! Target failed to terminate bus cycle: 000000000@sd word read
!STATUS  170! Emulator terminated hung bus cycle: 000000000@sd word read
!ERROR  702! Emulation memory access failed
```

To effectively use emulation memory, the monitor must be able to read and write to it. Read and write accesses to emulation memory are seen by the target system. Emulation memory will not be able to be loaded if it is interlocked and the target system asserts bus error on write cycles or does not terminate the cycle.

```
!ERROR  170! Target bus error: 0000badad@sd
!ERROR  702! Emulation memory access failed
```

If the memory is write-protected by the MMU, the monitor will temporarily disable this protection to complete the write. This applies to both emulation memory and target memory. Once emulation memory is mapped, it can be tested by performing accesses from the monitor.

If the MMU is turned on and there are no address translations for the requested emulation memory access, you will see the following error message:

```
!ERROR  170! Address translation error; non-resident page: 000f84000@sd
!ERROR  702! Emulation memory access failed
```

19

Installation and Service

Installation

This chapter shows you how to install emulation and analysis hardware and interface software. It also shows you how to verify installation by starting the emulator/analyzer interface for the first time. These installation tasks are described in the following sections:

- Installing hardware.
- Connecting the HP 64700 to a computer or LAN.
- 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.

64700 Operating Environment The Graphical User Interface requires version A.05.00, or later, of the 64700 Operating Environment. (The Graphical User Interface version is A.05.10.)

Motif/OSF For Series 9000/700 workstations, you must also have the Motif 1.1 dynamic link libraries installed. They are installed by default.

Hardware and Memory Any workstation used with the Graphical User Interface should have a minimum of 16 megabytes of memory (32 megabytes or more is recommended). 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 4.1.1 or greater (Solaris 1.X). At the time this manual was printed, the Graphical User Interface software would not run on Solaris 2.X; ask your Hewlett-Packard Sales Office if the present version of Graphical User Interface software will run on Solaris 2.X. The tape uses the QIC-24 data format.

64700 Operating Environment The Graphical User Interface requires version A.05.00 or greater of the 64700 Operating Environment. (The minimum Graphical User Interface version required is A.05.10.)

Hardware and Memory Any workstation used with the Graphical User Interface should have a minimum of 16 megabytes of memory (32 megabytes or more is recommended). 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.

Installing Hardware

This section describes how to install emulation and analysis hardware and how to connect the emulator probe to the demo target system.

Equipment supplied

The minimum system contains:

- HP 64783A/B 68040/68EC040/68LC040 PGA Emulator Probe (which includes the demo target system).
- HP 64748C Emulation Control card.
- HP 64794 Emulation-Bus Analyzer (deep analyzer) card, or HP 64704A Emulation-Bus Analyzer (1K analyzer) card.
- Ribbon cable.
- HP 64700 Card Cage.

Optional parts are:

- HP 64172A 256-Kbyte Memory Modules for additional memory depth.
- HP 64172B 1-Mbyte Memory Modules for additional memory depth.
- HP 64173A 4-Mbyte Memory Modules for additional memory depth.
- HP 64708A Software Performance Analyzer.

Equipment and tools needed

In order to install and use the MC68040 emulation system, you need:

- Flat-blade screwdriver with shaft at least 5 inches long (13 mm approx).

Installation overview

The steps in the installation process are:

1 Install optional memory modules on the deep analyzer card, if desired.
2 Connect the HP 64783A/B emulator probe to the HP 64748C emulator control card.
3 Install cards into the HP 64700 card cage.
4 Install emulation memory modules on the emulator probe.
5 Connect the emulator probe to the demo target system.
6 Apply power to the HP 64700 Card Cage.

Antistatic precautions

Printed-circuit boards contain electrical components that are easily damaged by small amounts of static electricity. To avoid damage to the emulator boards, follow these guidelines:

- If possible, work at a static-free workstation.
- Handle the boards only by the edges; do not touch components or traces.
- Use a grounding wrist strap that is connected to the HP 64700's chassis.

Caution

If you already have a modular HP 64700 Series Card Cage and want to remove the existing emulator and insert an HP 64783A/B emulator in its place, the HP 64700 Series generic firmware and analyzer firmware may NOT be compatible, and the software will indicate incompatibility. In this event, you must purchase a Flash EPROM board to update the firmware. Instructions for installing this board and programming it from a PC or HP 9000 are provided in the HP 64700 Card Cage Installation/Service manual. Instructions for installing and updating emulator firmware are covered in Chapter 20, "Installing/Updating Emulator Firmware".

Note

If you already have a modular HP 64700 Series Card Cage and want to remove the 1K analyzer and install the deep analyzer in its place, the analyzer firmware will be updated by your installation because the analyzer firmware is contained on the analyzer card.

Checking Hardware Installation

After hardware installation, run a performance test to verify that the emulator is working properly. The performance verification procedure is described under "Verifying the Installation" later in this chapter.

Service Information

Use this chapter when removing and installing hardware, running performance verification, and ordering parts. See the HP 64700 Series Installation/Service Guide for information on system configurations, installing product software, software updates, and ordering parts for the card cage. Turn off power to the card cage before removing or installing hardware.

Step 1. Install optional memory modules on Deep Analyzer card, if desired

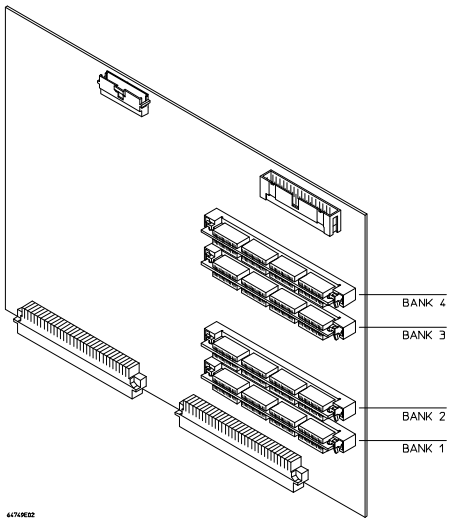
Observe antistatic precautions

With no optional memory modules installed on the deep analyzer card, the trace memory depth is 8K. If you are going to use the deep analyzer with this default trace memory depth, skip this step and proceed to Step 2 of this installation procedure.

1 Determine placement of the optional memory modules. Two types of modules may be installed: 256-Kbyte (HP 64172A), and 1-Mbyte (HP 64172B). Either module type may be installed in the banks on the analyzer card. Do not use HP 64171A/B or HP 64173A memory modules; they are too slow.

If you install no memory modules, the deep analyzer will have 8K maximum memory depth.
If you install four 256-Kbyte memory modules, the analyzer will have 64K maximum memory depth.
If you install four 1-Mbyte memory modules, the analyzer will have 256K maximum memory depth.

If you install a combination of 256-Kbyte memory modules and 1-Mbyte memory modules, the analyzer will have 64K maximum memory depth. All four connectors must have memory modules installed before the analyzer depth will be increased.



2 To ensure correct installation of optional memory modules on the deep analyzer card, there is a cutout at one end of the memory modules so they can only be installed the correct way.

To install a memory module:

Align the groove in the memory module with the alignment rib in the connector.

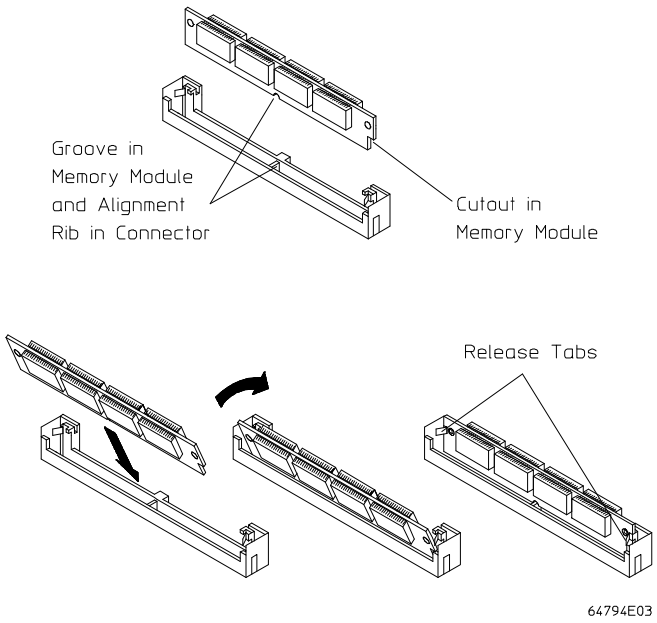
Align the cutout in the memory module with the projection in the connector.

Place the memory module into the connector groove at an angle.

Firmly press the memory module into the connector and make sure it is completely seated.

Rotate the memory module forward so that the pegs on the connector fit into the holes on the memory module.

Make sure the release tabs at each end of the connector snap around the memory module to hold it in place.



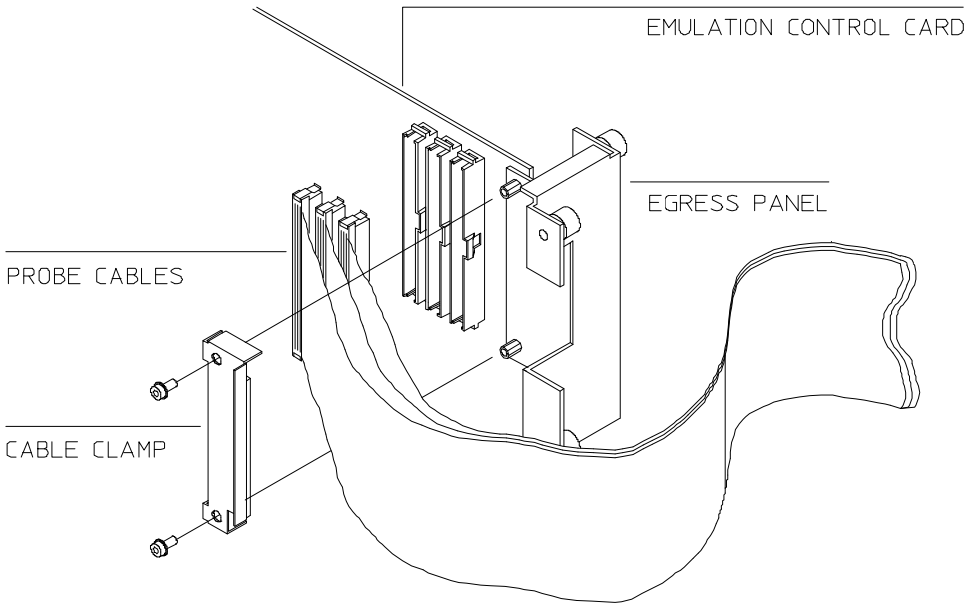
Step 2. Connect the Emulator Probe Cables

Three ribbon cables connect the HP 64748C emulation control card to the HP 64783A/B emulator probe.

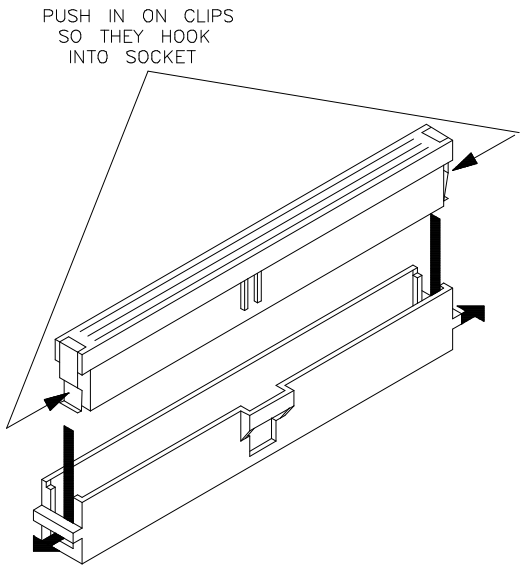
The shortest cable connects from J1 of the emulation control card to J3 of the emulator probe. The medium length cable connects from J2 of the emulation control card to J2 of the emulator probe. The longest cable connects from J3 of the emulation control card to J1 of the emulator probe.

Make sure the cable connectors are seated. There are stainless steel clips on the cable connectors; these must be properly latched inside the sockets. Otherwise, the cables will work loose and you will see erratic operation. See illustration next page (step 2).

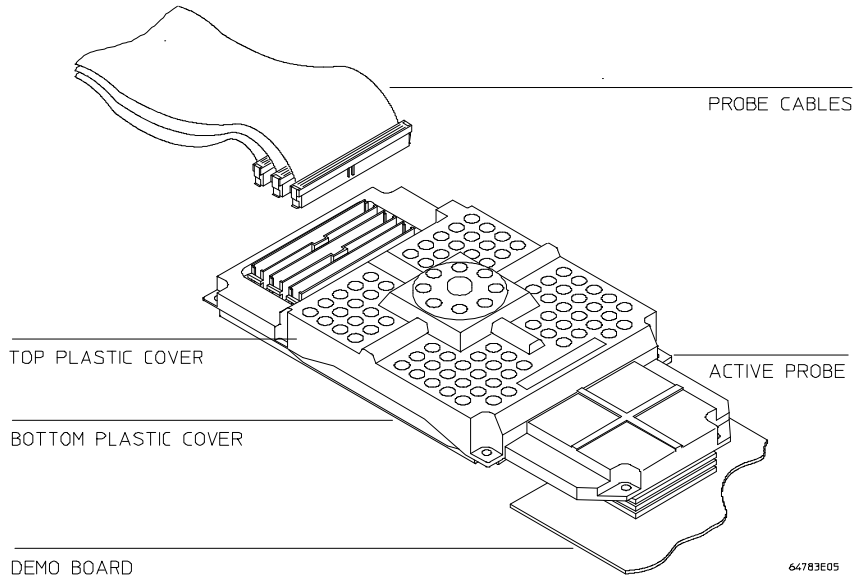
1 Connect the emulator probe cables to the emulation control card.



2 When inserting cable connectors into the sockets, press inward on the connector clips so that they hook into the sockets as shown. The order of connecting cables was given in step 1.



3 Connect the other ends of the cables to the emulator probe. Again, make sure the stainless steel clips on the cable connectors are properly latched within the sockets, as shown in step 2.

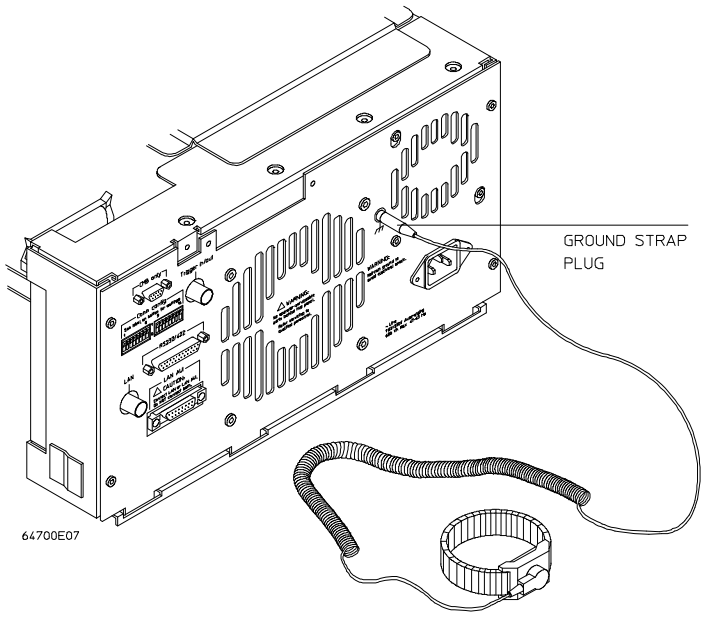


Step 3. Install Boards into the HP 64700 Card Cage

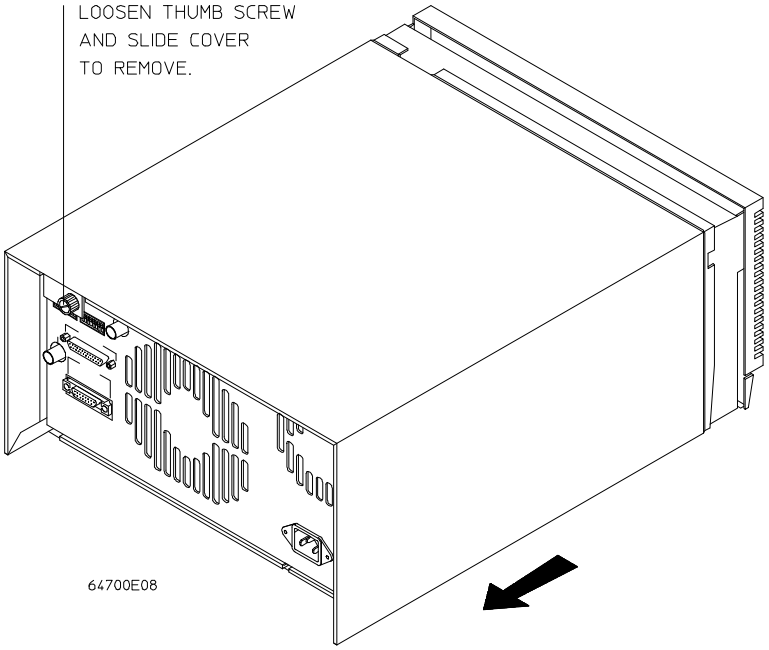
WARNING **Before removing or installing parts in the HP 64700 Card Cage, make sure that the card cage power is off and that the power cord is disconnected.**

CAUTION Do NOT stand the HP 64700 Card Cage on the rear panel. You could damage the rear panel ports and connectors.

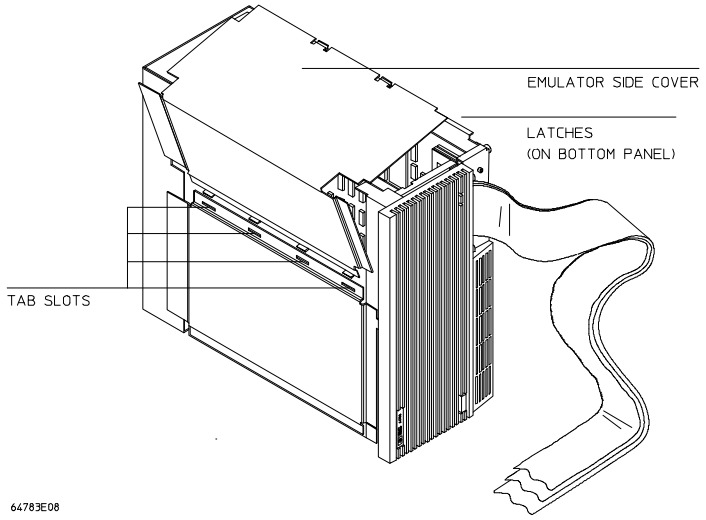
1 Use a ground strap when removing or installing boards into the HP 64700 Card Cage to reduce the risk of damage to the circuit cards from static discharge. A jack on the rear panel of the HP 64700 Card Cage is provided for this purpose.



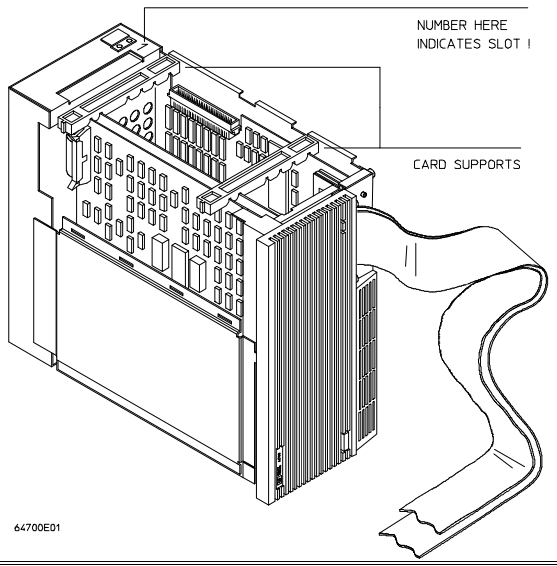
2 Turn the thumb screw and remove the top cover by sliding the cover toward the rear and up.



3 Remove the side cover by unsnapping the two latches and lifting off.

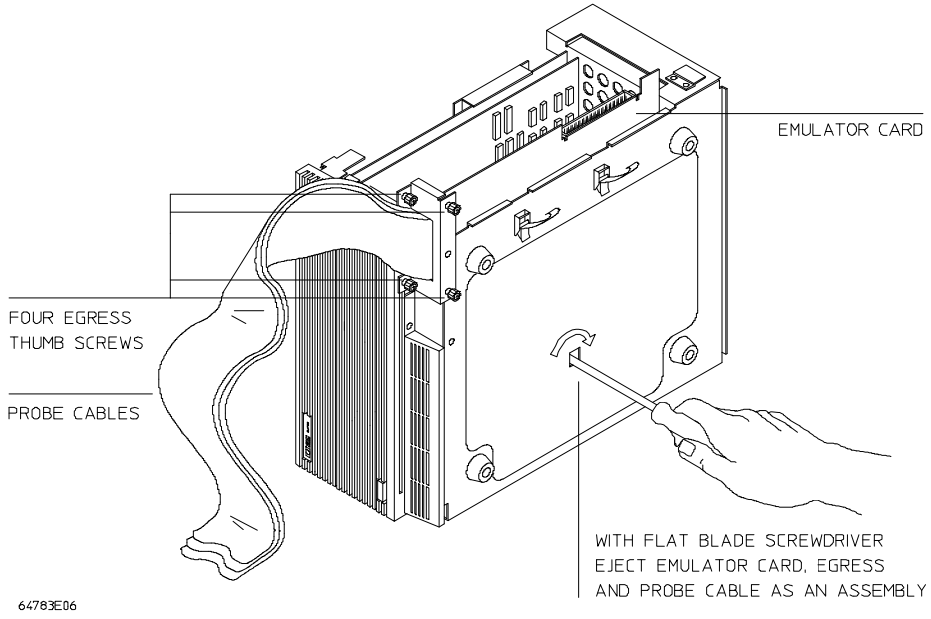


4 Remove the card supports.



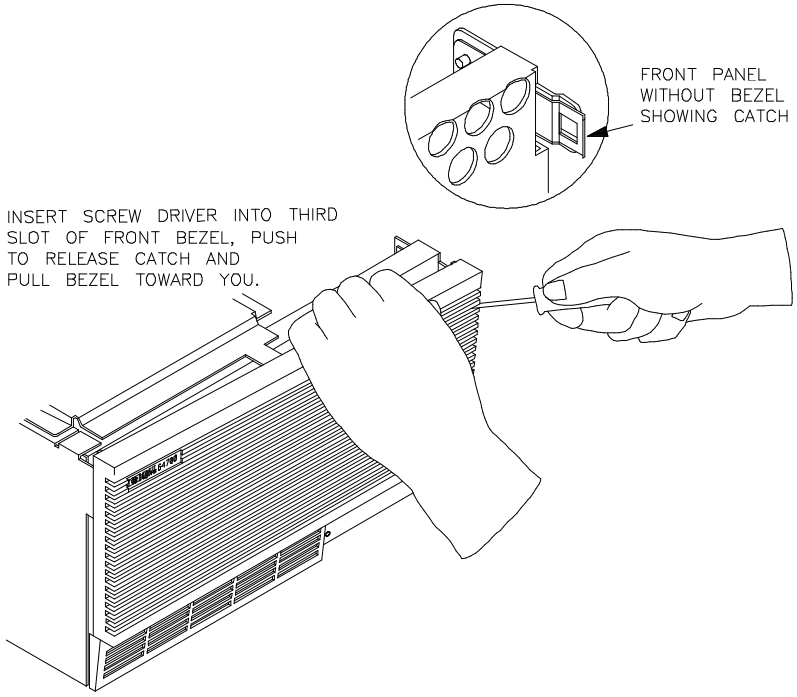
5 First, completely loosen the four egress thumb screws.

To remove emulator cards, insert a flat blade screwdriver in the access hole and eject the emulator cards by rotating the screwdriver.

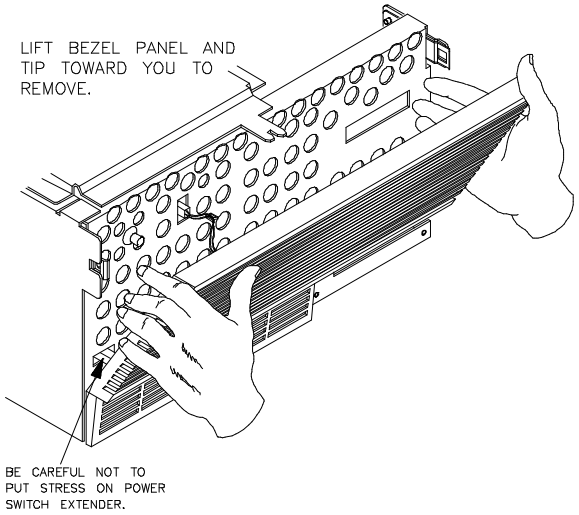


6 Insert a screw driver into the third slot of the right side of the front bezel, push to release catch, and pull the right side of the bezel about one-half inch away from the front of the HP 64700. Then, do the same thing on the left side of the bezel. When both sides are released, pull the bezel toward you approximately 2 inches.

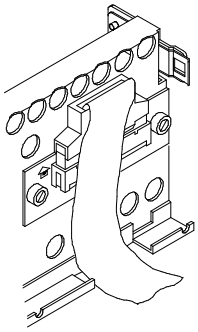
Be careful because the plastic ears are easily broken on the front bezel.



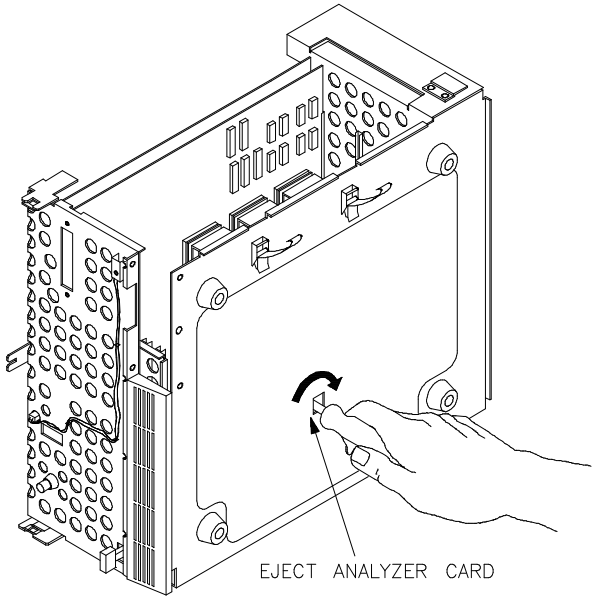
7 Lift the bezel panel to remove. Be careful not to put stress on the power switch extender.



8 If you're removing an existing analyzer card that provides external analysis, remove the right-angle adapter board by turning the thumb screws counterclockwise.



9 To remove the analyzer card, insert a flat blade screwdriver in the access hole and eject the analyzer card by rotating the screwdriver.

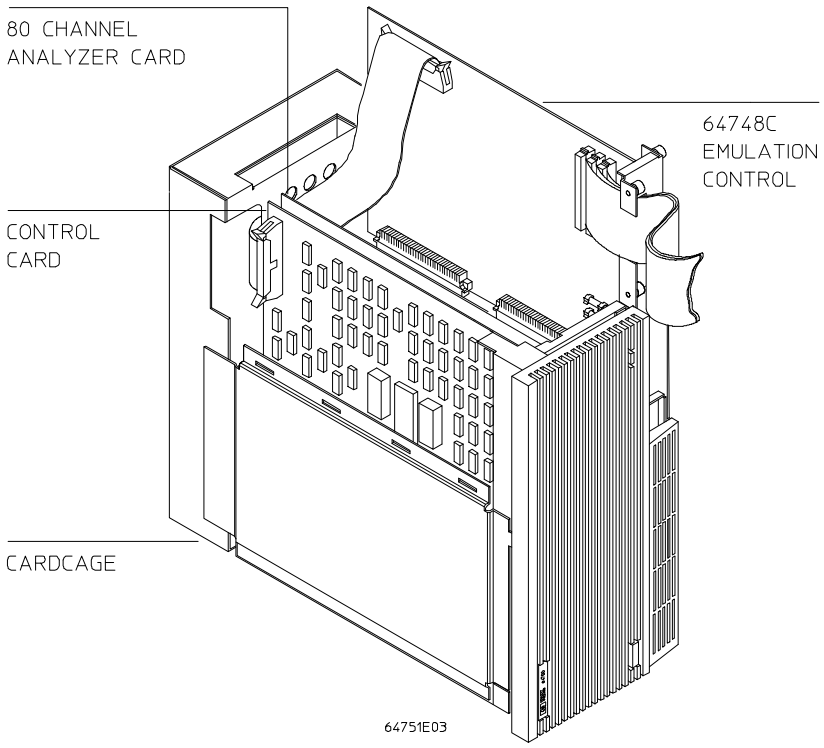


Do not remove the system control board. This board is used in all HP 64700 emulation and analysis systems.

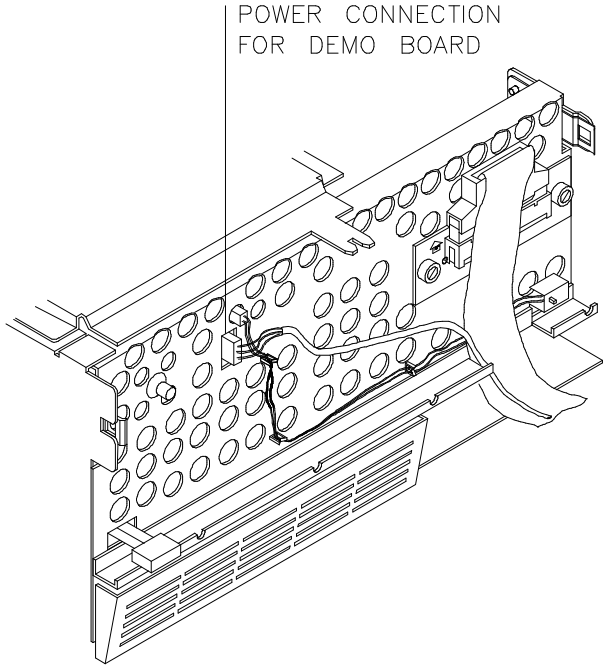
10 Install the analyzer and emulation control cards. The analyzer is installed in the slot next to the system control card. The emulation control card is installed in the second slot from the bottom of the card cage. The software performance analyzer card may occupy any slot between the emulation-bus analyzer and the emulation control card. These cards are identified with labels that show their model numbers and serial numbers. Note that components on the analyzer card face the opposite direction to the other cards.

To install a card, insert it into the plastic guides. Make sure the connectors are properly aligned; then, press the card into the mother board socket. Ensure that each card is seated all the way into its socket. If the cards can be removed with your fingers, the cards are NOT seated all the way into the mother board sockets.

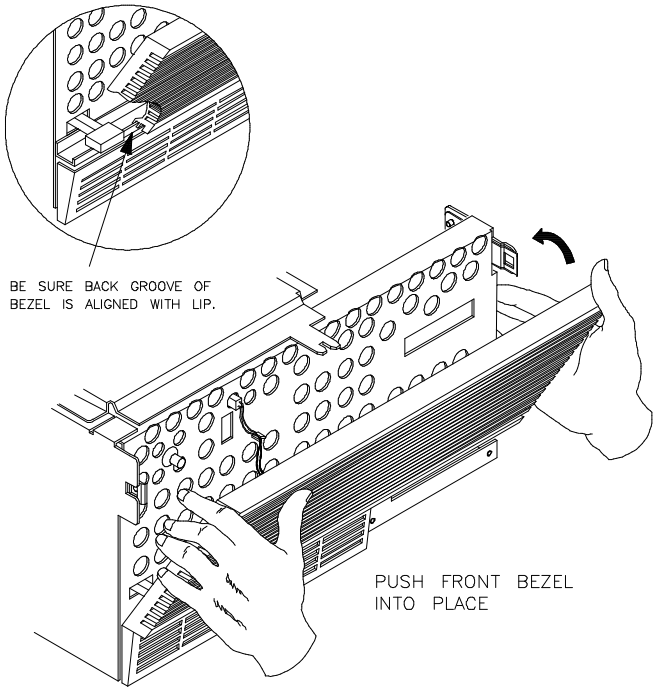
Attach the ribbon cable from the emulation control card to the analyzer card, and to the software performance analyzer, if installed. Tighten the thumbscrews that hold the emulation control card to the cardcage frame.



11 Connect the +5 V power cable to the connector in the HP 64700 front panel.

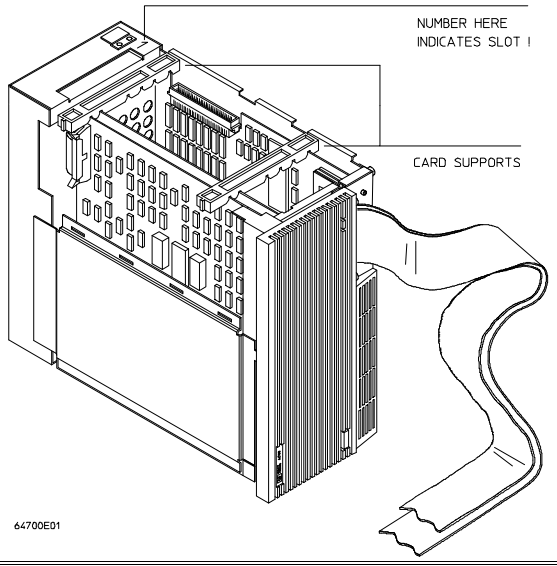


12 To reinstall the front bezel, be sure that the bottom rear groove of the front bezel is aligned with the lip as shown below.

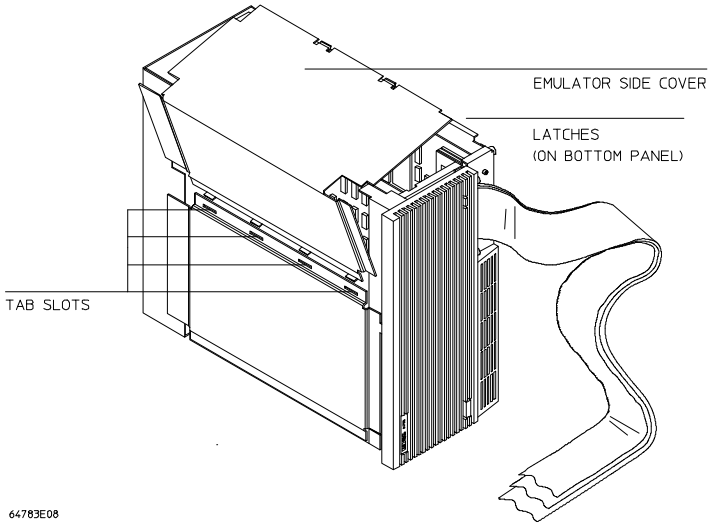


13 If you wish to install the Flash card (used for updating firmware, see Chapter 20

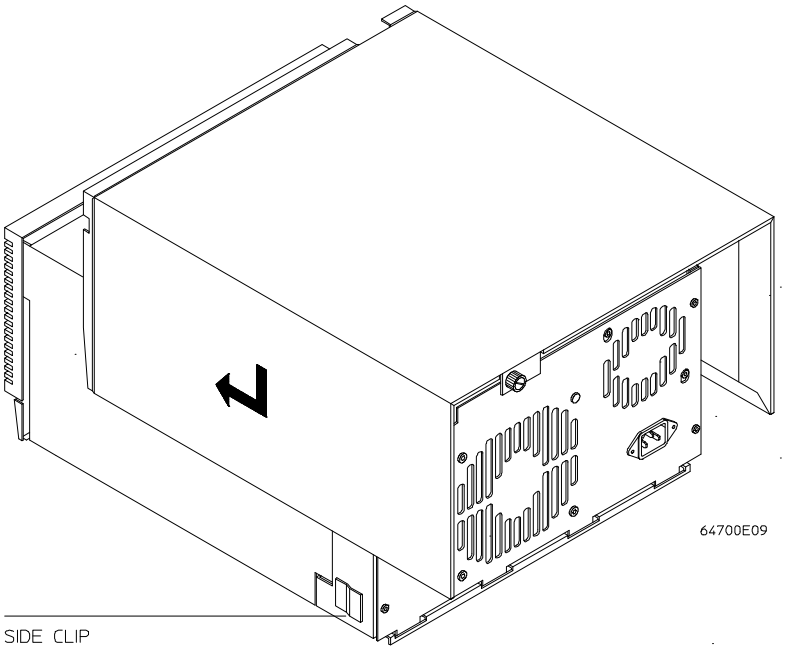
14 Install the card supports.



15 To install the side cover, insert the side cover into the tab slots and fasten the two latches.



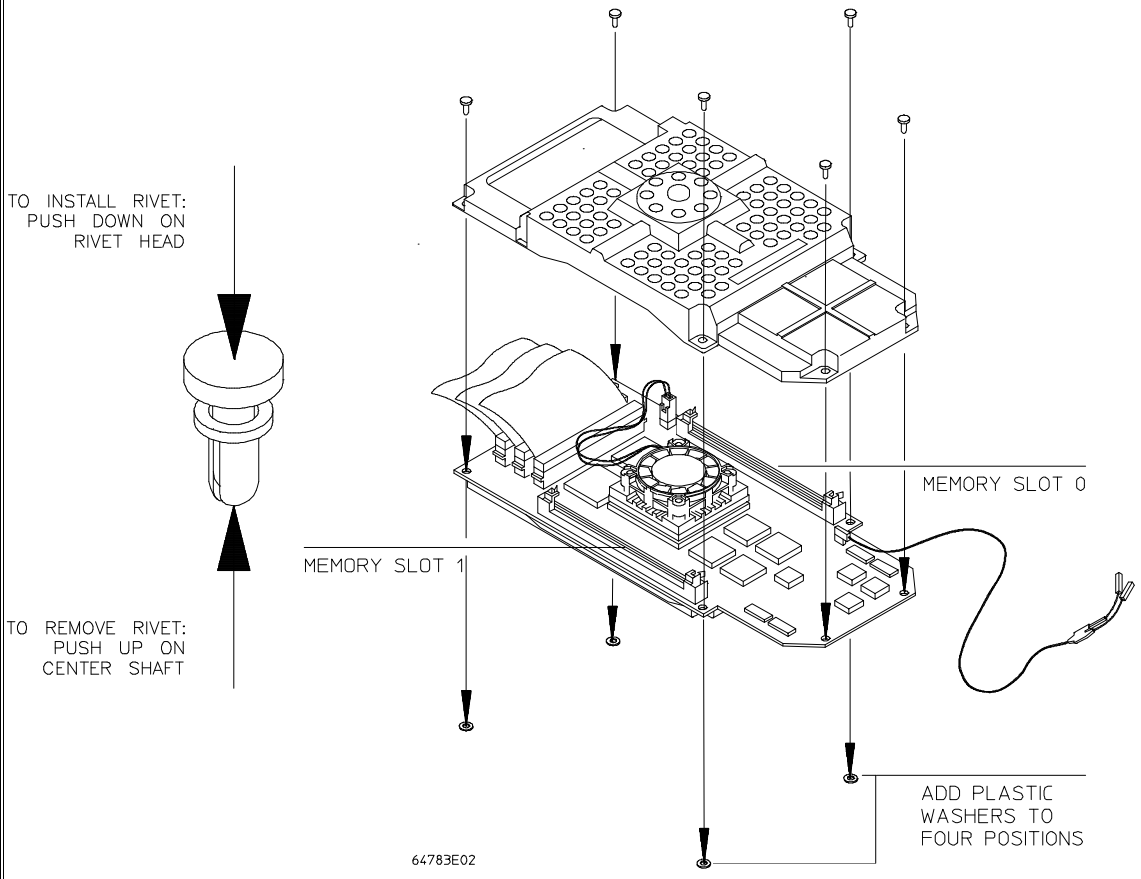
16 Install the top cover in reverse order of its removal, but make sure that the side panels of the top cover are attached to the side clips on the frame.



Step 4. Install emulation memory modules on emulator probe

(Observe antistatic precautions)

1 Remove plastic rivets that secure the plastic cover on the top of the emulator probe, and remove the cover. The bottom cover is only removed when you need to replace a defective active probe on the exchange program.



2 Determine the placement of the emulation memory modules. Three types of modules may be installed: 256 Kbyte (HP 64172A), 1 Mbyte (HP 64172B), and 4 Mbyte (HP 64173A). Any of the emulation memory modules can be installed in either memory slot on the probe. Do not use HP 64171A/B modules; they are too slow.

Memory in memory slot 0 is divided into four equal blocks that can be allocated by the memory mapper. Memory in memory slot 1 is divided into two equal blocks.

If you have only one emulation memory module, place it in memory slot 0.

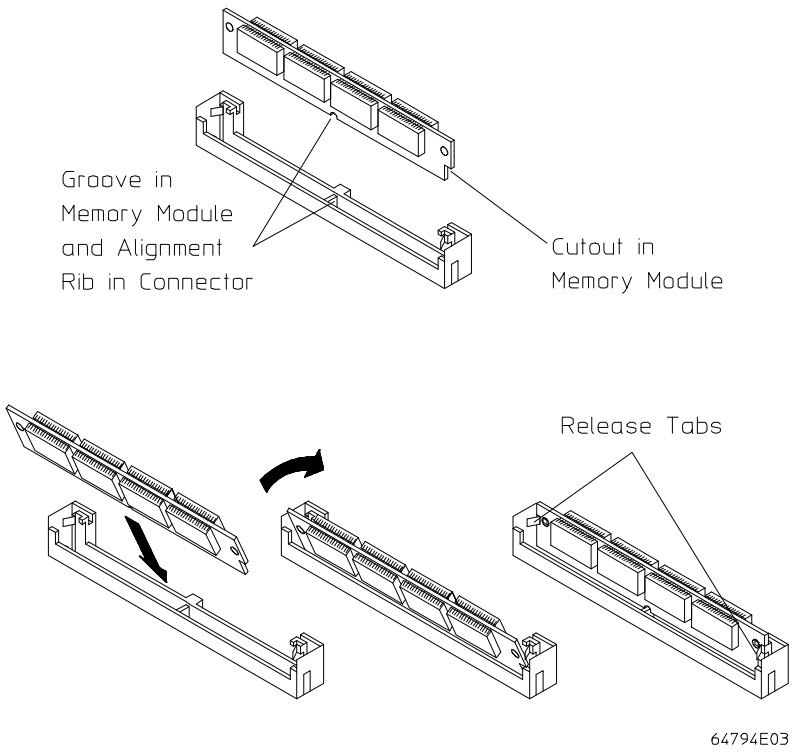
If you have two memory modules of different sizes, place the memory module with the greatest capacity in memory slot 0 to take advantage of the way memory slot 0 and memory slot 1 are divided by the emulator. For example, if you install a 1-Mbyte module in memory slot 0 and a 256-Kbyte module in memory slot 1, then the emulator will provide four 256-Kbyte blocks of memory in memory slot 0 and two 128-Kbyte blocks of memory in memory slot 1.

If you install any 4-Mbyte memory modules (HP 64173A) on the emulation probe, the emulator will detect their presence and add one wait state to accesses it makes to emulation memory. The 4-Mbyte memory modules are not as fast as the 256-Kbyte and 1-Mbyte memory modules.

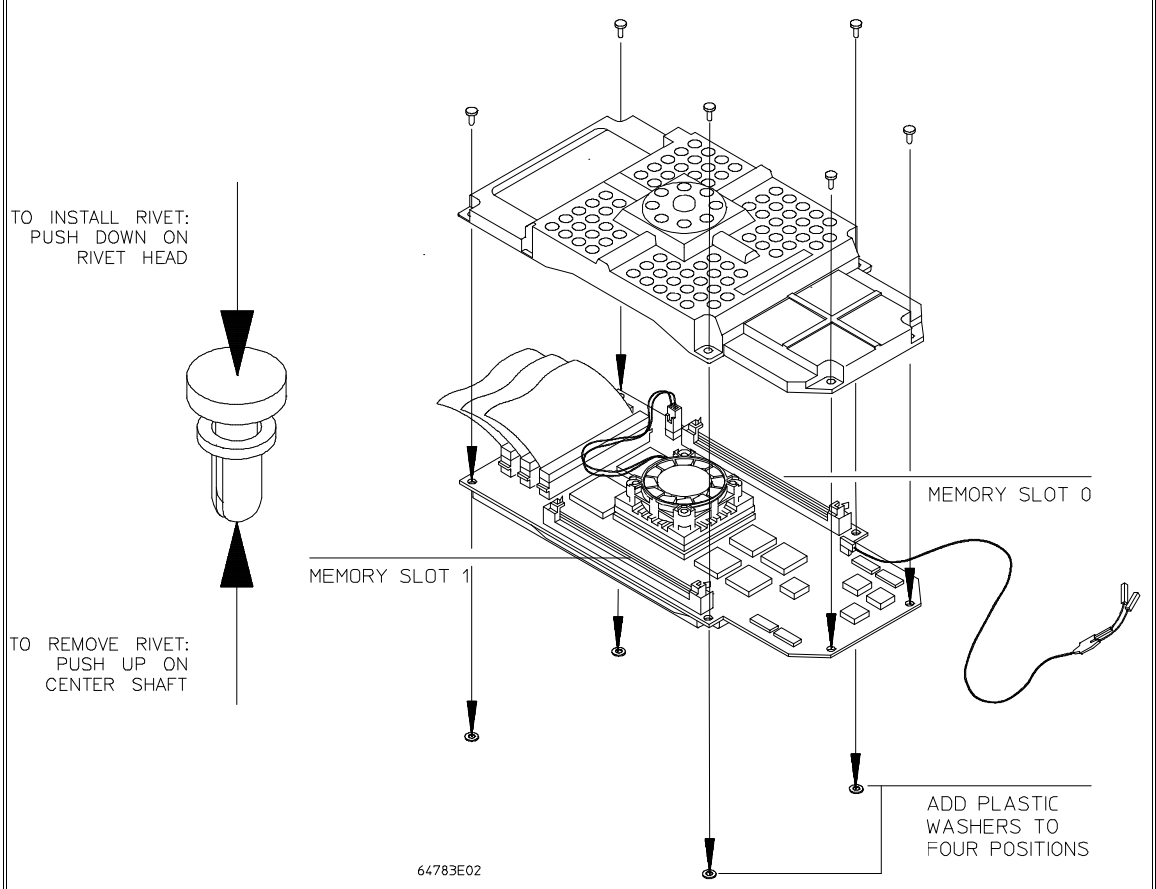
3 Install emulation memory modules on the emulator probe. There is a cutout at one end of the memory modules so they can only be installed the correct way.

To install a memory module:

1 Align the groove in the memory module with the alignment rib in the connector.
2 Align the cutout in the memory module with the projection in the connector.
3 Place the memory module into the connector groove at an angle.
4 Firmly press the memory module into the connector and make sure it is completely seated.
5 Rotate the memory module forward so that the pegs on the connector fit into the holes on the memory module.
6 Make sure the release tabs at each end of the connector snap around the memory module to hold it in place.

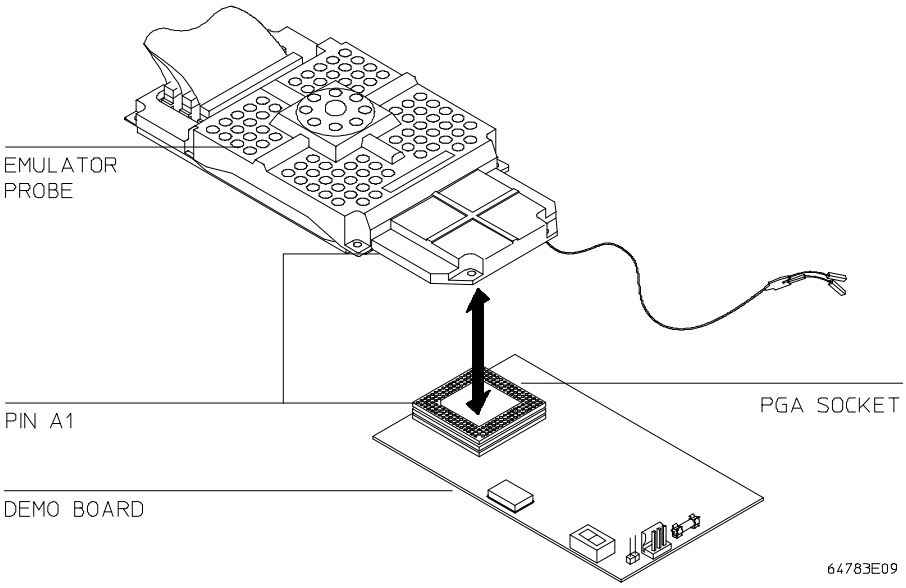


4 Replace the plastic cover, and insert new plastic rivets (supplied with the emulator) to secure the cover.

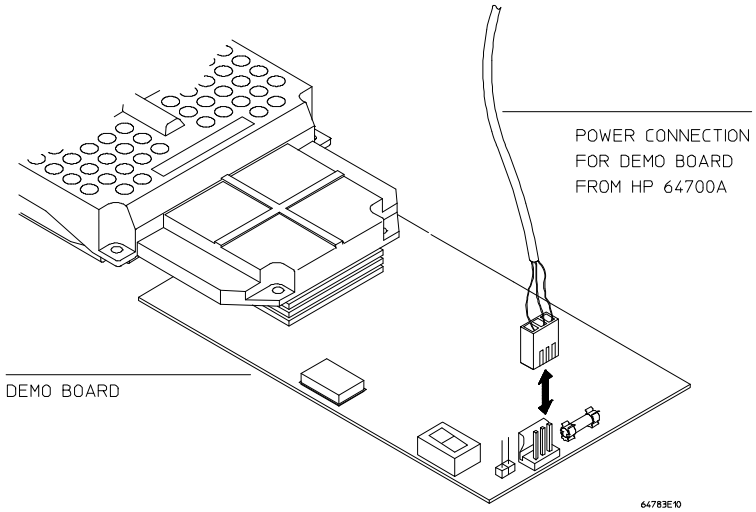


Step 5. Connect the emulator probe to the demo target system

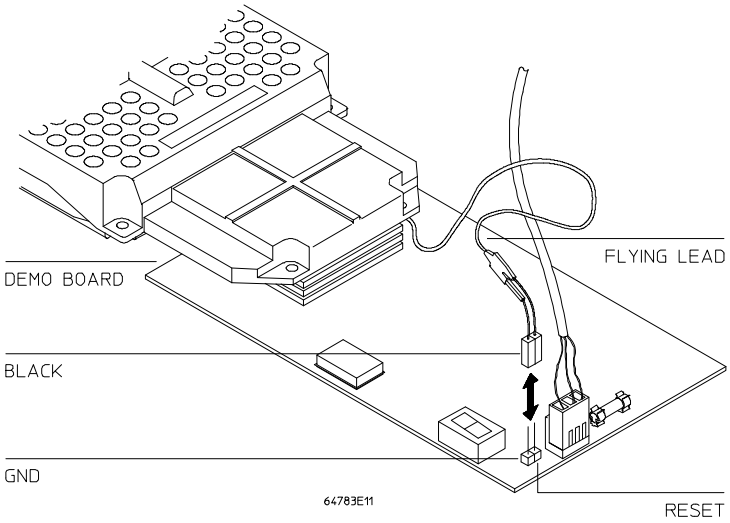
1 With HP 64700 power OFF, connect the emulator probe to the demo target system. When you install the probe into the demo board, be careful not to bend any of the pins. Do not insert the probe of the MC68040 emulator into the demo board socket incorrectly. Be very careful.



2 Connect the power supply wires from the emulator to the demo target system. The 3-wire cable has one power wire and two ground wires. **When attaching the 3-wire cable to the demo target system, make sure the connector is aligned properly so that all three pins are connected**.



3 Connect the reset flying lead from the probe to the demo board



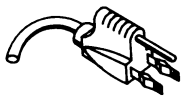
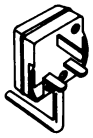
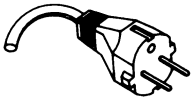
Step 6. Apply power to the HP 64700

The HP 64700B automatically selects the 115 Vac or 220 Vac range. In the 115 Vac range, the HP 64700B will draw a maximum of 345 W and 520 VA. In the 220 Vac range, the HP 64700B will draw a maximum of 335 W and 600 VA.

The HP 64700 is shipped from the factory with a power cord appropriate for your country. You should verify that you have the correct power cable for installation by comparing the power cord you received with the HP 64700 with the drawings under the "Plug Type" column of the following table.

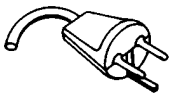
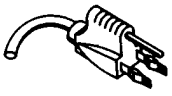
If the cable you received is not appropriate for your electrical power outlet type, contact your Hewlett-Packard sales and service office.

Power Cord Configurations

Plug Type	Cable Part No.	Plug Description	Length in/cm	Color
Opt 903 124V ** 	8120-1378 8120-1521	Straight * NEMA5-15P 90°	90/228 90/228	Jade Gray Jade Gray
Opt 900 250V 	8120-1351 8120-1703	Straight * BS136A 90°	90/228 90/228	Gray Mint Gray
Opt 901 250V 	8120-1369 8120-0696	Straight * NZSS198/ASC 90°	79/200 87/221	Gray Mint Gray
Opt 902 250V 	812001689 8120-1692 8120-2857	Straight * CEE7-Y11 90° Straight (Shielded)	79/200 79/200 79/200	Mint Gray Mint Gray Coco Brown

* Part number shown for plug is industry identifier for plug only.
Number shown for cable is HP part number for complete cable including plug.
** These cords are included in the CSA certification approval for the equipment.

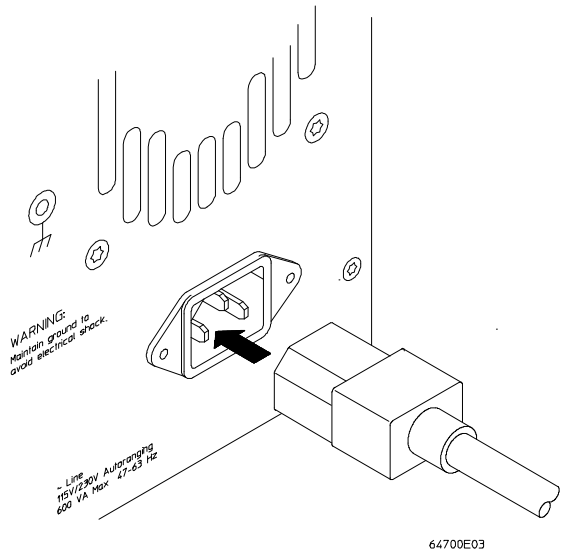
Power Cord Configurations (Cont'd)

Plug Type	Cable Part No.	Plug Description	Length in/cm	Color
Opt 906 250V 	8120-2104 8120-2296	Straight * SEV1011 1959-24507 Type 12 90°	79/20 79/200	Mint Gray Mint Gray
Opt 912 220V 	8120-2957	Straight *DHCK107 90°	79/200 79/200	Mint Gray Mint Gray
Opt 917 250V 	8120-4600 8120-4211	Straight SABS164 90°	79/200 79/200	Jade Gray
Opt 918 100V 	8120-4753 8120-4754	Straight Miti 90°	90/230 90/230	Dark Gray

* Part number shown for plug is industry identifier for plug only.
Number shown for cable is HP part number for complete cable including plug.
** These cords are included in the CSA certification approval for the equipment.

1 Connect the power cord and turn on the HP 64700.

The line switch is a pushbutton located at the lower, left-hand corner of the front panel. To turn ON power to the HP 64700, push the line switch pushbutton in to the ON (1) position. The power light at the lower, right-hand corner of the front panel will be illuminated.



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. Installation involves installing the interface and the 64700 Operating Environment. These instructions also tell you how to prevent installation of the Graphical User Interface if you want to use only the 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

During the installation process, you may make some choices about how much of the product software to load from the product media. As a general rule, you should load everything from the media because that ensures that you will not miss filesets and therefore have problems with the operation of the software. However, you may not need or want to install certain partitions or filesets from the product media when installing the Graphical User Interface. There are at least two reasons why that is so.

- If you were shipped the HP 64801 64000-UX Operating Environment instead of the HP B1471 64700 Operating Environment, you were shipped files that are not necessary to the operation of HP 64700 Series interface products. As of this writing, excluding these files will save you about 4.5 megabytes of disk space.
- You may not have the system performance necessary, or you may choose, for some other reason, not to use the Graphical User Interface and instead use the Softkey Interface. If that is the case, you should exclude the filesets that contain the Graphical User Interface because:
  - As of this writing, you will save about 3.5 megabytes of disk space.

– If you are using X Windows, the Graphical User Interface is the default interface. If you load the Graphical Interface, but do not use it, you will have to manually override it each time you begin an emulation session.

The following sub-steps assume that you may want to exclude partitions or filesets. Perform the following sub-steps to load the software on your system:

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 your HP-UX System Administrator documentation if you need to modify these values.

7 Select the choice on the update menu that allows you to view the product partitions.

8 If you see the partition named 64801 and described as "64000-UX Operating Environment", then you have the HP 64801 product. In this case, mark the 64801 partition with "n" to prevent loading this partition. Do this only if you have 64700 Series emulators and do not use "meas-sys".

9 Except for the emulator-specific partition and the 64700 Operating Environment partition, mark all other partitions with "y" to confirm that you want these partitions loaded. (Do not mark 64801 with "y" if you marked it with "n" in the last step.)

The emulator-specific partition will be named something like "<processor> Emulation Tools" where "<processor>" is the processor supported by the emulator.

10 If you plan to install and use the Graphical User Interface, do the following:

- Mark the emulator-specific partition and the 64700 Operating Environment partition with "y" to confirm the installation of these partitions.
- Skip to sub-step 12 of these instructions.

11 If you do not want to install the Graphical User Interface, do the following:

- View the filesets for the 64700 Operating Environment.
- Mark the fileset **64700XUI** with "n" to exclude it from installation.
- Mark all other filesets in the partition with "y" to confirm installation.
- Return to the partition screen.
- View the filesets in the emulator-specific partition (named something like <processor-type> Emulation Tools).
- Mark the fileset **647xxXUI** with "n" to exclude it from installation.
- Mark all other filesets in the partition with "y" to confirm installation.
- Return to the partition screen.

12 From the partition screen, choose the update utility softkey that starts the installation process.

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 installation process.

1 Verify the existence of the **HP64_Softkey** file in the **/usr/lib/X11/app-defaults** subdirectory by entering
ls /usr/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/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, because you installed the Graphical User Interface for the 68040 emulator, resource name strings will have **m68040** 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. Chapter 13, "Using X Resources", also discusses 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.

HP VUE differs slightly from other window managers in that it does not read your .Xdefaults file to find resources you may want to customize. Instead, it uses resources from the X resource database. In order to customize resources for the Graphical User Interface under HP VUE therefore, you must either merge a file of customized resources with the X resource database, or set an environment variable that causes the X resource manager to read a file of customized resources. For ease of use, choose the *.Xdefaults* file as your merge file.

- To merge the file *.Xdefaults* with the X resource database, enter

xrdb -merge .Xdefaults

at the HP-UX prompt.

Customized resources will be merged with the X resource database and will be available for retrieval by the Graphical User Interface.

- To enable the Graphical User Interface to find the *.Xdefaults* file directly, enter the following commands:

  XENVIRONMENT=$HOME/.Xdefaults

  export XENVIRONMENT

  The Graphical User Interface will be able to find and read the file in order to retrieve customized resources.

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. Installation involves installing the interface and the 64700 Operating Environment. These instructions also tell you how to prevent installation of the Graphical User Interface if you want to use only the 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 plan to use the Softkey Interface instead of the Graphical User Interface, you can save about 3.5 megabytes of disk space by not installing the filesets suffixed "XUI". (Also, if you choose not to install the Graphical User Interface, you will not have to use a special command line option to start the Softkey Interface.)

Refer to the *Software Installation Notice* for software installation instructions. After you are done installing the software, return here.

Step 2. Start the X server and OpenWindows

If you are not already running the X server, do so now. The X server is required to run the Graphical User Interface because it is an X application.

- Start the X server by entering **/usr/openwin/bin/openwin** at the UNIX prompt.

  Consult the OpenWindows documentation if you do not know about using OpenWindows and the X server.

Step 3. Set the necessary environment variables

The DISPLAY environment variable must be set before the Graphical User Interface will start. Also, you should modify the PATH environment variable to include the "usr/hp64000/bin" directory, and, if you have installed software in a directory other than "/", you need to set the HP64000 environment variable.

The following instructions show you how to set these variables at the UNIX prompt. Modify your ".profile" or ".login" file if you wish these environment variables to be set when you log in. The following instructions also assume that you're using "csh"; if you're using "sh", environment variables are set in the "<VARIABLE>=<value>; export <VARIABLE>" form.

1 The DISPLAY environment variable is usually set by the **openwin** startup script. Check to see that DISPLAY is set by entering

echo $DISPLAY

If DISPLAY is not set, you can set it by entering

setenv DISPLAY=<hostname>:<server_number>.<screen_number>

For example:

setenv DISPLAY=myhost:0.0

Consult the OpenWindows documentation for an explanation of the DISPLAY environment variable.

2 Set the HP64000 environment variable.

For example, if you installed the HP 64000 software relative to the root directory, "/", you would enter

setenv HP64000 /usr/hp64000

If you installed the software relative to a directory other than the root directory, it is strongly recommended that you use a symbolic link to make the software appear to be under /usr/hp64000. For example, if you installed the software relative to directory /users/team, you would enter

ln -s /users/team/usr/hp64000 /usr/hp64000

If you do not wish to establish a symbolic link, you can set the HP64000 variable to the full path that contains the HP 64000 software; also set the LD_LIBRARY_PATH variable to the directory containing run-time libraries used by the HP 64000 products. Again, if you installed relative to /users/team, you would enter

setenv HP64000 /users/team/usr/hp64000
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${HP64000}/lib

3 Set the PATH environment variable to include the **usr/hp64000/bin** directory by entering

setenv PATH ${PATH}:${HP64000}/bin

Including **usr/hp64000/bin** in your PATH relieves you from prefixing HP 64700 executables with the directory path.

4 Set the MANPATH environment variable to include the **usr/hp64000/man** and **usr/hp64000/contrib/man** directories by entering

setenv MANPATH ${MANPATH}:${HP64000}/man
setenv MANPATH ${MANPATH}:${HP64000}/contrib/man

Including these directories in your MANPATH variable lets you access the 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 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
TERMINFO=/usr/hp64000/lib/terminfo
export TERMINFO
```

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).

Step 6. Restart the window system

1 Exit the window system you are using.

- If you are using SunView windows, use the Exit SunView menu item.
- If you are using Open Windows, use the Exit item in the Workspace menu.

2 Start your window system again.

- To restart SunView windows, type: sunview
- To restart Open Windows windows, type: openwin

The new function key settings are now active

Step 7. Run the interface in a window

Here are two ways to start the interface:

- In an OpenWindows or SunView shelltool window, type:

```
emul700 <logical_name>
```

- If you are running OpenWindows, you can use the installed script to open an xterm window:

```
$ /usr/hp64000/bin/hp64term
```

Then in the new window, type:

```
$ emul700 <logical_name>
```

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.
- Step through the demo by using the action keys.
- Exit the emulator/analyzer interface.
- Run performance verification.

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 68040 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:     em68040     m68040      21.17.9.143
```

A typical entry for a 68040 emulator connected to an RS-422 port would appear as follows:

```
#-------------------------------------------------------------------------------
#         |            |           |      |            |Xpar|Parity|Flow|Stop|Char
# Channel| Logical     | Processor | Host |  Physical  |Mode|      |    |Bits|Size
#  Type  |  Name       |   Type    | Name |   Device   |    |      |XON |    |
#         |            |           |      |            |OFF | NONE |RTS | 2  | 8
#-------------------------------------------------------------------------------
  serial:   em68040     m68040    myhost /dev/emcom23 OFF   NONE  RTS   2    8
```

Step 2. Start the interface with the emul700 command

The **emul700** software is part of the HP 64700 Operating Environment software you installed during the update process.

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 the **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 Softkey Interface.

You should include an ampersand ("&") with the command to start the Graphical User Interface as a background process. Doing this 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 Softkey Interface under X Windows, but you must include a command line argument to **emul700** to override the default Graphical User Interface. Start the Softkey Interface by entering

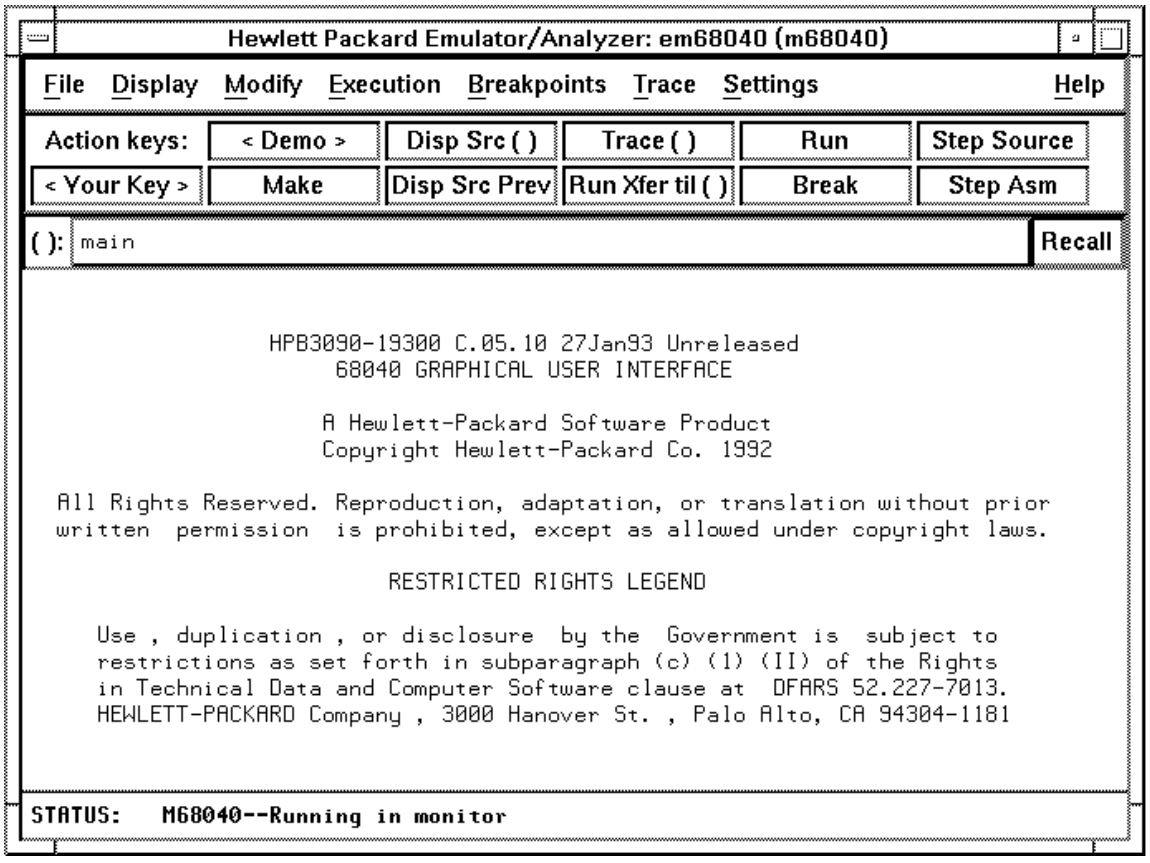
emul700 -u skemul <logical name>.

Example

Suppose you have discovered that the logical name for a 68040 emulator connected to the LAN is "m68040". To start the Graphical User Interface and begin communicating with that emulator, enter (assuming your $PATH includes **$HP64000/bin**)

```
emul700 m68040
```

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: em68040 (m68040)
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 til ( ) Break Step Asm
( ): main Recall
HPB3090-19300 C.05.10 27Jan93 Unreleased
68040 GRAPHICAL USER INTERFACE
A Hewlett-Packard Software Product
Copyright Hewlett-Packard Co. 1992
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
STATUS: M68040--Running in monitor

Step 3. Step through the demo with the Action Keys

Action keys are unique to the Graphical User Interface. When you first install and start the interface, a set of default Action Keys will be present. These Action Keys will let you load and start the emulator and run the demo program supplied with the interface. Doing so will help you familiarize yourself with the Graphical User Interface.

- Click on the <Demo> Action Key. A window of instructions will appear, telling you how to set up and run the demo program. Step through the demo program by clicking the Action Keys from left to right and top to bottom.

  After you have seen some of the capabilities by stepping through the demo, try the various menus, work with the command line, and read some on-line help topics. What you do is not important; it is important to begin getting comfortable with the interface.

Step 4. Exit the Graphical User Interface

After you have experimented with the interface, you can exit the session, or continue on to the next chapters and try some of the commands given. The following chapters give you instructions for performing various tasks with the interface, and so you do not have to work through them from front to back. Skip around to topics that interest you.

If, instead, you want to exit the interface now, do the following:

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.

Step 5. Verify the performance of the emulator

1 If you have a special configuration or session in progress, save it now. This procedure will cause your session to be lost.

2 Turn off power to the HP 64700 Card Cage.

3 Plug the emulator probe into the Demo Board if it is not already there.

4 Connect Demo Board power cable from the Demo Board to the HP 64700 Card Cage front panel. (See the diagrams under "Installing Hardware" in this chapter.)

5 Connect the Reset Flying Lead from the Emulation Probe to the Demo Board. (See "Step 4. Connect the emulator probe to the demo target system".)

6 Turn on power to the HP 64700 Card Cage.

7 Establish communication with the emulator from your host, and use the Softkey User Interface or Graphical User Interface.

8 On the command line, enter: **display pod_command**

9 On the command line, enter: **pod_command "pv 1"**

Don’t be alarmed if you notice an error message, such as: "I/O error; power down detected." If using the Graphical User Interface, the message, "Fatal System Error End Release System?" will also appear. These are normal conditions. The

performance verification tests will be complete, and you can read the results of the tests on screen.

Here's what caused the fatal error. To check the emulator status, your interface constantly polls the emulator in the card cage. The last step in the performance verification procedure reinitializes the card cage with the power down option. During initialization, the card cage is unable to communicate with your interface. When the polling signals of the interface are unanswered, the interface assumes communication has been lost between it and the card cage. Then the interface reports the fatal error, but no harm has been done. You must "end_release_system" in the Softkey User Interface, or click on OK in the Fatal System Error box in the Graphical User Interface.

If you are using a LAN, there is another way to run performance verification without the loss of communication between the interface and the emulator hardware. It uses the telnet capability and the Terminal Interface:

1 From your host computer, enter the command: **telnet <emulator_name>**.

2 Now enter the command: **pv 1**

3 When your performance verification test is complete, use the keyboard **<CTRL>d** keys to end the emulation session.

Examples

Start the performance verification test routines from the Softkey User Interface with the commands:

```
display pod_command

pod_command "pv <N>"
```

where **<N>** is an integer that specifies the number of times to run the set of performance verification tests. It is sometimes necessary to enter the pod command **tcf -e** before running performance verification.

A message similar to the following should appear:

```
Testing:  HP64783 Motorola 68040 Emulator
     PASSED:
  Number of tests:  1           Number of failures:  0
Testing:  HP64740 Compatible (PPN: 64794A) Deep Emulation Analyzer
     PASSED:
  Number of tests:  1           Number of failures:  0

                Copyright (c) Hewlett-Packard Co. 1987
All Rights Reserved.  Reproduction, adaptation, or translation
without prior
```

```
written permission is prohibited, except as allowed under copyright
laws.

  HP64700B Series Emulation System
    Version:   B.01.00 20Dec93
    Location:  Flash
    System RAM:1 Mbyte

  HP64783 Motorola 68040 Emulator
  HP64740 Compatible (PPN: 64794A) Deep Emulation Analyzer
```

If you have an emulation failure, you can replace the assembly that failed through your local Hewlett-Packard representative, and through the Support Materials Organization (SMO). Refer to the list of replacable parts at the end of this chapter.

To verify installation of memory modules in the deep analyzer card or in the emulation probe, choose **Settings→Pod Command Keyboard**, and on the command line, type: **ver**.

```
HP64783 Motorola 68040 Emulator
  Version:  A.04.00 22Oct92
  Control:  HP64748C ABG Control Board
  Speed:    33 MHz
  Memory:   260 Kbytes
    Bank 0: HP64172A (20ns) 256 Kbyte Memory Module

  HP64740 compatible (PPN: 64794A) Deep Emulation Analyzer
    Version:   A.03.00 25Jun93 Unreleased
    PC Board:  794-01-A1
    Depth:     80ch X 1K states selected, 80ch x 64K states available
      Bank 0:    HP64172A (20ns) 256 Kbyte Module
      Bank 1:    HP64172A (20ns) 256 Kbyte Module
      Bank 2:    HP64172A (20ns) 256 Kbyte Module
      Bank 3:    HP64172A (20ns) 256 Kbyte Module
```

What is pv doing to the Emulator?

The performance verification procedures provide a thorough check of the functionality of all of the products installed in the HP 64700 Card Cage. The Test Suite for the HP 64783A/B Emulator consists of the following modules.

```
Tests available in Emulator Subsystem:
  test # 1 (ABG68000 RAM)
  test # 2 (ABG Type Map)
  test # 3 (ABGDeMMU Map)
  test # 4 (low DMMU RAM)
  test # 5 (up DMMURAM)
  test # 6 (68000 side RAM)
  test # 7 (Host DPRAM)
  test # 8 (Clock Test)
  test # 9 (Release tobg)
  test #10 (Release to fg)
  test #11 (MonTransistion)
  test #12 (Break Detection)
```

```
test #13(Dual Port RAM)
test #14 (Emul Mem Bank 0)
test #15(Emul Mem Bank 1)
test #16 (Demo Reset)
test #17(Demo Data)
test #18 (Demo Address)
test #19 (DemoStatus)
test #20 (Demo IPL)
test #21 (Demo Cache)
test #22 (Demo DMA)
test #23 (Demo MDIS)
test #24(Demo LED)
test #25 (Analysis Intrfc)
test #26(DeMMUer)
test #27 (CMB)
```

Troubleshooting

The test results for all of these modules are indicated by a simple PASS/FAIL message. The PASS message gives a high level of confidence that all major functions and signals are operating because it includes a loopback test that includes read and write tests to the demo board. The demo board also stimulates inputs to the emulator.

A FAIL message on the other hand indicates that one or more of the tested functions is NOT working. In this event, an HP field representative can either swap assemblies to isolate the failure to an individual board, or replace all the major assemblies shown in the replaceable parts list. The emulation memory modules and plastic cover are not part of the probe assembly. The emulation memory modules must be ordered separately and the plastic covers should be removed from the probe assembly before replacing the probe assembly.

Parts List

What is an Exchange Part?

Exchange parts are shown on the parts list. A defective part can be returned to HP for repair in exchange for a rebuilt part.

Probe (exchange)

The Probe for the HP 64783A is not interchangable with the Probe for the HP 64783B. Make sure you order the Probe replacement part number that is compatible with your emulator.

To replace the Probe on the exchange program, you must remove certain parts, and return only that part considered an exchange part. When returning the Probe, you must remove the:

- cable assembly.
- top and bottom plastic covers.
- SRAM modules.
- demo board.

Emulation Control Card (exchange)

To replace the Emulation Control Card on the exchange program, you must remove certain parts, and return only that part considered an exchange part. When returning the Emulation Control Card, you must remove the:

- ribbon cable that connects the Emulation Control Card to the analyzer card.
- cable assembly.
- egress panel.

Main Assembly **Component Part**	**New**	**Exchange**
HP 64783A/B Probe and Demo Board		
68040 Emulator Firmware Floppy	64783-18000	
64700 SW UTIL	64700-18006	
MC68040 Probe Board for HP 64783A	64783-66504	64783-69504
MC68040 Probe Board for HP 64783B	64783-66505	64783-69505
(Order the following parts separately:)		
Top Plastic Cover	64783-04101	
Bottom Plastic Cover	64783-04102	
Plastic Rivets Kit (rivets and washers)	64748-68700	
Reset Flying Lead	64762-61602	
HP 64783A Demo Board for HP 64783A/B	64783-66502	
(Order the following part separately:)		
External Power Cable	5181-0201	
HP 64748C Emulation Control Card Subassembly		
Egress Panel	64748-00205	
Bracket (used with Egress Panel)	64748-01201	
Spacer, Hex M3X6	0515-1146	
Screw, Machine M3X8	0515-0372	
Cable-100 36"	64748-61601	
Cable-100 37"	64748-61602	
Cable-100 38"	64748-61603	
Cable Clamp	64744-01201	
Rubber Strip	64744-81001	
Emulation Control Card (without external cable or egress panel)	64748-66515	64748-69515
Wrist strap	9300-1405	
HP 64172A 256 Kbyte SRAM Module	64172A	64172-69501
HP 64172B 1 Mbyte SRAM Module	64172B	64172-69502
HP 64173A 4 Mbyte SRAM Module	64173A	64173-69501

Main Assembly **Component Part**	**New**	**Exchange**
HP 64794A Emulation-Bus Analyzer (deep) card	64794-66502	64794-69502
34-pin ribbon cable	64708-61601	
Analyzer Card HP 64740 with 1K memory depth	64740-66526	64740-69526
34-pin ribbon cable	64772-61602	

20

Installing/Updating Emulator Firmware

Installing/Updating Emulator Firmware

If you ordered the HP 64783A/B MC68040 emulator probe and the HP 64748C emulation control card together, the control card contains the correct firmware for the emulator.

However, if you ordered the emulator probe and the HP 64748C separately, or if you are using an HP 64748C that has been used previously with a different emulator probe, you must download the correct firmware into the emulation control card.

The 68040 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 68040 emulator probe on MS-DOS format floppies. The floppies are for users who 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 19, "Installation and Service".

This chapter shows you how to:

- Update firmware with the "progflash" command.
- Display current firmware version information.

To update emulator firmware with "progflash"

- Enter the **progflash -v <emul_name> <products ...>** command.

  The **progflash** command downloads code from files on the host computer into Flash EPROM memory in the HP 64700.

  The **-v** option (verbose) 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, such as m68040.

  The <products> option names the products whose firmware is to be updated, such as 64783.

  If you enter the **progflash** command without options, the downloading process becomes interactive. If you don't include the <emul_name> option, your computer displays the logical names in the /usr/hp64000/etc/64700tab.net file and asks you to choose one. If you don't include the <products> option, your computer displays the products that have firmware update files on the system and asks you to choose one. (In the interactive mode, only one product at a time can be updated.) You can abort the interactive **progflash** command by pressing <CTRL>c.

  progflash will print "Flash programming SUCCEEDED" and return 0 if it is successful; otherwise, it will print "Flash programming FAILED" and return a nonzero (error).

  You can verify the update by displaying the firmware version information.

Examples

To update the HP 64783 firmware in the HP 64700 that contains the "em68040" emulator:

```
$ progflash


    HPB1471-19309 A.05.00 03Jan94
  64700 SERIES EMULATION COMMON FILES

   A Hewlett-Packard Software Product
   Copyright Hewlett-Packard Co. 1988

  All Rights Reserved. Reproduction, adaptation, or translation without prior
  written  permission  is prohibited, except as allowed under copyright laws.

                        RESTRICTED RIGHTS LEGEND

     Use , duplication , or disclosure  by the  Government is  subject to
     restrictions as set forth in subparagraph (c) (1) (II) of the Rights
     in Technical Data and Computer Software clause at  DFARS 52.227-7013.
     HEWLETT-PACKARD Company , 3000 Hanover St. , Palo Alto, CA 94304-1181

           Logical Name        Processor
         1 em68k               m68000
         2 em80960             i80960
         3 m68040              m68040

Number of Emulator to Update? (intr (usually cntl C or DEL) to abort)
```

To update firmware in the HP 64700 that contains the 68040 emulator, enter "3".

```
           Product
         1 64700
         2 64703/64704/64706/64740
         3 64744
         4 64760
         5 64783

Number of Product to Update? (intr (usually cntl C or DEL) to abort)
```

To update the HP 64783 68040 emulator firmware, enter "5".

```
Enable progress messages? [y/n] (y)
```

To enable status messages, enter "y".

```
Checking System firmware revision...
Mainframe is a 64700B

Reading configuration from '/usr/hp64000/inst/update/64783.cfg'
ROM identifier address = 2FFFF0H
Required hardware identifier = 1FFFH,1FFCH
Control ROM start address = 280000H
Control ROM size = 40000H
Control ROM width = 16
Programming voltage control address = 2FFFFEH
Programming voltage control value = FFFFH
Programming voltage control mask = 0H

Rebooting HP64700...
Checking Hardware id code...
Erasing Flash ROM
Downloading ROM code: /usr/hp64000/inst/update/64783.X
    Code start 280000H (should equal control ROM start)
    Code size 29A3EH (must be less than control ROM size)
Finishing up...

Rebooting HP64700...
Flash programming SUCCEEDED
$
```

You could perform the same update as in the previous example with the following command:

```
$ progflash -v m68040 64783
```

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
  pod_command "ver"

Examples

The Terminal Interface **ver** command displays information similar to:

```
                Copyright (c) Hewlett-Packard Co. 1987
All Rights Reserved.  Reproduction, adaptation, or translation without prior
written permission is prohibited, except as allowed under copyright laws.

  HP64700B Series Emulation System
    Version:   B.01.00 20Dec93
    Location:  Flash
    System RAM:1 Mbyte

  HP64783 Motorola 68040 Emulator
    Version:   A.01.00 28Jan93
    Control:   HP64748C  Emulation Control Board
    Speed:     40 MHz
    Memory:    260 Kbytes
      Bank 0:  HP64172A (20ns) 256 Kbyte Memory Module

  HP64740 Emulation Analyzer
    Version:   A.02.02 13Mar91
```

If there is a power failure during a firmware update

If there is a power glitch during a firmware update, some bits may be lost during the download process, possibly resulting in an HP 64700 that will not boot up.

- ☐ Repeat the firmware update process.

- ☐ If the HP 64700 is connected to the LAN in this situation and you are unable to connect to the HP 64700 after the power glitch, try repeating the firmware update with the HP 64700 connected to an RS-232 or RS-422 interface.

Glossary

Absolute Count

in the trace list count column, an absolute count indicates the total count accumulated between the displayed state and the trigger state. For example, an absolute time count shown beside trace memory line number 100 indicates the elapsed time between capture of the trigger state and capture of state 100.

Absolute File

a file consisting of machine-readable instructions in which absolute addresses are used to store instructions, data, or both. These are the files that are generated by the compiler/assembler/linker and are loaded into HP 64700 Series emulators.

Access Breakpoint

a break from execution of your target program to execution of the emulation monitor when the emulator detects an access violation, such as an attempt to write to ROM or guarded memory space. The same effect can be obtained for an emulation break due to trigger recognition within the emulation-bus analyzer, or due to a signal from an external device supplied over the CMBT or the rear-panel BNC. Access breakpoints do not obtain immediate transfer to the monitor program. Several instruction cycles may be executed after the access violation occurs before execution begins in the monitor. Refer to Chapter 4, "Using the Emulator", for details of how to use breakpoints, and effects of their use on execution of your target program. Also, refer to Execution Breakpoints in this glossary.

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. No external analyzer is supported by the MC68040 emulators because all analysis bits are used by the emulation-bus analyzer.

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 3 clocks, then the analyzer clock speed (bus cycle rate) is 21/3 = 7 MHz.

Applications

are composed of a hierarchy of widgets.

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

a memory that parallels (and overlaps) the emulation processor's normal address range. Entry to background can only take place under emulator control, and cannot be reached via your target program.

Background Monitor

a monitor program that operates entirely in the background address space. 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.

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.)

Class names

names that may apply to many instances of a widget.

Command File

a file containing a sequence of commands to be executed.

Compatible Mode

configures the deep 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.

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 64306 IMB/CMB Interface to allow coordinated measurements.

Counter Overflow

when the trace tag counter of the deep analyzer reaches maximum count and begins a new count from zero. The counter in 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).

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, "deep analyzer" refers to the HP 64794 Emulation-Bus Analyzer with deep trace memory.

Downloading

the process of transferring absolute files from a host computer into the emulator.

DTE (Data Terminal Equipment)

a specific RS-232C hardware interface configuration. Typically, DTE is a terminal or printer.

Emulation-bus Analyzer

a system component built into the HP 64700 that captures the emulation processor’s address, data, and status information.

Emulation Memory

high-speed memory (RAM) in the emulator that can be used in place of target system memory.

Emulator

a tool that replaces the processor in your target system. The goal of the emulator is to operate just like the processor it replaces. The emulator gives information about the bus cycle operation of the processor and control over target system execution. Using the emulator, you may view 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.

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. Refer to Chapter 4, "Using the Emulator", for details of how to use execution breakpoints, and effects of their use on execution of your target program. Also, refer to Access Breakpoints in this glossary.

Foreground

the directly addressable memory range of the emulation processor.

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.

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 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.

Instance Name

the name of a single, unique widget.

Inverse Assembler

a program that translates absolute code into assembly language mnemonics.

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. Refer to Chapter 10, "Using MC68030 Memory Management", for a detailed explanation.

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 system resources. For example, when you enter a command that requires access to your system resources, the system controller writes a command code to a storage area and breaks the execution of the emulation processor from your target program into the monitor. The monitor program then reads the command from the storage area and executes the processor instructions that access the target system. After the system resources have been accessed, execution returns to your target program.

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).

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. Refer to Chapter 9, "Solving Problems", for a detailed explanation.

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.

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

refers to the emulator configuration in which commands that temporarily interrupt target program execution (for example, display/modify target memory or processor registers) are not allowed.

Relative Count

in the trace list count column, a relative count 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.

Sequencer

a state machine in the analyzer that searches for execution of states in a particular order.

Single-step

the execution of one microprocessor instruction. Single-stepping the emulator allows you to view program execution one instruction at a time.

Softkey Interface

the host computer interface program used in the UNIX environment. The Softkey Interface is a friendly interface used to control HP 64700 emulators.

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 analyzer.

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. The trigger signals are called trig1 and trig2. They 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. For details of how to configure and use trig1 and trig2, refer to the chapter on "Tasks you can do with the deep analyzer" in this manual, and the chapter on making coordinated measurements in your emulator/analyzer manual(s).

Note that there is delay when you use trig1 and/or trig2 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.

Widget

an OSF/Motif graphic device from which X applications are built. For example, pushbuttons and menu bars are Motif widgets.

Window

a specified rectangular area of virtual space shown on the display in which data can be observed.

X Resource Specification

a resource name and value. The resource name identifies the element whose appearance or behavior is to be defined, and the value specifies how the element should look or behave.

1K Analyzer

in this manual, "1K analyzer" refers to the HP 64703, HP 64704, and HP 64706 Emulation-Bus analyzers with 1K trace memories.

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.

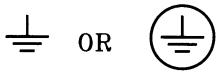
Safety Symbols Used In Manuals

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.

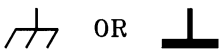
Hot Surface. This symbol means the part or surface is hot and should not be touched.

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.



rame or chassis terminal. A connection to the frame (chassis) of the equipment hich 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.