HEWLETT PACKARD

User’s Guide for the PC Interface

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

Notice

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.

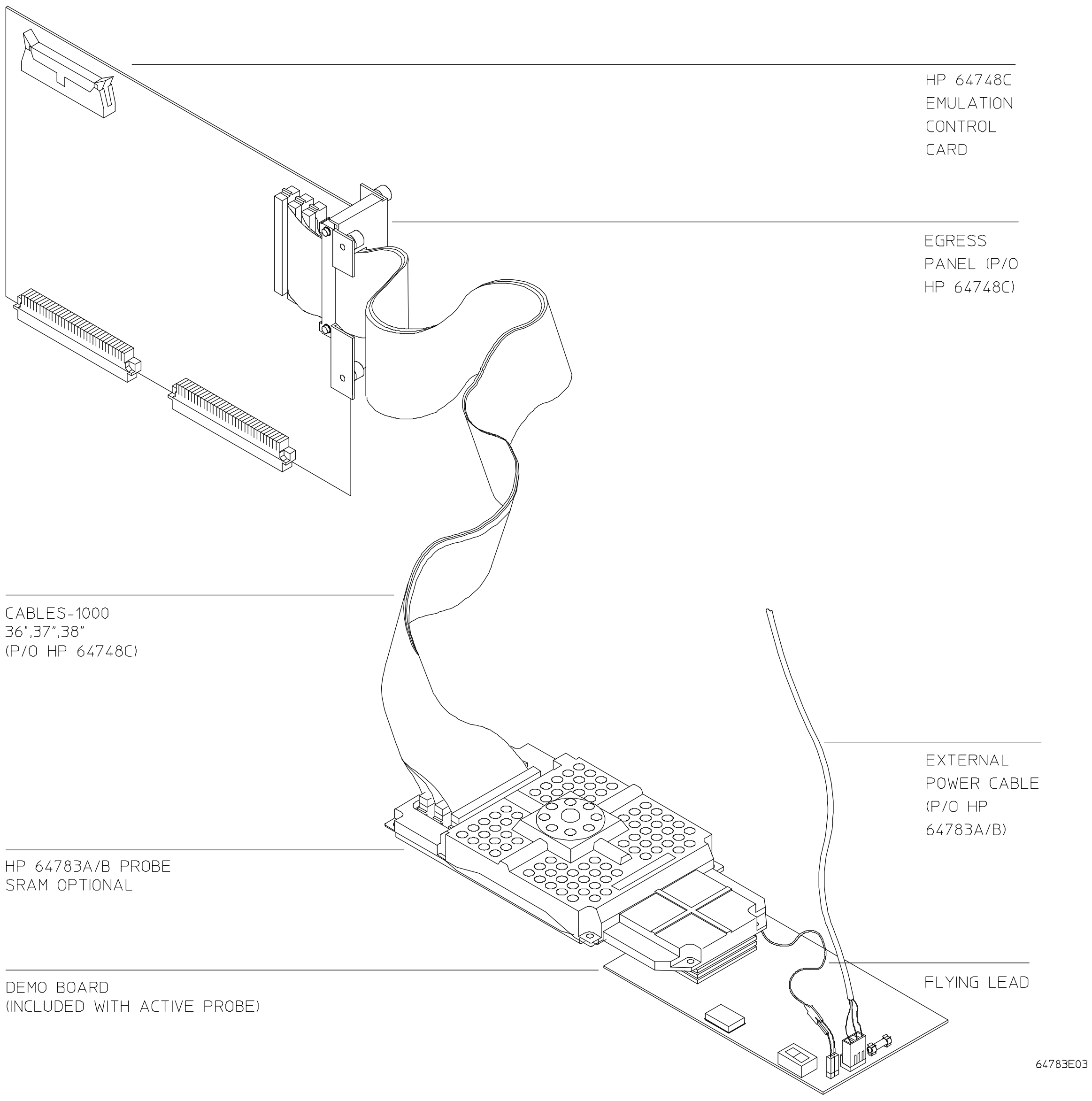
A software code may be printed before the date; this indicates the version level of the software product at the time the manual was issued. Many product updates and fixes do not require manual changes, and manual corrections may be done without accompanying product changes. Therefore, do not expect a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

Edition 1	64783-97001, April 1993
Edition 2	64783-97003, October 1993
Edition 3	**64783-97005, January 1994**

Safety and Certification and Warranty

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

The HP 64783A/B Emulator



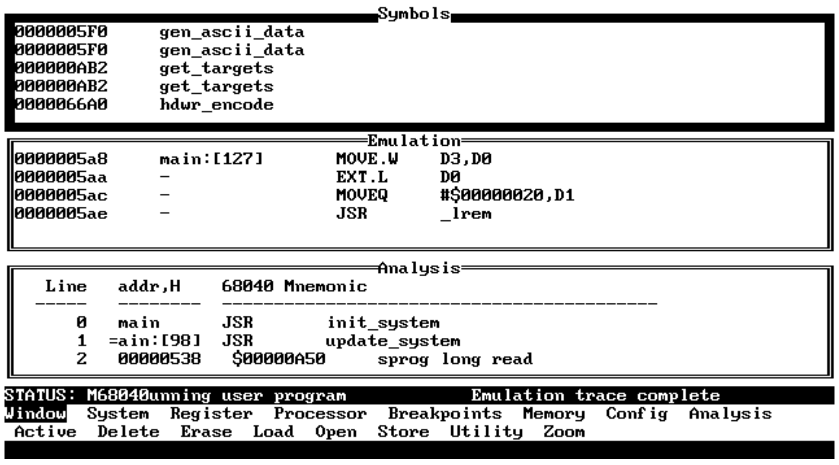
Symbols Display----------->

Emulation Memory Display---------------------->

Analysis Trace Display--->

Status line gives active information---------------->

Command selection area activates the function with your input



```
                                        Symbols
0000005F0      gen_ascii_data
0000005F0      gen_ascii_data
000000AB2      get_targets
000000AB2      get_targets
0000066A0      hdwr_encode

                                       Emulation
0000005a8      main:[127]       MOVE.W     D3,D0
0000005aa      -                EXT.L      D0
0000005ac      -                MOVEQ      #$00000020,D1
0000005ae      -                JSR        _lrem

                                        Analysis
   Line    addr,H    68040 Mnemonic
  -----   --------   -----------------------------------------
      0    main      JSR       init_system
      1   =ain:[98]  JSR       update_system
      2    00000538   $00000A50      sprog long read
STATUS: M68040unning user program            Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Active  Delete  Erase  Load  Open  Store  Utility  Zoom
```

An Example of the PC Interface in Use

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 the chapter titled "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.

The emulators plug into the modular HP 64700 instrumentation card cage and offer 80 channels of processor bus analysis with the HP 64704A 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
- Post-processed dequeued trace with symbols
- Eight events, each consisting of address, status, and data comparators
- Events may be sequenced eight levels deep and can be used for complex trigger qualification and selective store

Emulation memory

- 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 book covers the HP 64783A/B emulator. All information in the manual applies to the three versions of the emulated microprocessor, unless it is marked with the processor name (MC68040, MC68EC040, or MC68LC040).

Part 1, “Quick Start Guide,” is designed to quickly familiarize you with the emulator. It outlines the major steps required to install the emulator hardware and PC Interface software, and explains how to use the main emulator features.

Part 2, “User Guide,” provides detailed explanations on how to use the PC Interface commands to operate the emulator and analyzer. It also explains how to to configure the emulator to make measurements in your target system, and how to solve problems you may encounter when using the emulator.

Part 3, “Reference,” gives detailed information on such things as menu hierarchies, command syntax, data file formats, demo program information, and specifications. You will probably find this part of the book most helpful after you have become comfortable with your emulator.

Part 4, "Installation and Service Guide," shows you how to install and maintain the emulator.

Contents

Part 1

Quick Start Guide

The Emulation Process

The emulator is a powerful tool that can help you debug and integrate your target system hardware and software. There are three steps to the emulation process:

Develop Your Programs

Before you can use the emulator to debug your target system, you must have a target program to analyze. This may be developed on a host computer and programmed into target system ROM. Or, you can download programs into emulation memory, which allows testing, debugging and modification before the code is committed to hardware. When first learning how the emulator operates, you may want to use the demo program supplied with the emulator, rather than developing your own program, so you can get started quickly.

Configure the Emulator

Each target system has different resource requirements for memory and I/O locations. Also, there may be variations in system bus usage or chip configuration. The emulator configuration controls allow you to adapt the emulator to match the needs of your target system hardware and software. You usually define this configuration once; then change it only as your target system design definition changes.

Use the Emulator

After you configure the emulator, you can load the programs you want to test, run them, and make various measurements to verify their functionality. The emulator allows you to control program runs, display and modify memory and registers, and record program execution.

In This Part

Chapter 1, “Getting Started,” tells how to set up the emulator and how to begin making simple measurements. The chapter is organized as a practice tutorial. You can use the supplied demo program of the emulator to learn about emulator operation.

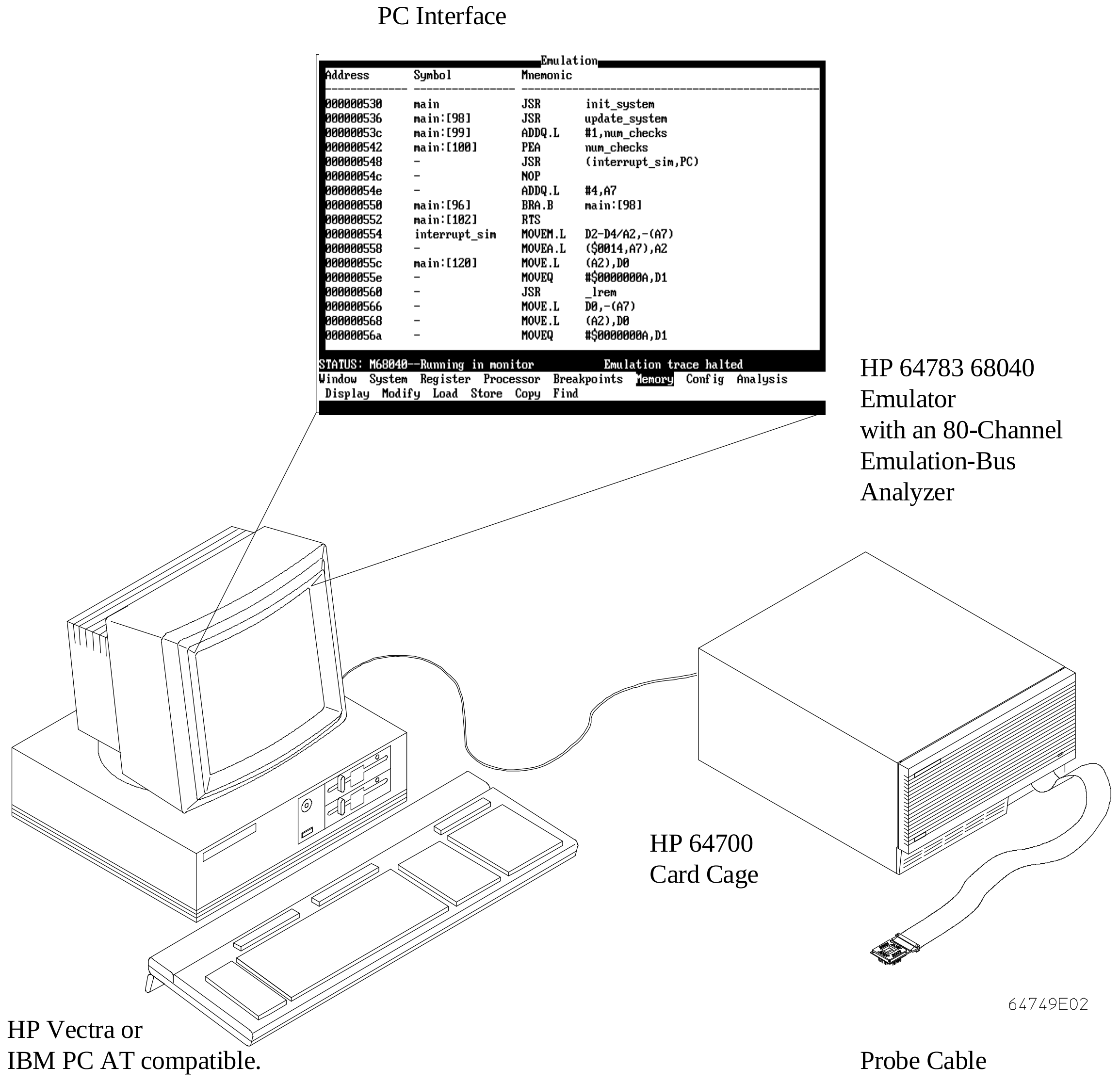
Chapter 2, “Troubleshooting,” contains tips on solving some of the more common problems that you may find when you begin using the emulator.

Part 1 of this book is designed to get you started using the major features of the emulator, as quickly as possible. Once you have familiarized yourself with basic operation of the emulator, Part 2 explains how to perform many common tasks using the PC Interface commands. Part 3 contains detailed reference information, such as a complete list of all commands, and a list of error messages. Part 4 provides installation and service information.

1

Getting Started

The 68040 Emulator — At a Glance


PC Interface
HP 64783 68040 Emulator with an 80-Channel Emulation-Bus Analyzer
HP 64700 Card Cage
64749E02
HP Vectra or IBM PC AT compatible.
Probe Cable
Window System Register Processor Breakpoints Memory Config Analysis
Display Modify Load Store Copy Find

```
                                   Emulation
Address       Symbol            Mnemonic
------------- ----------------- ------------------------------------------------
000000530     main              JSR        init_system
000000536     main:[98]         JSR        update_system
00000053c     main:[99]         ADDQ.L     #1,num_checks
000000542     main:[100]        PEA        num_checks
000000548     -                 JSR        (interrupt_sim,PC)
00000054c     -                 NOP
00000054e     -                 ADDQ.L     #4,A7
000000550     main:[96]         BRA.B      main:[98]
000000552     main:[102]        RTS
000000554     interrupt_sim     MOVEM.L    D2-D4/A2,-(A7)
000000558     -                 MOVEA.L    ($0014,A7),A2
00000055c     main:[120]        MOVE.L     (A2),D0
00000055e     -                 MOVEQ      #$0000000A,D1
000000560     -                 JSR        _lrem
000000566     -                 MOVE.L     D0,-(A7)
000000568     -                 MOVE.L     (A2),D0
00000056a     -                 MOVEQ      #$0000000A,D1

STATUS: M68040--Running in monitor           Emulation trace halted
```

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

- The HP 64700 emulation card cage is connected to the personal computer.
- The hardware has been installed in the card cage as described in the "Installation and Service" chapter of this manual.
- The emulation probe assembly is connected to the demo board assembly as shown in the "Installation and Service" chapter of this manual.
- The PC Interface software has been installed as described in the "Installing/Updating Emulator Firmware" chapter.
- The emulator contains emulation memory (at least one 256-Kbyte memory module in Bank0 of the emulation probe).

Step 1. Copy the demo program files

The demo program and the associated output files, including the IEEE-695 format absolute file, are provided with the 68040 PC Interface.

- Copy files from \HP64700\DEMO\64783.

  For example:

```
C> MKDIR DEMO040 <Enter>
C> CD DEMO040 <Enter>
C> COPY \HP64700\DEMO\64783\*.*. <Enter>
```

Step 2. Start the PC Interface

If you have set up the emulator device table and the **HPTABLES** shell environment variable as shown in the "Installation" chapter, you are ready to start up the PC Interface.

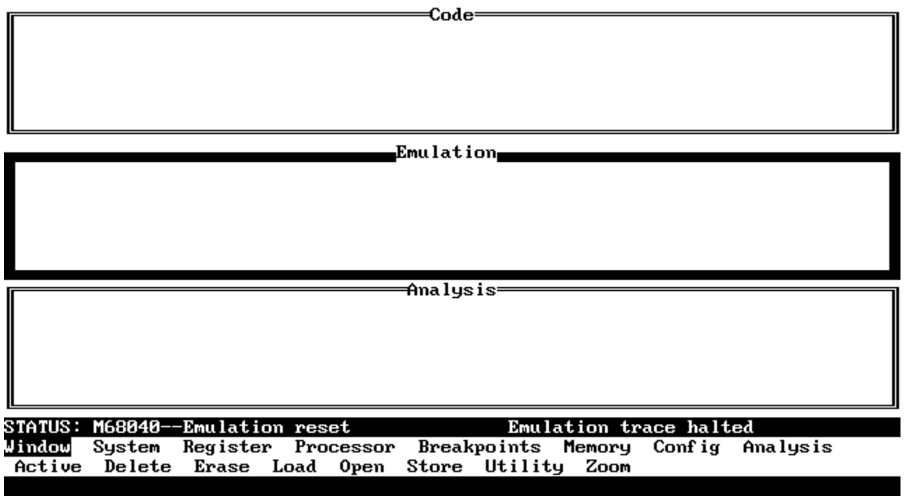
- Enter the PCM040 command at the MS-DOS prompt.

For example:

```
C:\DEMO040\> PCM040 EMUL_COM1 <Enter>
```

The EMUL_COM1 in the command above is the logical emulator name given in the HP 64700 emulator device table file (\HP64700\TABLES\64700TAB).

If the PC Interface starts successfully, you will see a display similar to the following.



```
STATUS: M68040--Emulation reset              Emulation trace halted
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Active  Delete  Erase  Load  Open  Store  Utility  Zoom
```

Step 3. Learn how to select commands

- Use the the left and right arrow keys to highlight the command or option and press the <Enter> key.

Or:

- Type the first letter of the command or option.

If you select the wrong option, you can press the <ESC> key to move back up the command tree.

When a command option is highlighted, either the next level of options or a short message describing the option is shown on the bottom line of the display.

Many PC Interface commands present fields in which to specify command parameters.

Step 4. Configure the emulator for the demo program

In the emulation configuration, you will set up the emulator to use its background monitor. This is the most simple monitor.

You will also define the address to be used as the initial stack pointer, and the address to be installed in the program counter.

- To obtain the emulation configuration form, enter the command:

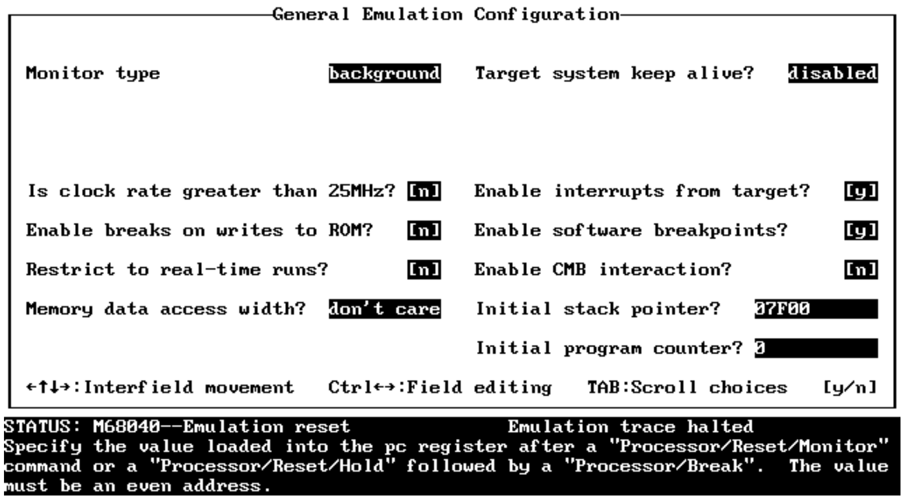
Config **G**eneral

Use the arrow keys to place the cursor in the "Monitor type" field, and press the <Tab> key to select "background".

Move the cursor to the "Initial stack pointer?" field and type in 07F00.

Move the cursor to the "Initial program counter?" field and type in 0.

To save your configuration, press the <Enter> key while the cursor is in the field in the lower right corner of the form.



```
                    General Emulation Configuration

Monitor type                   background   Target system keep alive?   disabled



Is clock rate greater than 25MHz? [n]   Enable interrupts from target?   [y]
Enable breaks on writes to ROM?   [n]   Enable software breakpoints?     [y]
Restrict to real-time runs?       [n]   Enable CMB interaction?          [n]
Memory data access width?  don't care   Initial stack pointer?   07F00
                                        Initial program counter? 0
←↑↓→:Interfield movement   Ctrl←→:Field editing   TAB:Scroll choices   [y/n]
STATUS: M68040--Emulation reset              Emulation trace halted
Specify the value loaded into the pc register after a "Processor/Reset/Monitor"
command or a "Processor/Reset/Hold" followed by a "Processor/Break".  The value
must be an even address.
```

Step 5. Map memory for the demo program

Because the emulator can use target system memory or emulation memory (or both), it is necessary to map ranges of memory so that the emulator knows where to direct its accesses. You can map up to eight memory ranges with 256-byte resolution (beginning on 256-byte boundaries and at least 256 bytes in length).

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

The "ecs" demo program occupies ROM locations from 0 through 2C66H and RAM locations from 6000H through 0FFFFH. (You can see this by looking at the linker load map output listing that was generated when compiling the demo program.) Note that the ecs.x demo program was developed for the MC68331/332 emulator. It will work well for the MC68040 emulator.

- Select the **C**onfig **M**ap **M**odify command to enter the memory map configuration screen.

To select the **C**onfig **M**ap **M**odify command: Press the "c" key, then press the "m" key, and finally press the "m" key again. The shorthand form used to describe this sequence is:

```
Config Map Modify
```

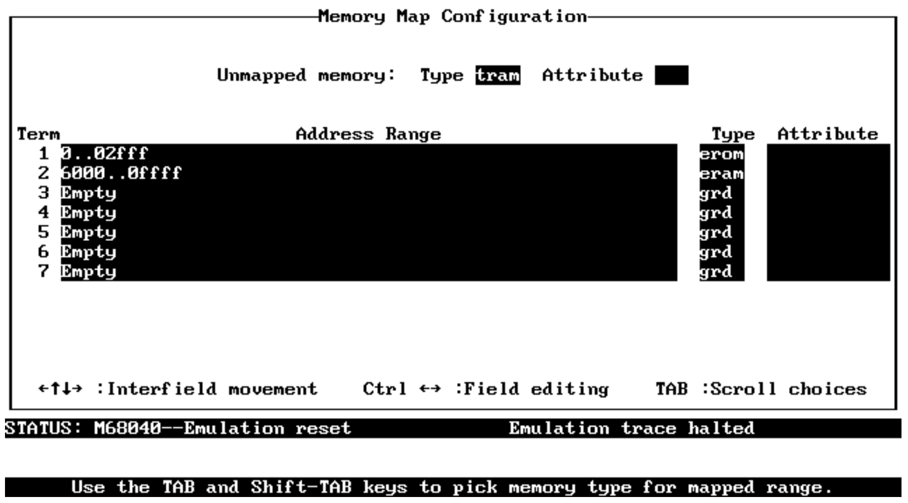
Using the arrow keys, move the cursor to the "address range" field of term 1. Enter:

```
0..02fff
```

Move the cursor to the "type" field of term 1, and press the <Tab> key to select the "erom" (emulation ROM) type. Next, move the cursor to the "address range" field of term 2. Enter:

```
6000..0ffff
```

Move the cursor to the "type" field of term 2, and press the <Tab> key to select the "eram" (emulation RAM) type. The memory configuration display follows.



```
───────────────────────Memory Map Configuration───────────────────────

              Unmapped memory:  Type tram  Attribute

Term                    Address Range                    Type  Attribute
  1 0..02fff                                             erom
  2 6000..0ffff                                          eram
  3 Empty                                                grd
  4 Empty                                                grd
  5 Empty                                                grd
  6 Empty                                                grd
  7 Empty                                                grd


  ←↑↓→ :Interfield movement   Ctrl ←→ :Field editing   TAB :Scroll choices
STATUS: M68040--Emulation reset            Emulation trace halted

     Use the TAB and Shift-TAB keys to pick memory type for mapped range.
```

The "Unmapped memory type" field in the memory map specifies that unmapped memory ranges are treated as target system RAM by default.

To save your memory map, use the <Enter> key to exit the field in the lower right corner. (The <End> key on PC keyboards moves the cursor directly to the last field.)

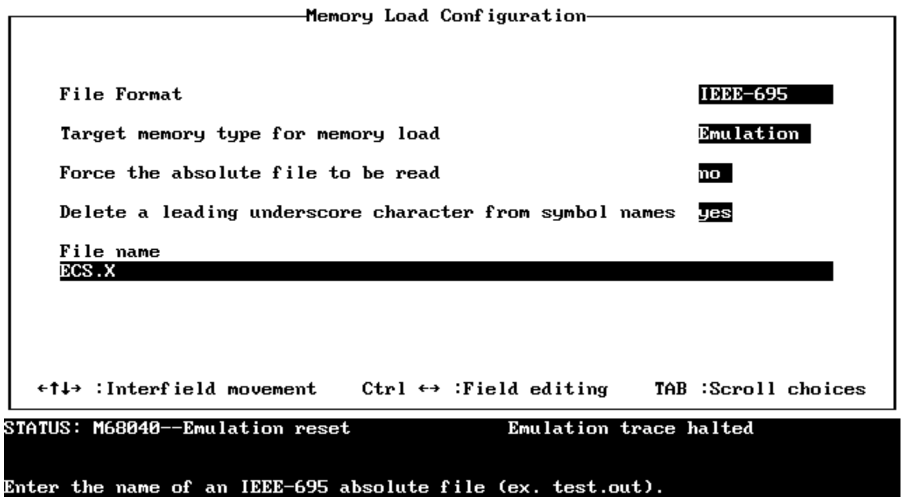
Step 6. Load the demo program absolute file

Some 68000 software development tools generate IEEE-695 format absolute files. The PC Interface provides an IEEE-695 format absolute file reader. However, you can also load absolute files in the following formats: HP absolute, Intel hexadecimal, Extended Tektronix hexadecimal, and Motorola S-records.

- Select the **M**emory **L**oad command.

For example, you can load the demo program's absolute file by selecting:

`Memory Load`



```
Memory Load Configuration

File Format                                                  IEEE-695
Target memory type for memory load                           Emulation
Force the absolute file to be read                           no
Delete a leading underscore character from symbol names      yes
File name
ECS.X

←↑↓→ :Interfield movement    Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: M68040--Emulation reset             Emulation trace halted

Enter the name of an IEEE-695 absolute file (ex. test.out).
```

The "ECS.X" absolute file is an IEEE-695 format file, so use the <Tab> key to select "IEEE-695".

The next field (again, use the arrow keys to move the cursor from field to field) allows you to selectively load the portions of the absolute file which reside in emulation memory, target system memory, or both emulation and target system memory. Since emulation memory is mapped for demo program locations, you can enter either "Emulation" or "Both".

The next field allows you to force the file format reader to be run before memory is loaded. For this example, enter "no".

Next, you can specify whether you want the IEEE-695 reader to strip leading underscores from symbols. For this example, enter "yes".

Finally, enter the name of your absolute file ("ECS.X" in this example) in the last field, and press <Enter> to start the memory download.

Step 7. Transfer symbols to the emulator

When you load memory with IEEE-695 or HP64000 format absolute files, global symbols are automatically loaded. However, before you can view symbols in mnemonic memory and trace displays, you must transfer them to the emulator.

1 To transfer global symbols into the emulator, select the **S**ystem **S**ymbols **G**lobal **T**ransfer command.

For example, you can transfer global symbols from the demo program to the emulator by selecting:

```
System Symbols Global Transfer
```

2 To transfer local symbols into the emulator, select the **S**ystem **S**ymbols **L**ocal **T**ransfer command.

For example, to transfer local symbols from all modules to the emulator, select:

```
System Symbols Local Transfer All
```

Step 8. Display the demo program in memory

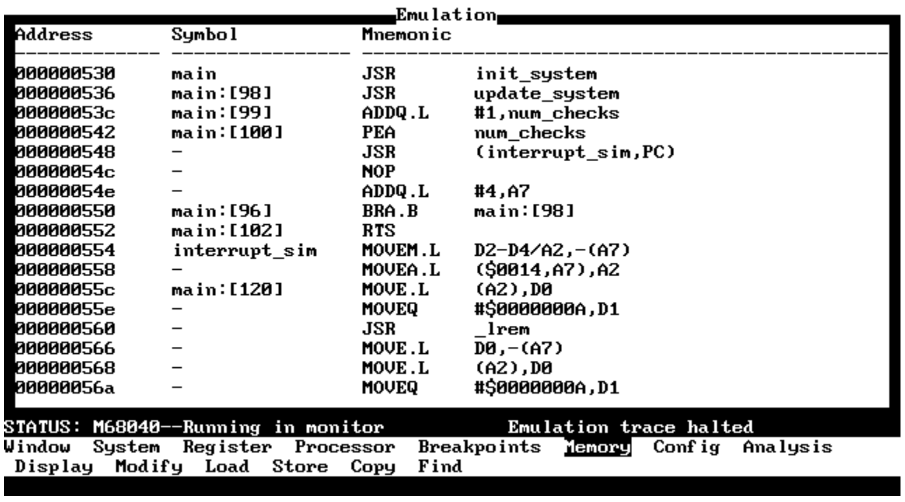
Once you have loaded a program into the emulator, you can view the program in memory by displaying memory in mnemonic format. This causes the contents of memory locations to be disassembled and displayed in assembly language mnemonic format.

- Select the **M**emory **D**isplay **M**nemonic command.

For example:

```
Memory Display Mnemonic
```

Enter the address range "main.." and press <Enter>. The emulation window automatically becomes the active window as a result of this command. You can press <CTRL>z to zoom the window and <Home> to see the beginning of the range.



```
                                      Emulation
Address        Symbol            Mnemonic
-------------  ----------------  ------------------------------------------------
000000530      main              JSR        init_system
000000536      main:[98]         JSR        update_system
00000053c      main:[99]         ADDQ.L     #1,num_checks
000000542      main:[100]        PEA        num_checks
000000548      -                 JSR        (interrupt_sim,PC)
00000054c      -                 NOP
00000054e      -                 ADDQ.L     #4,A7
000000550      main:[96]         BRA.B      main:[98]
000000552      main:[102]        RTS
000000554      interrupt_sim     MOVEM.L    D2-D4/A2,-(A7)
000000558      -                 MOVEA.L    ($0014,A7),A2
00000055c      main:[120]        MOVE.L     (A2),D0
00000055e      -                 MOVEQ      #$0000000A,D1
000000560      -                 JSR        _lrem
000000566      -                 MOVE.L     D0,-(A7)
000000568      -                 MOVE.L     (A2),D0
00000056a      -                 MOVEQ      #$0000000A,D1

STATUS: M68040--Running in monitor           Emulation trace halted
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Display  Modify  Load  Store  Copy  Find
```

As with any window, you can use the <UpArrow>, <DownArrow>, <PgUp>, and <PgDn> keys to scroll the information in the window.

Step 9. Run the demo program

- Select the **P**rocessor **G**o **A**ddress command.

The **P**rocessor **G**o **A**ddress command causes the emulator to run from a specified address.

For example, to start the emulator executing the demo program from the ENTRY address, select:

```
Processor Go Address
```

Enter "ENTRY" in the address field and press <Enter>.

The status line will show that the emulator is "Running user program".

Step 10. Trace demo program execution

When you start a trace measurement, the analyzer 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."

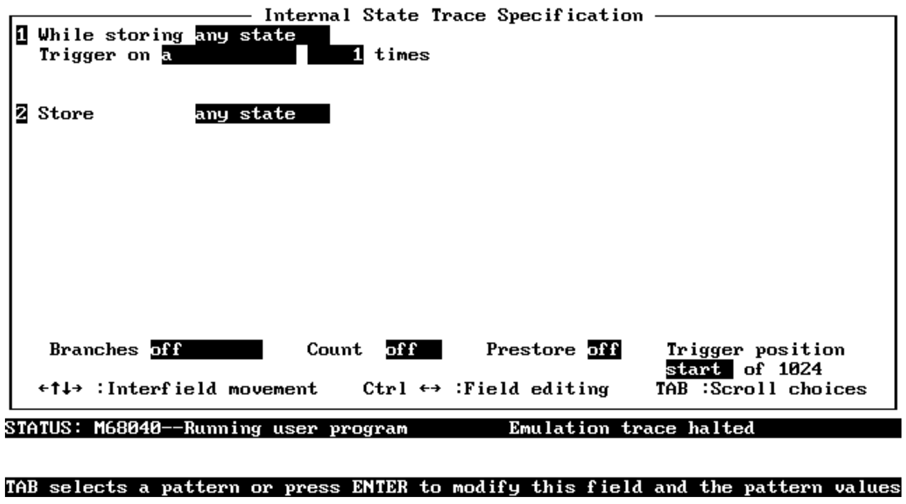
1 Select the **A**nalysis **T**race **M**odify command to modify the trace specification.

The **A**nalysis **T**race **M**odify command provides one screen (with one subscreen) from which you make a complete trace specification.

For example, to modify the trace specification so that the analyzer triggers on the address "main", select:

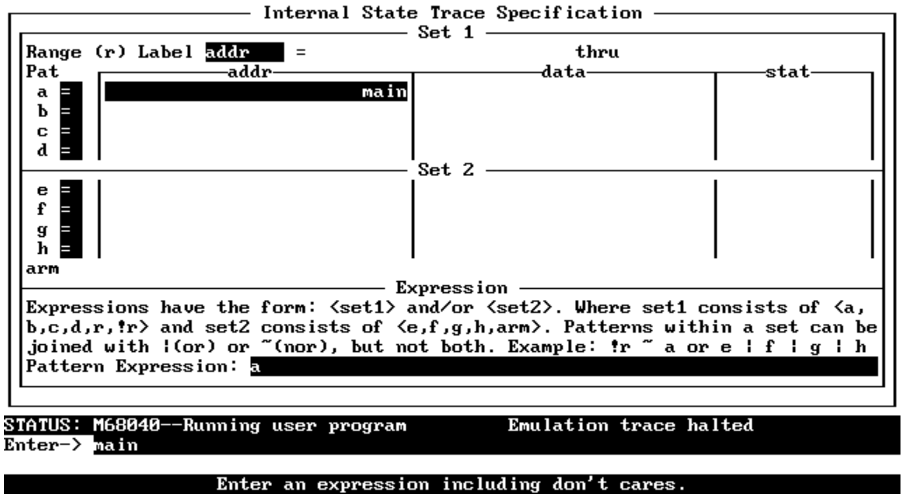
```
Analysis Trace Modify
```

Move the cursor to the "Trigger on" field of the first sequence term. Use the <Tab> key to select pattern "a".



```
                  Internal State Trace Specification
1 While storing any state
  Trigger on a                 1 times

2 Store          any state



  Branches off          Count off     Prestore off    Trigger position
                                                      start  of 1024
 ←↑↓→ :Interfield movement    Ctrl ↔ :Field editing   TAB :Scroll choices
STATUS: M68040--Running user program        Emulation trace halted

TAB selects a pattern or press ENTER to modify this field and the pattern values
```

Press <Enter>. You are now presented a screen in which you assign values to patterns. Move the cursor to the "addr" column of pattern "a". Then type in the symbolic value "main" beside "Enter ->".



```
─────────────────── Internal State Trace Specification ───────────────────
                                     Set 1
Range (r) Label addr     =                      thru
Pat              addr                     data                stat
 a =                         main
 b =
 c =
 d =
                                     Set 2
 e =
 f =
 g =
 h =
arm
                                  Expression
Expressions have the form: <set1> and/or <set2>. Where set1 consists of <a,
b,c,d,r,!r> and set2 consists of <e,f,g,h,arm>. Patterns within a set can be
joined with !(or) or ~(nor), but not both. Example: !r ~ a or e ! f ! g ! h
Pattern Expression: a

STATUS: M68040--Running user program        Emulation trace halted
Enter-> main

                 Enter an expression including don't cares.
```

Press the <Enter> key to see the symbol "main" appear in the addr column beside a=.

Press the <End> key and then press the <Enter> key to exit the patterns and expressions screen and save your assigned values.

Press the <End> key and then press the <Enter> key to exit the trace specification screen and save your changes.

2 Transfer emulator execution from running the user program to running the monitor. This will allow you to restart the user program after the analyzer has started its trace. More about the **P**rocessor **B**reak command later in this chapter. Enter the command:

```
Processor Break
```

3 Select the **A**nalysis **B**egin command to start the trace measurement.

You can start the trace by selecting:

```
Analysis Begin
```

Notice that the status line shows "Emulation trace running". In this case, this means the analyzer has not found the trigger state, but you will also see this status when the analyzer has found the trigger state but not captured enough states to fill trace memory.

4 Select the **P**rocessor **G**o **A**ddress command to run the demo program from its ENTRY address.

```
Processor Go Address
```

Enter "ENTRY" in the address field and press <Enter>.

Notice that the status line now shows "Emulation trace complete". This means the trigger state has been found and trace memory has been filled with states.

5 Select the **A**nalysis **D**isplay command to obtain a form that lets you select the desired trace display.

To obtain the trace display form, select:

```
Analysis Display
```

The "States available" shows the number of states available for display in the trace list.

The "Dequeuing" field lets you select "off" for a trace display that shows all bus cycles that were captured, or "on" for a more readable trace list that eliminates unused prefetch cycles and aligns opcodes and operands. Select "on".

You are given two fields in which to specify the states to display. Type the number of the first state you want to display in the "Start" field, and press <Enter>. Type in the number of the last state you want to display. For this trace list, select "-2" for Start, and "100" for End.

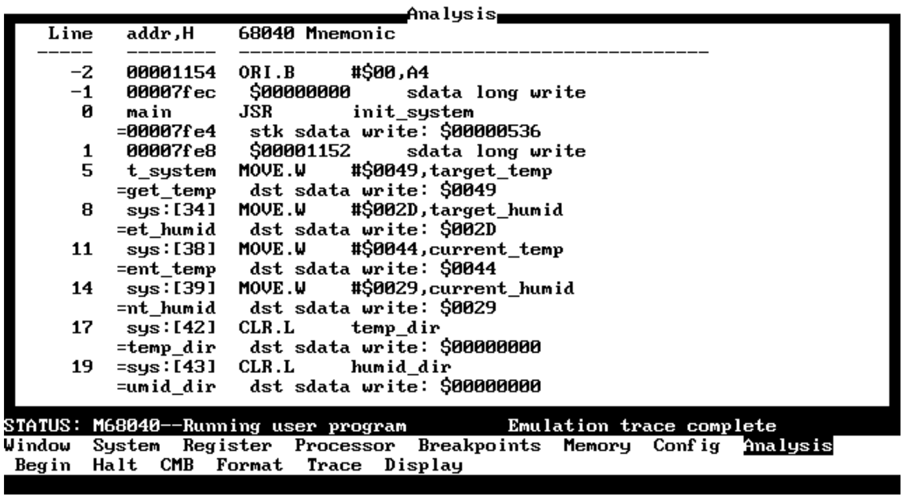
The "Cycles" field allows you to specify either display of "all" cycles, or only the cycles that contain program instructions ("instr only"). Select "all".

The "Address mode" field allows you to specify whether addresses, symbols, or both are shown in the address column of the trace. Select "both".

The "Start on" field lets you specify whether to start trace disassembly on the high word or low word of the long word at the address you specify. Instruction opcodes may be contained in the high word, low word, or both words of the long words captured by the analyzer. Trace disassembly must begin on an instruction opcode. For this disassembly, select "high word".

To display the trace list according to your specifications, press <End> and then <Enter>.

Press <CTRL>z to zoom the analysis display window. An example trace display is shown below. Press the <Home> key to see the start of the trace.



```
                                   Analysis
  Line   addr,H    68040 Mnemonic
  -----  --------  ------------------------------------------
    -2   00001154  ORI.B     #$00,A4
    -1   00007fec   $00000000     sdata long write
     0   main      JSR       init_system
        =00007fe4   stk sdata write: $00000536
     1   00007fe8   $00001152     sdata long write
     5   t_system  MOVE.W    #$0049,target_temp
        =get_temp   dst sdata write: $0049
     8    sys:[34] MOVE.W    #$002D,target_humid
        =et_humid   dst sdata write: $002D
    11    sys:[38] MOVE.W    #$0044,current_temp
        =ent_temp   dst sdata write: $0044
    14    sys:[39] MOVE.W    #$0029,current_humid
        =nt_humid   dst sdata write: $0029
    17    sys:[42] CLR.L     temp_dir
        =temp_dir   dst sdata write: $00000000
    19  =sys:[43]  CLR.L     humid_dir
        =umid_dir   dst sdata write: $00000000

STATUS: M68040--Running user program         Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Begin  Halt  CMB  Format  Trace  Display
```

The first column in the trace list contains the line number. The trigger state is always on line number 0.

The second column contains the address information associated with the trace states. Addresses in this column may be locations of instruction opcodes on fetch cycles, or they may be sources or destinations of operand cycles. Selecting "both" for the address disassembly mode causes both hexadecimal values and symbols to appear in this column.

See the equals signs "=" preceding some of the addresses. Addresses preceded by "=" are "equivalent" addresses. They were emitted by the inverse assembler to identify the low words in the long words captured by the analyzer.

The third column shows mnemonic information about each emulation bus cycle.

Step 11. Stop (break from) program execution

The **P**rocessor **B**reak command causes emulator execution to break from the user program and begin executing the monitor program.

The monitor state is implemented using the 68040 Background monitor Debug Mode (BDM). When the emulator is running the monitor program, it can access internal registers and target system resources.

When the emulator is running the user program, commands that require access to internal registers or target system resources will cause temporary breaks to the monitor (unless the emulator is restricted to real time runs).

When the emulator is running in the monitor, no bus cycles or processor instructions occur, except for accesses to target memory when requested.

- Select the **P**rocessor **B**reak command.

To break emulator execution from the demo program to the monitor program, select:

```
Processor Break
```

The status line shows that the emulator is "Running in monitor".

Step 12. Display processor registers

- Select the **R**egister **D**isplay command.

For example, to display the contents of the basic registers, select:

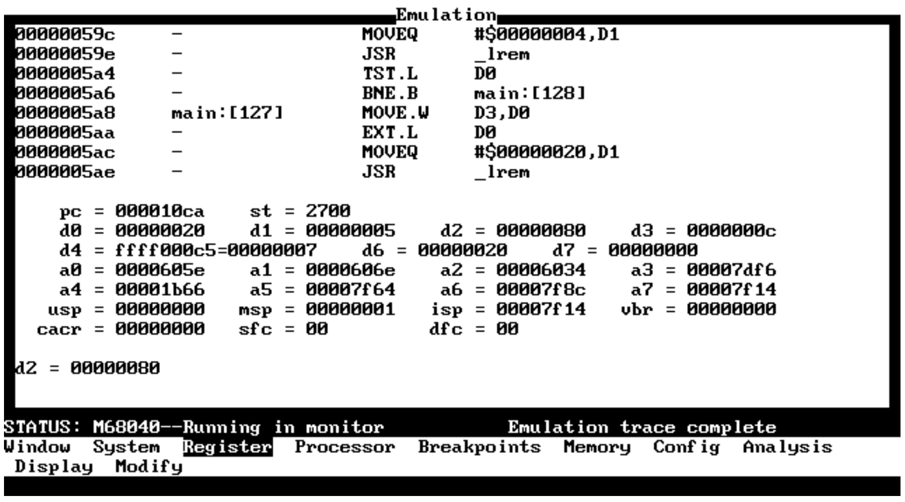
Register **D**isplay **B**asic

Press <CTRL>z to zoom the emulation window. To display the contents of register D2, alone, select:

Register **D**isplay **S**ingle

Use the <Tab> key to select "d2" and press <Enter>.

The register contents are displayed in the emulation window as shown in the following display.



```
                                    Emulation
00000059c     -                  MOVEQ     #$00000004,D1
00000059e     -                  JSR       _lrem
0000005a4     -                  TST.L     D0
0000005a6     -                  BNE.B     main:[128]
0000005a8     main:[127]         MOVE.W    D3,D0
0000005aa     -                  EXT.L     D0
0000005ac     -                  MOVEQ     #$00000020,D1
0000005ae     -                  JSR       _lrem

    pc = 000010ca    st = 2700
    d0 = 00000020    d1 = 00000005    d2 = 00000080    d3 = 0000000c
    d4 = ffff000c5=00000007    d6 = 00000020    d7 = 00000000
    a0 = 0000605e    a1 = 0000606e    a2 = 00006034    a3 = 00007df6
    a4 = 00001b66    a5 = 00007f64    a6 = 00007f8c    a7 = 00007f14
   usp = 00000000   msp = 00000001   isp = 00007f14   vbr = 00000000
  cacr = 00000000   sfc = 00         dfc = 00

d2 = 00000080

STATUS: M68040--Running in monitor           Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Display  Modify
```

Step 13. Step through program execution

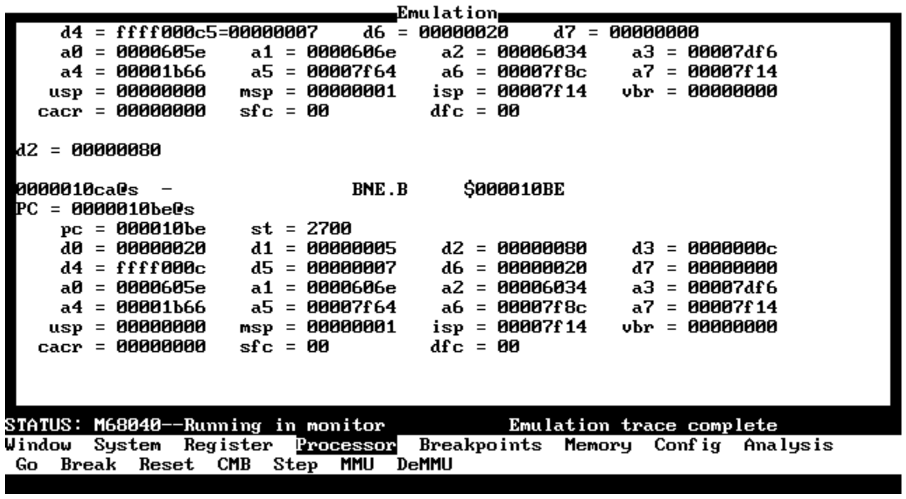
The emulator allows you to execute one instruction or a number of instructions with the step command.

- Use the **P**rocessor **S**tep command.

To step one instruction from the emulator's current program counter, select:

Processor **S**tep **P**c

Enter a step count of 1, and press <Enter>. The executed instruction, the program counter address, and the resulting register contents are shown.

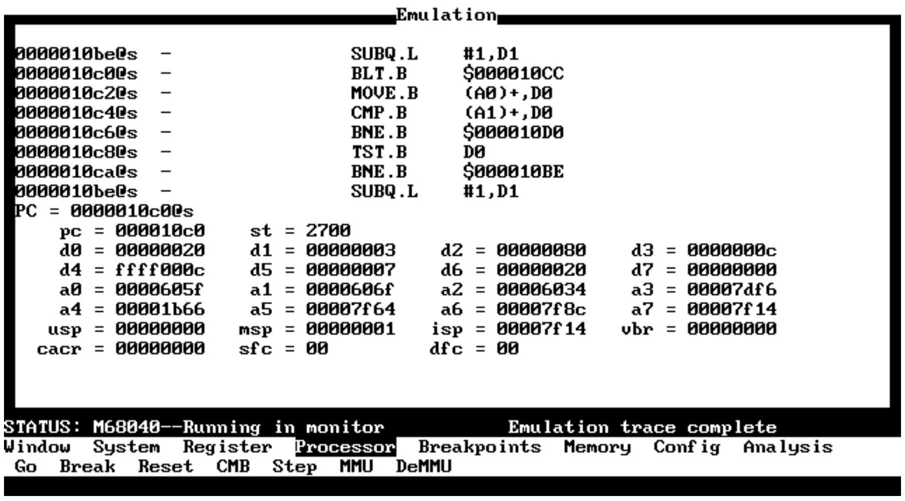


```
                                    Emulation
    d4 = ffff000c5=00000007    d6 = 00000020    d7 = 00000000
    a0 = 0000605e    a1 = 0000606e    a2 = 00006034    a3 = 00007df6
    a4 = 00001b66    a5 = 00007f64    a6 = 00007f8c    a7 = 00007f14
   usp = 00000000   msp = 00000001   isp = 00007f14   vbr = 00000000
  cacr = 00000000   sfc = 00          dfc = 00

d2 = 00000080

0000010ca@s  -                  BNE.B      $000010BE
PC = 0000010be@s
    pc = 000010be    st = 2700
    d0 = 00000020    d1 = 00000005    d2 = 00000080    d3 = 0000000c
    d4 = ffff000c    d5 = 00000007    d6 = 00000020    d7 = 00000000
    a0 = 0000605e    a1 = 0000606e    a2 = 00006034    a3 = 00007df6
    a4 = 00001b66    a5 = 00007f64    a6 = 00007f8c    a7 = 00007f14
   usp = 00000000   msp = 00000001   isp = 00007f14   vbr = 00000000
  cacr = 00000000   sfc = 00          dfc = 00

STATUS: M68040--Running in monitor          Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Go  Break  Reset  CMB  Step  MMU  DeMMU
```

To step eight instructions from the current program counter, select:

Processor **S**tep **P**c

The previous step count is displayed in the "# of instructions to single step" field. You can enter a number from 1 through 99 to specify the number of times to step. Type 8 into the field, and press <Enter>. The executed instructions and ending register contents are shown in the emulation display.



```
                                        Emulation
0000010be@s  -               SUBQ.L    #1,D1
0000010c0@s  -               BLT.B     $000010CC
0000010c2@s  -               MOVE.B    (A0)+,D0
0000010c4@s  -               CMP.B     (A1)+,D0
0000010c6@s  -               BNE.B     $000010D0
0000010c8@s  -               TST.B     D0
0000010ca@s  -               BNE.B     $000010BE
0000010be@s  -               SUBQ.L    #1,D1
PC = 0000010c0@s
    pc = 000010c0    st = 2700
    d0 = 00000020    d1 = 00000003    d2 = 00000080    d3 = 0000000c
    d4 = ffff000c    d5 = 00000007    d6 = 00000020    d7 = 00000000
    a0 = 0000605f    a1 = 0000606f    a2 = 00006034    a3 = 00007df6
    a4 = 00001b66    a5 = 00007f64    a6 = 00007f8c    a7 = 00007f14
   usp = 00000000   msp = 00000001   isp = 00007f14   vbr = 00000000
  cacr = 00000000   sfc = 00         dfc = 00

STATUS: M68040--Running in monitor           Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Go  Break  Reset  CMB  Step  MMU  DeMMU
```

Step 14. Reset the emulator

- Use the **P**rocessor **R**eset command.

To reset the emulator and hold the processor in the reset state, select:

```
Processor Reset Hold
```

The status line shows "Emulation reset".

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 the Getting Started chapter.

For more information, refer to the "Solving Problems" chapter and the "Emulator Error Messages" chapter later in this manual.

If the PC interface won’t start

When you start the PC Interface, the first thing it attempts to do is communicate with the emulator. If it is unable to do so, the program simply exits, returning you to the MS-DOS prompt.

☐ Make sure you have the proper cable to connect your PC to the emulator. See the *HP 64700 Card Cage Installation/Service Guide.*

☐ Make sure that all switches on the HP 64700 Card Cage are set correctly. For a standard RS-232 connection to a PC, the COMM switches on the back of the Card Cage should be all set to 0, with switch 13 set to 1. If you are using an RS-422 connection, you will need different switch settings. See the *HP 64700 Card Cage Installation/Service Guide.*

☐ Make sure that the emulator is connected to power and that the power switch is on.

☐ Make sure that the 64700tab configuration file (usually installed in \hp64700\tables) matches the communications port number, communications parameters and emulator description for the port you are using and the emulator. If you are unable to determine the source of the problem, contact your local HP Sales and Service Office for assistance.

If the emulator starts, but won't respond to commands

☐ Make sure that the data communications switch settings (or settings made with the **stty** command in the terminal interface) are correct for the terminal or host computer and cable you are using.

If the emulator seems to execute a command but doesn't echo what you typed, check the local echo switch setting or the echo setting in the **stty** command in the terminal interface.

If you need more information about power or datacomm connections, see the *HP 64700 Series Card Cage Installation/Service Guide.* If you are unable to determine the source of the problem, contact your local HP Sales and Service Office for assistance.

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.

- ☐ Check to see that the emulation configuration and memory map are correct. Refer to the chapter titled, "Configuring the Emulator."

- ☐ Make sure you are using the correct load procedure for the emulator communications configuration. Refer to the chapter titled "Using the Emulator" for examples of different configurations and the appropriate load procedures.

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 to see that the configuration and memory map are correct. See the chapter titled "Configuring the Emulator".

If the emulator won’t run the program

☐ Check to see that the configuration and memory map are correct. See the chapter titled "Configuring the Emulator".

☐ Make sure that the emulator is properly connected to the demo board. See the chapter titled "Installation and Service" in this manual.

If you can’t break to the monitor

☐ Run performance verification to test the emulation controller. See "To verify the performance of the emulator" in the chapter titled "Installation and Serivce" in this manual.

☐ Make sure your stack pointer is valid.

If the emulator won't reset

☐ Run performance verification to test the emulation controller. See "To verify the performance of the emulator" in the chapter titled "Installation and Serivce" in this manual.

Part 2

Using the Emulator

Making Measurements

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

In This Part

Chapter 3, “Using the PC Interface,” tells you how to use the PC Interface commands.

Chapter 4, “Using the Emulator,” shows you how to use the PC Interface commands to control the emulation processor and make simple emulation measurements.

Chapter 5, “Using the 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, “Configuring the Emulator,” explains how to use the PC Interface commands to allocate emulation resources such as memory and how to enable and disable certain emulator features.

Chapter 8, “Solving Problems,” explains some of the problems that you might encounter when you use the emulator, and how to solve them.

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

3

Using the PC Interface

How to control the emulator by using commands and forms

The PC Interface allows you to use the emulator and analyzer through a simple guided interface. You are prompted to select commands and fill in electronic "forms" to operate the emulation and analysis functions.

What are PC Interface forms?

In contrast to commands, which perform some action, the PC Interface uses a number of electronic forms to accept data entry for later use. A form is a screen used to enter multiple pieces of related information. For example, emulator configuration information is entered via a form. Forms almost always perform actions, and the data entered via a form always affects the behavior of certain subsequent commands.

All PC Interface forms are similar in that they have a number of *fields* where you type in information. In some cases, a field may only accept a small set of predetermined values (such as "yes" or "no"). For these fields, you can either type in the value, or use the **Tab** and **Shift-Tab** keys to cycle through the valid choices for the field.

When you are filling in a form, you can use arrow keys to move from field to field. Pressing **<Enter>** accepts the value in the field, and moves to the next field. **Cntl Left, Cntl Right**, **Ins(ert)**, **Del(ete)**, and **Backspace** allow you to back up within a single field and edit it. When you finish filling in a form, position the cursor on the last field of the form (usually in the lower right corner), and press **<Enter>**. You can move to the last field with the (up and down) arrow keys, or you can press the keyboard **<End>** key. If you need to cancel your changes, press **<Esc>** to exit the form unchanged.

Running the PC Interface and Composing Commands

Before you can use the PC Interface with your emulator, you must first install the emulator hardware and software, normally done only once, and then develop target system programs to run.

Installing the emulator hardware and software is described in the chapter titled "Installation and Service", and in the first steps of the chapter titled "Getting Started".

Developing programs is described in the “Programs” section of the chapter titled "Using the Emulator".

To use the PC Interface with your emulator

Caution Possible damage to emulator! When you use the emulator with a target system, always apply power to the emulator before turning on power to the target system. Otherwise, the emulator might be damaged.

1. Apply power to the emulator.
2. Apply power to your target system.
3. Start the PC Interface software (described later in this chapter).
4. Use the PC Interface to configure the emulator as needed for the target system and your programs. See the chapter titled, “Configuring the Emulator.”
5. Use the PC Interface commands to load, run and debug your programs as described in this chapter and the next two chapters in this manual.

To apply power

Caution

Possible damage to emulator! When you use the emulator with a target system, always apply power to the emulator before turning on power to the target system. Otherwise, the emulator might be damaged.

1 Make sure that the emulator probe is plugged into the demo board or your target system. (Do this with emulator power OFF and target system power OFF.)

2 Apply power to the emulator by moving the front panel power switch to the ON position.

3 Apply power to the target system.

You must apply power to the emulator before you start the PC Interface software. Otherwise, the PC Interface cannot communicate with the emulator, and won't start.

To start the PC Interface

1 If you wish to start the interface running in a specific directory, use the MS-DOS **cd** command to change to that directory before you start the PC Interface.

The PC Interface allows you to enter complete pathnames whenever a file name is required. However, if you have most of your files in one directory, you can save typing by changing to that directory before you start the PC Interface. Or, you can change the current directory from inside the PC Interface later.

2 Start the PC Interface by typing

```
pcm68040 <options> <logical name>
```

at the MS-DOS command prompt.

The **<options>** parameter can be left blank, or can be one or more of the following. Each option must be separated by at least one blank space from the next option.

Option	**Meaning**
/m	Tells the PC Interface that you're using a monochrome monitor.
/w <filename>	Loads a PC Interface configuration file designated by <filename>.
/c <filename>	Loads and executes a PC Interface command file designated by <filename>.
/?	Displays information about the PC Interface startup options.

The **<logical name>** parameter is one of the communications port names assigned in the file \hp64700\tables\64700tab. At installation, the 64700tab file assigns **emul_com1** to the COM1 port, and **emul_com2** to the COM2 port. You may wish to add to this (if you have more than two COM ports) or change the names to something more meaningful. For example, if you had both an MC68020 and an MC68040 emulator, you might assign **em020** to the COM1 port, and **em040** to the COM2 port. The 64700tab file can be changed with a text editor.

When you start the PC Interface, it initializes the emulator as specified in your last exit command (for example, **S**ystem **E**xit **L**ocked or **U**nlocked), and presents the initial PC Interface windows. If the startup is unsuccessful, you'll be returned to the MS-DOS prompt.

Examples

To start the MC68040 PC Interface on a system with a monochrome monitor and the emulator attached to the serial port COM1 (named emul_com1 in the hp64700tab file), type:

```
pcm68040 /m emul_com1
```

To start the MC68040 PC Interface with a color monitor, the emulator attached to the serial port named em040 and load a configuration file named config1.cfg, type:

```
pcm68040 /w config1.cfg em040
```

To compose commands in the command line

- Press the key corresponding to the first letter of the command option you want.

  or

  Use the right and left arrow keys to highlight a command option. Press the **<Enter>** key to select that option.

- Press the **<Esc>** key to back up in the command tree when you select the wrong option.

- Press **<Ctrl>R** to repeat the last command.

When a command or option is highlighted, the bottom line of the display shows the next level of options or a short message describing the current option.

Example

Press the keys “p,” “g,” and “p” to select the command **P**rocessor **G**o **P**c.

To view the emulator status

- To see the emulator status, look at the status line near the bottom of the screen. This line is always visible.

The emulator status line tells you whether the emulation processor is running in the user program, running in the monitor, or in the reset state. It also tells you if an analysis trace is in progress, complete, or was halted.

To exit the PC Interface

- To save the current configuration, exit, and then restore the current configuration the next time you enter the PC Interface, select **S**ystem **E**xit **L**ocked.

- To exit and start with the default configuration the next time you enter the PC Interface, select **S**ystem **E**xit **U**nlocked.

- To exit the PC Interface without saving the current configuration, select **S**ystem **E**xit **N**o_Save.

When you start the PC Interface, it looks in the current directory for the file pcm68040.cfg. If present, that configuration file is loaded. When you exit “locked,” the current configuration is written to that file. When you exit “unlocked” or “no save,” that file is left unchanged.

Using Windows

Most of the information reported to you by the PC Interface is displayed in one or more windows. For example, the output from a trace listing is displayed in one window. Another window is used to show you the results of a memory display . Yet another window might contain a listing of a text file which you requested the PC Interface to display for you. These windows allow multiple views of different information to be present on your screen at the same time.

Each window behaves like a miniature terminal screen. Since there are multiple windows displayed on the screen at one time, they tend to be fairly small (about 5 lines), but they can be scrolled forward and backward (within limits). Or, you can “zoom” a window so that it occupies most of the screen. You can even hide a window, to make room for another window to be displayed.

There are two types of windows in the PC Interface: system windows and user-defined windows.

System windows are used by the PC Interface to display the results of various Interface commands, such as analysis traces or the terminal interface. Although these windows can be hidden or resized, they are a permanent part of the Interface and cannot be deleted.

User-defined windows can be opened by the user to display ASCII files, such as source code for use in debugging. These windows can be hidden, resized, and deleted.

Many Interface commands will automatically access the correct system window in order to complete the requested task. However, commands which require access to user windows must always have that window explicitly defined.

This section explains how to manage PC Interface windows using the **W**indow commands.

To create a new user window

1. Select **W**indow **O**pen.

2. Now you see the window definition form. Type a window name of 12 characters or less. Press **<Enter>**.

3. Type a number in the range 1..20 (a row number) to position the top edge of the window, then press **<Enter>**. Or press **<Enter>** to accept the default position.

4. Type a number in the range 1..20 to position the bottom edge of the window, then press **<Enter>**. Or, press **<Enter>** to accept the default position.

5. Now the cursor is in the “Autoclear” field. Type **y** if you want information that is written to the window to overwrite existing data in the window. Type **n** if you want information written to the window to be appended to the existing data. Press **<Enter>**.

6. Type in a number in the range 20..1030 to specify the buffer size (number of lines) of the window. Press **<Enter>**.

7. Type a number in the range 0..79 (a column number) to position the left edge of the window, then press **<Enter>**. Or, press **<Enter>** to accept the default position.

8. Type a number in the range 0..79 to position the right edge of the window, then press **<Enter>**. Or, press **<Enter>** to accept the default position.

9. Now the cursor is in the “Display” field. Type **y** if you want the window to be displayed when you finish this procedure. Type **n** if you want the window to be hidden.

10. To save the changes you have made to the form and create the new window, press **<End> <Enter>**. Or, press **<Esc>** to discard your changes and exit the window definition form.

A user-defined window provides a unique window into which you may load ASCII files, such as source code files or other information that will help you with your system debugging problems.

Since the window definition screen is a form, you may use the arrow keys to move to only the fields you want to change. Also, as with all forms, you can use the **End** key to move to the last field on the form. Note that although the buffer can contain many more lines than the on-screen portion, each 20 lines in the window buffer consumes up to 2 Kbytes of system memory in your PC.

You can display hidden windows with the **W**indow **U**tility **V**iew command, as described in “To view a window,” later in this chapter.

To delete a user window

1 Select **W**indow **D**elete.

2 Use **Tab** or **Shift-Tab** to select the name of the user-defined window you wish to delete. Or, type in the desired window’s name. Press **<Enter>**.

3 You are asked to confirm the deletion. Type **y** if you want to delete the selected window. Type **n** to abort the deletion.

4 Press **<Enter>** to delete the window, or press **<Esc>** to abort the command and return to the PC Interface.

When you are finished with a user-defined window, you may wish to delete it. Deleting a user-defined window removes it from the PC Interface display and frees the system memory associated with its corresponding window buffer.

To make a window the active window

- Use **<Ctrl-A>** to page through windows, sequentially activating them.

or

1. Select **W**indow **A**ctivate.
2. Use **Tab** or **Shift-Tab** to select the name of the window you want to activate. Or, type in the desired window’s name. (The currently active window is the default.)
3. Press **<Enter>** to confirm your selection and activate the window. Press **<Esc>** to abort the window activation and return to the PC Interface screen.

The active window is the one highlighted by a bold border on the PC Interface screen. This is the one that is acted on by PC Interface window commands, such as **W**indow **Z**oom, or by the cursor movement keys.

You can’t activate a hidden window. You must first make it visible with the **W**indow **U**tility **V**iew command.

To hide a window

1. Select **W**indow **U**tility **H**ide.
2. Use **Tab** or **Shift-Tab** to select the name of the window you want to hide. Or, type in the desired window’s name. (The currently active window is shown initially, making it the default choice.)

3 Press **<Enter>** to confirm your selection and hide the window. Press **<Esc>** to abort the process and return to the PC Interface screen.

You can hide a window to temporarily remove it from the PC Interface screen. This can help clarify the display. The window definition still exists, and data is written to the window as if it were still displayed.

To view a window

1 Select **W**indow **U**tility **V**iew.

2 Use **Tab** or **Shift-Tab** to select the name of the window you want to view. Only windows that are currently hidden are listed. Or, type in the desired window's name.

3 Press **<Enter>** to confirm your selection and view the window. Press **<Esc>** to abort the process and return to the PC Interface screen.

You can't see the contents of a hidden window, though the PC Interface writes data to hidden windows. To reveal a hidden window and its contents, you use the **W**indow **U**tility **V**iew command.

You also need to use this command if a window is hidden and you want to activate it.

To erase the contents of a window

- Use **<Ctrl-E>** to erase the active window.

or

1 Select **W**indow **E**rase.

2 Use **Tab** or **Shift-Tab** to select the name of the window you want to erase. Or, type in the desired window’s name. (The currently active window is the default.)

3 Press **<Enter>** to confirm the name selection.

4 You are asked to confirm the erasure. Type **y** if you want to erase the contents of the selected window. Type **n** to abort the erasure.

5 Press **<Enter>** to erase the window contents, or press **<Esc>** to abort the erase command and return to the PC Interface.

Sometimes you may want to erase the contents of a window to remove incorrect measurement results, or to remove a file that you loaded and no longer need. You use the **W**indow **E**rase command to do this. You can also use the key combination **<Ctrl-E>** to erase the currently active window, but you are not prompted to confirm the erasure.

To load a file into a window

1 Select **W**indow **L**oad.

2 Use **Tab** or **Shift-Tab** to select the name of the window into which you want to load (display) a file. Or, type in the name of the window to use. (The currently active window is the default.) Press **<Enter>**.

3 Type in the name of a file to load into the window. You should load only ASCII text files. The results of loading non-text file types are unpredictable. You can include a drive specifier, directory path, and file extension if desired.

4 Press **<Enter>** to load the file contents into the selected window. Press **<Esc>** to abort the load command and return to the main PC Interface screen.

You can load a file into a window to view the file’s contents. This is handy for viewing source code files while debugging. Or, you might load files that describe system specifications while you are debugging that part of the system.

To store the contents of a window to a file

1 Select **W**indow **S**tore.

2 Use **Tab** or **Shift-Tab** to select the name of the window whose contents you want to store. Or, type in the name of the window whose contents will be stored. (The currently active window is the default.)

3 Press **<Enter>** to confirm the name selection.

4 Type in the number of the first line in the range of lines you want to store.

or

Position the cursor in the window on the first line in the range of lines you want to store. Use the up and down arrow keys to do this. The line number will be reflected in the "From line" field of the form.

Press **<Enter>** to move to the next field.

5 Type in the number of the last line in the range of lines you want to store.

or

Position the cursor in the window on the last line in the range of lines you want to store. Use the up and down arrow keys to do this. The line number will be reflected in the "Thru line" field of the form.

Press **<Enter>** to move to the next field.

6 Type in the name of a file into which the window contents should be stored. You can include a drive specifier, directory path, and file extension if desired.

7 Press **<Enter>** to store the window contents into the selected file. Press **<Esc>** to abort and return to the main PC Interface screen.

You may want to store the results of a measurement for later use. The **W**indow **S**tore command allows you to do this. By specifying a range of lines to be stored, you can limit the information stored to that which interests you.

You can't store an empty window to a file. If you specify the name of an existing file, the window contents are appended to the contents of the file.

Example

To print lines 20 through 30 of the Symbols window on a system printer attached to your PC, enter:

Window **S**tore **Symbols <Enter> 20 <Enter> 30 <Enter> prn <Enter>**

To search a window for a string

1 Select **W**indow **U**tility **S**earch.

2 Use **Tab** or **Shift-Tab** to select the name of the window whose contents you want to search. Or, type in the desired window's name. (The currently active window is the default.) Press **<Enter>**.

3 Type in the number of the first line in the range of lines you want to search.

or

Position the cursor in the window on the first line in the range of lines you want to search. Use the up and down arrow keys to do this. The line number will be reflected in the "From line" field of the form.

Press **<Enter>** to move to the next field.

4 Type in the number of the last line in the range of lines you want to search.

or

Position the cursor in the window on the last line in the range of lines you want to search. Use the up and down arrow keys to do this. The line number will be reflected in the "Thru line" field of the form.

Press **<Enter>** to move to the next field.

5 Type in the string that you want to find in the selected window.

6 Press **<Enter>** to search the window contents for the selected string. Press **<Esc>** to abort the command and return to the main PC Interface screen.

If the window buffer is large and the buffer is nearly full, you may find it tedious to page through the window display while searching for a particular measurement value or source line. Instead, you can use the **W**indow **U**tility **S**earch command to find the data.

When the PC Interface finds the string in the window, it places the cursor immediately after the string with the string at the top of the window.

To zoom a window

- Use **<Ctrl-Z>** to expand and condense the currently active window.

or

1 Select **W**indow **Z**oom.

2 Use **Tab** or **Shift-Tab** to select the name of the window that you want to zoom. Or, type in the desired window's name. (The currently active window is the default.)

3 Press **<Enter>** to confirm the selection and zoom the window to full screen size. Press **<Esc>** to abort the window zoom.

Often you'll want to see more information than can be displayed in the default window size, but you don't want to change the window parameters permanently. You can temporarily expand any window to the full PC Interface screen size. Other windows are obscured under the zoomed window. Only one window may be zoomed at a time.

If you perform the zoom operation on a window that already occupies the full PC Interface screen, it will be returned to the size defined by the window parameters.

To change window parameters

1 Select **W**indow **U**tility **P**arameters.

2 Use **Tab** or **Shift-Tab** to select the name of the window that you want to modify. Or, type in the desired window's name. (The currently active window is the default.) Press **<Enter>**.

3 Type a number in the range 0..20 to position the top edge of the window, then press **<Enter>**. (Or, press **<Enter>** to accept the default position.)

4 Type a number in the range 0..20 to position the bottom edge of the window, then press **<Enter>**. (Or, press **<Enter>** to accept the default position.)

5 Now the cursor is in the "Autoclear" field. Type **y** if you want information that is written to the window to overwrite existing data in the window. Type **n** if you want information written to the window to be appended to the existing data. Press **<Enter>**.

6 Type in a number in the range 20..1030 to specify the buffer size of the window. (Note that each 20 lines in the window buffer consumes about 2 Kbytes of system memory.) Press **<Enter>**.

7 Type a number in the range 0..79 to position the left edge of the window, then press **<Enter>**. (Or, press **<Enter>** to accept the default position.)

8 Type a number in the range 0..79 to position the right edge of the window, then press **<Enter>**. (Or, press **<Enter>** to accept the default position.)

9 Now the cursor is in the "Scroll" field. Type **y** if you want the PC Interface to write data to the window one line at a time as it becomes available. (Usually more useful for large trace listings or displaying large source files.)

Type **n** if you want the PC Interface to wait until it has all requested data available before it writes the data to the window. (The second method is faster overall, but you may notice an initial delay while the PC interface gathers the information.)

10 To save the changes, press **<End> <Enter>**. Press **<Esc>** to discard your changes and exit the window parameters form.

You may find that the default window definitions don't meet your needs. You can use the **W**indow **U**tility **P**arameters form to change the window definition to your tastes. Also, this form is the only way that you can change the way in which data is written to the window (the "Scroll" parameter).

Example

To reposition and resize the Symbols window, enter the following command:

```
Window Utility Parameters Symbols <Enter>
0 <Enter>
12 <Enter>
y <Enter>
48 <Enter>
30 <Enter>
79 <Enter>
n <Enter>
```

To change window colors

1 Select **W**indow **U**tility **C**olor.

2 Use **Tab** and ***Shift-Tab*** to select color if you have a display card and monitor that are capable of displaying color. Select mono if either your display card or monitor is restricted to monochrome.

3 Use **Tab** or **Shift-Tab** to select the foreground color from the following choices:

white, black, blue, green, cyan, red, magenta

Press **<Enter>**.

4 Use **Tab** or **Shift-Tab** to select the background color from the above choices.

5 Press **<Enter>** to save your changes, or press **<Esc>** to abort the command and leave the colors unchanged.

Changing colors is useful only if you have a color monitor. Although the monochrome monitor choice will allow you to change color selections, the display will always have black background color and white foreground color.

To move around in a window

- To move the cursor up in the window, use the up arrow key.
- To move the cursor down in the window, use the down arrow key.
- To move to the next page in the window buffer, use the **Pg Dn** key.
- To move to the previous page in the window buffer, use the **Pg Up** key.
- To move the cursor to the right in the window, use **Ctrl-right arrow**. If more than 78 columns right are present, moving the cursor right from right-most column will cause window to scroll horizontally.
- To move the cursor to the left in the window, use **Ctrl-left arrow**.
- To move to the beginning of the window buffer, press the **Home** key.
- To move to the end of the window buffer, press the **End** key.

To move around in a window, the window must be active. You use the **W**indow **A**ctivate command (described earlier in this chapter) to make a window active.

Defining and Using Function Key Macros

Function key macros allow you to assign keystroke sequences to a function key or function key combination. Thus, you can reduce typing and simplify repeated measurements by assigning your most frequent command sequences to a function key macro.

There are forty possible function key combinations that may be assigned to macros.

- Function keys **F1-F10**
- Function keys **F1-F10** combined with **Shift**
- Function keys **F1-F10** combined with **Alt**
- Function keys **F1-F10** combined with **Ctrl**

The default PC Interface configuration predefines three macros:

- F1 is defined as **P**rocessor **S**tep **P**c **1**.
- F2 is defined as **<Ctrl>**\ (the system terminal abort sequence).
- F10 is defined as **S**ystem **E**xit **L**ocked.

You can redefine these keys to any sequence that you want.

Nesting and Chaining Macros

You can nest macros to perform complex measurement sequences. For example, you might have the following sequence assigned to **F3**:

```
key_seq1<ShftF5>key_seq2
```

<Shift><F5> may then have another sequence.

Nesting of macros is supported. For example, **<Shift><F5>** may be the following key sequence:

```
key_seq3<f4>key_seq4
```

Nesting is limited to 16 levels. Direct or indirect recursion of macros is not permitted, except as a chain. For example, suppose you have the following assigned to F3:

```
key_seq1<f3>key_seq2
```

This isn't a valid macro. But, you can do the following:

```
key_seq1 key_seq2<f3>
```

Keystroke Representations

When you select a keystroke for activation of the macro, you can use **<Tab>** and **<Shift><Tab>** to scroll through the choices, or you can directly type the desired key combination. This works for all key macro definitions. You can create configuration files that define key macros by using a text editor, as long as you follow the rules for representing keystrokes.

- All keystroke sequences that are not part of the standard printable ASCII character set must be enclosed in brackets. This includes characters in the range 0..31 decimal and 127..255 decimal. For example, function key 3 is represented as **<F3>**.
- The control key (**<Ctrl>**) is represented using the circumflex symbol (^) when the character that follows it is part of the ASCII character set, and is represented by the string "Ctrl" when the following character is not part of the ASCII character set. For example, **<Ctrl>M** (**<Enter>**) is shown as **<^M>**; **<Ctrl>F5** is shown as **<CtrlF5>**.

When you use **C**onfig **K**ey_macro to define a macro, it's easier to simply use a particular key rather than enter a string representing that key. But, when you're using a text editor to create a configuration file, pressing that key may have other consequences. For example, if you press **<Enter>** in your editor, a new line is started. So, you type "<^M>", which the PC Interface uses to represent the **<Enter>** key.

Organizing the Macros

You may want to arrange your macro definitions to help you remember their functions. For example, you might assign all unshifted function keys to System functions, all Alt function keys to Analysis functions, and so on. Or, if you are using the emulator in a production testing environment, you might arrange the macros so that the key number corresponds to steps in a sequence.

For long keystroke sequences, it may be better to have the macro call a command file. This is particularly true when you are configuring the emulator for a measurement. For example, suppose you want to change the configuration, map memory, and load an absolute file. Simplify the process by building command and configuration files that will be loaded with one macro.

To create a function key macro

1 Select **C**onfig **K**ey_macro.

2 Use the **<Tab>** and **<Shift><Tab>** keys to select the function key combination to which you want a macro assigned. Press **<Enter>**.

3 Use the **<Tab>** and **<Shift><Tab>** keys to select the keystroke that terminates the function key macro definition (default is **<Esc>**). Press **<Enter>**.

4 Enter the keystrokes that will be assigned to the function key.

5 Press the **<Esc>** key (or other key defined as the termination key in step 3) to end the macro definition.

You create a function key macro by choosing the key sequence that will run the macro, then defining the sequence of operations to be performed in the macro.

You can create a function key macro outside the PC Interface by using a text editor to create a configuration file. The key macro definitions must be preceded and followed by the $KEYMAC separator. Each macro definition begins with the key combination that represents the macro. A colon separates the macro assignment from the definition. The definition itself appears as the sequence of keystrokes in the macro. The following shows the default function key macros defined by the PC Interface, as represented in a configuration file:

```
$KEYMAC
<F1>:psp1<^M>
<F2>:<^\>
<F10>:sel
$KEYMAC
```

Examples

Build a macro to end modification of a PC Interface form and save its contents:

```
<F5>:<End><^M>
```

Build a macro to create a user-defined window and load a source file into that window. This example uses the <F5> macro as part of its sequence.

```
<F6>:wosource1<^M><^M><^M><^M>1000<^M>40<F5>wl<^M>
demo.asm<^M>
```

Build a set of macros to implement a “trace continuous” feature:

```
<ShftF1>:abi

<ShftF2>:adi<F5>

<ShftF3>:<ShftF1><ShftF2><ShftF3>
```

If you press **<ShftF3>** after defining this macro, the analyzer repeats trace measurement and display. If there are no states to display, the PC Interface will beep. The analyzer displays the first 16 states by default. For your measurements, you may want to define other macros to set up the trace specification, and redefine the macro assigned to **<Shift><F2>** to select the states for display.

You can improve the execution of the last macro by selecting **W**indow **U**tilities **P**arameters, then set “Autoclear” to **y** and **scroll** to **n**. When you run the macro, the entire analyzer display will be updated simultaneously.

To execute a function key macro

- To execute a function key macro for a particular command sequence, press the key that is assigned to that sequence.
- To cancel repetition of nested or chained macros, press the **<Esc>** key.

To run a function key macro, you type the key sequence for the macro that you want. For example, if System Symbols Global Display is assigned to **<Alt><F3>**, press and hold the **<Alt>** key, then press **<F3>**.

To edit a function key macro

1 Select **C**onfig **K**ey_macro.

2 Use the **<Tab>** and **<Shift><Tab>** keys to select the function key macro that you want to edit. Press **<Enter>**.

3 Use the **<Tab>** and **<Shift><Tab>** keys to select the keystroke that terminates the function key macro definition (other than **<Esc>**). Press **<Enter>**.

4 Edit the macro definition.

To move the cursor through the macro definition, use **<Ctrl> left arrow** and **<Ctrl> right arrow**.

To delete a keystroke character, position the cursor at that character, and then press **<Delete>**.

To type a keystroke character, press the key combination that creates that character (for example, **<Ctrl>m** to enter the character <^**M**>).

5 Press the macro definition termination key, assigned in step 3, to save the changes.

You can edit a keystroke macro after you have created it. You might do this if you find a more efficient way to make the measurement, or if you want to change macro parameters. The PC Interface allows intelligent editing of the macros. For example, if you have the character sequence **<^M>** in your macro, and position the cursor anywhere in that sequence, pressing **<Delete>** will remove the entire sequence.

To delete a function key macro

1 Select **C**onfig **K**ey_macro.

2 Use the **<Tab>** and **<Shift><Tab>** keys to select the function key macro that you want to edit. Press **<Enter>**.

3 Use the **<Tab>** and **<Shift><Tab>** keys to select the keystroke that terminates the function key macro definition (other than **<Esc>**). Press **<Enter>**.

4 Use the **<Delete>** key to remove all keystrokes from the macro definition.

5 Press the macro definition termination key, assigned in step 3, to save the changes and clear the macro definition.

You may want to delete a key macro sequence to save space in configuration files. You can delete the macro definition using the above procedure, then save the configuration file. Or, you can remove the macro definition text from the configuration file by using a text editor.

To save and restore function key macros

- To save the function key macro definitions, create a configuration file by selecting **C**onfig **S**tore and specifying a file name.

- To load a set of function key macro definitions, load a configuration file by selecting **C**onfig **L**oad and specifying the name of a configuration file that contains the macro definitions you want to load.

The function key macro definitions are stored in PC Interface configuration files. You can create an explicit configuration file that contains your macro definitions and the other configuration items. Also, if you exit the PC Interface with **S**ystem **E**xit **L**ocked, the macro definitions and other configuration information will be stored in a default configuration file. This file will be loaded when you reenter the PC Interface.

Building Command Files

A command file is an ASCII file containing PC Interface commands. You can create command files from within the interface by logging commands to a command file as you execute the commands. Or, you can create command files outside the interface with an ASCII text editor. You can send a command file to the PC Interface and have it execute the commands found there as if you typed them directly into the interface command line.

With a single command file, you can implement a complete test procedure. For example, you could start the interface and execute your command file. The command file could load a configuration, load an absolute file, modify registers or memory, set up a trace specification, start the program, capture the trace, and save the trace listing to a file. (The ability to capture information from the emulator may be limited, and depends on the host computer configuration.)

You can build a command file by creating a list of commands with an ASCII editor, or by logging commands to a file during a work session. Because the command file is an ASCII text file, you may use an ASCII editor to add, modify, or remove commands.

You can put most PC Interface commands into a command file. The only things that you cannot do in a command file are:

- Define function key macros.
- Set up the analyzer trace specification.
- Change the emulator configuration.

These things must be done in a configuration file. See the section "Using Configuration Files."

As with any source file, comments in command files help to explain the operation of the command file and can also contain creation and modification information for the command file. You can put comments in command files by using a text editor; a "#" character anywhere in a line means that the rest of the text on the line is a comment.

Command files may be nested up to eight levels. (Nesting means that one command file calls another.)

To create a command file using an editor

- Use an ASCII text editor to create and save a file containing abbreviated PC Interface commands and parameters.

You can create a command file outside the PC Interface by using a text editor. The following table shows how things are represented in command files:

Token	Action	Example
Single character	Selects the PC Interface command or option that starts with that character	abi
@<VALUE>	<VALUE> is a parameter for a data form	@20
<	Resets to top of command file	<
!	Wait for one second	!
#	Starts comment text	#This is a comment.

Example

Create a file named **cmdfile.cmd** that contains the following text:

```
#open a new window
wo
#window name
@WIN1
#top row number
@0
#bottom row number
@5
#set autoclear on
@Y
#buffer size
@20
#left edge
@0
#right edge
@15
#display window when done
@Y
```

To create a command file by logging commands

- To log both commands and the command results to a file, select **S**ystem **L**og **B**oth **E**nable, type in a filename, and press **<Enter>**.

- To log only commands to a file, select **S**ystem **L**og **I**nput **E**nable, type in a filename, and press **<Enter>**.

- To log only command results to a file, select **S**ystem **L**og **O**utput **E**nable, type in a filename, and press **<Enter>**.

Command logging makes it easy to create a command file. You enable logging and work through the sequence of steps needed for your measurement. When you're finished, disable command logging and edit the file if necessary. You then have a file ready for use as a command file.

You can specify a pathname with the log file if you want the file in a directory other than the current directory. If you specify the name of an existing file, the existing file is automatically overwritten.

To use a command file

1. Select **S**ystem **C**ommand.

2. Type in the name of the file that contains the commands that you want to execute, and press **<Enter>**.

You can also start a command file automatically by specifying the **/c <filename>** option when you start the PC Interface. See "To start the PC Interface" in this chapter.

You can stop command file execution by pressing the **<Esc>** key.

To add delays during command execution

- To have the PC Interface wait for any keystroke before executing the next command, select **S**ystem **W**ait **K**ey.

- To have the PC Interface wait for the current measurement to finish before executing the next command, select **S**ystem **W**ait **M**easurement.

- To have the PC Interface wait for a definite period of time to elapse before executing the next command, select **S**ystem **W**ait **T**ime, type in a value in seconds, then press **<Enter>**.

Command delays can help make a command file more useful. Suppose that you have a command file that makes several measurements, but before each measurement, you must reset the target system. You can use the **S**ystem **W**ait command to delay the next command until you've had time to reset the target and press a key to continue the command file.

Using Configuration Files

The PC Interface uses configuration files to save and restore environment, emulator, and analyzer settings. Refer to Chapter 7, "Configuring the Emulator," and Chapter 5, "Using the Analyzer," for more information on configuring the emulator and analyzer.

The configuration file is an ASCII file having several different sections. These sections begin and end with a section name; between the names are the current configuration settings for that section.

$SYSWIN	Contains settings for the PC Interface system-defined windows.
$USERWIN	Contains settings for user-defined windows (if any).
$MISC	Contains color settings for display monitors.
$KEYMAC	Contains function key macro definitions.
$SYMDB	Defines the current symbol database.
$EMUL	Contains the current emulator configuration as a series of Terminal Interface configuration commands.
$TIMTRIG	Contains the trigger specification for the external timing analyzer, if present.
$STLABINT	Lists the analyzer trace format and label information.
$STPATINT	Lists pattern and range specifications for the emulation-bus analyzer.
$STSEQINT	Lists sequencer term definitions for the emulation-bus analyzer.

You can modify a configuration file using an ASCII text editor, and then load it into the PC Interface. However, the simplest and safest way to change a configuration file is to load an existing configuration, modify it using the PC Interface commands, and then save it to a new file.

To load a configuration file

1 Select **C**onfig **L**oad.

2 Type in the name of a file containing the configuration that you want to load, and press **<Enter>**.

You can also use the **/w <filename>** option to load a configuration file when you start the PC Interface. See “To start the PC Interface” in this chapter.

To store a configuration file

1 Select **C**onfig **S**tore.

2 Type in the name of a file under which you want to save the emulator configuration, and press **<Enter>**.

You can save the PC Interface configuration in a file for later use. This is especially useful if you have several sets of emulator configurations or analyzer trigger specifications that you use for different measurements. You can set up each configuration and save it in a file, then load each configuration when you’re ready to repeat that measurement.

The emulator configuration is also saved in a file when you select **S**ystem **E**xit **L**ocked. The filename is the same as that used to start the program (**pcm68040**) with the extension .CFG.

Accessing the Terminal Interface

You can control the emulator with a set of primitive commands resident in the emulator's firmware. This command set is called the Terminal Interface, because you only need an ordinary terminal to use these commands. The Terminal Interface is described in the *MC68040/EC040/LC040 Emulator/Analyzer Terminal Interface User's Guide*.

For most operations, you do not need to use the Terminal Interface commands because the PC Interface translates commands you give it into an appropriate sequence of Terminal Interface commands automatically. However, there may be times when you want to use the Terminal Interface commands directly. To do this, you can cause the PC Interface to activate the "Terminal" window. While the Terminal window is active, the PC Interface emulates an ordinary terminal. Anything you type while this window is active is sent to the emulator, which treats it as a Terminal Interface command. Any output produced by the emulator as a result is displayed in the same window.

This section explains how to access the Terminal Interface from within the PC Interface.

To access the Terminal Interface

- To enter the Terminal Interface, select **S**ystem **T**erminal.
- To leave the Terminal Interface, press **Ctrl-\**.

The Terminal Interface allows access to the low-level commands of the emulator. You need to use caution when using the Terminal Interface commands because the PC Interface operation may be disturbed by changes to the emulator configuration using the Terminal Interface. This is because you can change configuration parameters in the emulator, but the PC Interface will not be aware of the changes. In general, you should not use any commands that change the emulator's configuration or data communications parameters, nor any of the commands listed in the following table.

Commands	Reason 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 exit unlocked.

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

Examples

To access the Terminal Interface and display memory locations 0 through 20 in long word format, enter:

System **T**erminal
m -dl 0..20
Ctrl-\

To access the Terminal Interface and check the emulator status, and then the trace configuration, enter:

System **T**erminal
es
tcf
Ctrl-\

To get help on Terminal Interface commands

1 Select **S**ystem **T**erminal.

2 Enter: **help <cmd_name>**

where **<cmd_name>** is the Terminal Interface command for which you want help.

3 Type **Ctrl-\** to exit the Terminal Interface.

You can access the emulator's low-level Terminal Interface using the **S**ystem **T**erminal command. 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

To get help on the Terminal Interface **cf** command, enter:

```
System Terminal
help cf
Ctrl-\
```

To get help on all Terminal Interface command groups, enter:

```
System Terminal
help *
Ctrl-\
```

Accessing the Operating System

The PC Interface includes commands that allow you to perform simple MS-DOS operating system functions without leaving the PC Interface. You can enter a single command, and then return immediately to the PC Interface, or enter a series of commands at the MS-DOS prompt.

When you use the operating system access capability, you should not run any programs that modify the communications ports (COM1-COM4), such as terminal emulation programs. Also, you should not run any programs that consume system memory and do not release it. Many network management programs and TSR (terminate-and-stay-resident) programs are in this category.

Also, there is a limited amount of memory available when you use the system commands from the PC Interface. You cannot run programs that require large amounts of system memory, such as compilers or complex text editors.

This section explains how to use the system access commands.

To enter a single MS-DOS command

1 Select **S**ystem **M**S-DOS **C**ommand.

2 Type in the MS-DOS command you want to execute (include any command parameters such as filenames or command options).

3 Press **<Enter>** to execute the command. Press **<Esc>** to discard the command and return to the PC Interface.

4 When the command has finished executing, press any key to return to the PC Interface.

Examples

To display all files in the current directory, enter:

`System MS-DOS Command dir <Enter>`

To format a floppy in the A: drive, enter:

`System MS-DOS Command format a: <Enter>`

To enter multiple MS-DOS commands

1 Select **S**ystem **M**S-DOS **F**ork to invoke an MS-DOS command shell.

2 An MS-DOS prompt will appear. Type in the desired series of MS-DOS commands.

3 When you are ready to return to the PC Interface, type **exit** at the MS-DOS prompt, and then press **<Enter>**.

4

Using the Emulator

How to control the processor and view system resources

The emulator has many commands and features that allow you to control execution of your program, such as single stepping and setting breakpoints. You can also display or modify memory locations and registers.

Preparing the Emulator for Use

In order to use the emulator, it must first be properly installed and power must be applied to both the emulator and the target system (See “To apply power” in the chapter titled, “Using the PC Interface.” The emulator may then need to be configured to meet your target system’s design, see below.

To configure the emulator

- Set up the emulator for use by configuring it as described in the chapter titled, "Configuring the Emulator".

Before you can run or test your programs with the emulator, you need to specify a number of configuration items that adapt the emulator to specific target system designs and program requirements. You should check the configuration and modify it for your needs before using the emulator. Otherwise, some emulator functions may not operate correctly.

Loading and Storing Programs

The PC Interface provides commands that allow you to move files into emulation or target memory from the PC through the serial ports of the HP 64700 Card Cage. Or, you can 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 reload it later for further testing.)

The PC interface can load HP64000 and IEEE-695 absolute files, in addition to the following hexadecimal formats:

- Intel hexadecimal.
- Tektronix hexadecimal.
- Motorola S-records.

Two file format readers come with the PC Interface. The readers convert HP64000 or IEEE-695 files into two files that are usable with your emulator. This means that you can use available language tools to create absolute files, and then load those files into the emulator using the PC Interface.

The readers can operate from within the PC Interface or as a separate process from the MS-DOS prompt. You may need to use them separately if there is not enough memory on your personal computer to operate the PC Interface and a reader simultaneously. You also can include the readers as part of a "make file."

To build programs

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

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

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

3 Link all relocatable object files with a linker/loader that produces absolute object files in the IEEE-695 format or HP64000 (HP-OMF) format. (The IEEE-695 format is preferred.)

4 (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 you'll want your files in IEEE-695 absolute format, since it is widely supported in a number of environments.

If you compile or assemble programs on your PC, you will, of course, need to use a cross compiler or assembler. Language tools available from Microtec Research™ and Intermetrics™ provide cross compilers and assemblers. Another alternative is to use programs developed on a workstation using compatible cross-development language tools, and then downloading the absolute files to your PC. Most HP 9000 series workstations have language tools available which produce IEEE-695 format and HP64000 format absolute files.

Processor	**C Compiler**	**Assembler**
MC68040	HP B1463	HP B1465

In addition to the language tools already mentioned, you may use other language tools, provided they produce either IEEE-695 or HP64000 absolute file formats. (The PC interface also supports other file formats, including Motorola S-records, Intel hex, and Tektronix hex; however, these file formats do not support symbol records and are therefore less useful in the PC Interface.)

To prepare programs for the PC Interface

- To prepare HP64000 absolute files for use with the PC Interface, type:

```
RHP64000 [-q] [-f@fc] <filename>
```

or

- To prepare IEEE-695 absolute files for use with the PC Interface, type:

```
RIEEE695 [-u] [-q] [-f@fc] <filename>
```

where

[-u]	Specifies that the first leading underscore of a symbol is not removed.
[-q]	Specifies the “quiet” mode. This option suppresses the display of messages.
[-f@*fc*]	Specifies an optional function code *fc* to be supplied for the load addresses of data in the absolute file. The optional function codes available for the MC68040 emulator are:

<fc>	**Meaning**
none	Emulator defaults to supervisor space
s	Supervisor address space
u	User address space

<filename>	Specifies the name of the file containing the absolute program (<file.ext>) when using the IEEE-695 reader or the linker symbol file (<file.L>) when using the HP64000 reader.

The HP64000 reader produces an absolute and an ASCII file using the absolute file (<file.X>), the linker symbol file (<file.L>), and the assembler symbol files (<scr1.A>,<scr2.A>,...).

The IEEE-695 reader produces an absolute and an ASCII file using any IEEE-695 MUFOM (Microprocessor Universal Format for Object Modules) absolute file (<file.ext>).

The reader programs are installed in the directory named \hp64700\bin by default. For the reader to operate properly, this directory must be in the environment variable PATH usually defined in the "\autoexec.bat" file. If not, you must supply the directory name when executing the reader program.

You may want to incorporate the reader as the last step in your "make file," or as a step in your construction process. This to eliminate the possibility of having to exit the PC Interface due to space limitations. Also, the files necessary for emulation will then be updated automatically every time you modify your program.

Examples

The following command will create the files "TESTPROG.HPA" and "TESTPROG.HPS" using the HP64000 reader:

```
RHP64000 TESTPROG.L
```

The following command will use the IEEE-695 reader to create the files "TESTFILE.HPA" and "TESTFILE.HPS" and cause all absolute code/data to be loaded into program space.

```
RIEEE695 -f@p TESTFILE.HP
```

To load a program

1 Check to make sure that you have mapped memory as appropriate for your system design, as described in the "Memory configuration" section of chapter 7.

2 Select **M**emory **L**oad.

3 Use **Tab** and **Shift-Tab** to select the format of your absolute file. Press **<Enter>** to accept your choice.

4 Use **Tab** and **Shift-Tab** to select whether to load emulation memory, target system memory, both, or a custom foreground monitor. Press **<Enter>** to accept your choice.

5 Use **Tab** and **Shift-Tab** to select either user or supervisor address space to be loaded. Press **<Enter>** to accept your choice.

You can select from the following list of optional function codes:

<fcode>	**Meaning**
none	Emulator defaults to supervisor space
s	Supervisor address space
u	User address space

6 If you're using the HP64000 or IEEE-695 absolute file formats, **Tab** to select **yes** if you want the reader to re-read the absolute file and produce new .HPA and .HPS files. You would want to do this if you had changed any of the load options and needed to re-load the program in order to have the changes take effect (for example, deleting leading underscore characters). Press **<Enter>** to accept your choice.

7 If you're using the IEEE-695 format, **Tab** to select **yes** if you want the reader to delete the first leading underscore character from symbol names (C compilers add a leading underscore to most symbols). Press **<Enter>** to accept your choice.

8 Specify the name of the file containing the absolute program (<file.ext>) when using the IEEE-695 or the linker symbol file (<file.L>) when using the HP64000.

The file extension can be something other than those shown above, but cannot be ".HPA," ".HPS," or ".HPT." (<file>.HPT is a temporary file used by the reader to process the symbols.)

9 Press **<Enter>** to load the file, or press **<Esc>** to discard your entries and return to the PC Interface command line.

Using the file that you specify (TESTFILE.ABS, for example), the PC Interface does the following:

- It checks to see if two files with the same base name and extensions .HPS and .HPA already exist (for example, TESTFILE.HPS and TESTFILE.HPA).
- If TESTFILE.HPS and TESTFILE.HPA don't exist, the PC Interface invokes the reader to create them. The new absolute file, TESTFILE.HPA, is then loaded into the emulator.
- If TESTFILE.HPS and TESTFILE.HPA already exist but the creation dates and times are earlier than that of TESTFILE.ABS, the PC Interface invokes the reader to recreate them. The new absolute file, TESTFILE.HPA, is then loaded into the emulator.
- If TESTFILE.HPS and TESTFILE.HPA already exist but the creation dates and times are later those for the file TESTFILE.ABS, the PC Interface does not invoke the reader. The current absolute file, TESTFILE.HPA, is then loaded into the emulator.

Only the PC Interface does date/time checking. When you run the reader at the MS-DOS command line prompt, it will always update the absolute and symbol files.

When the reader operates on a file, a status message will be displayed showing which reader is in use. When the reader is finished, another message shows that the absolute file is being loaded.

The PC Interface can load HP64000 or IEEE-695 format absolute files, plus several other types of absolute files, into emulation or target system memory. IEEE-695 format is preferred, since it is so widely used and has a bit more flexibility in terms of symbol handling.

The memory type and function code parameters work with your memory map. Each memory map term has a memory type and function code associated with it. Based on what you enter here as the memory type and function code, the PC Interface selects all memory map terms that match the specified type and function code, and comes up with a set of addresses that are eligible for loading.

The PC Interface then reads your absolute file and loads only those addresses that are eligible. Addresses in your absolute file that are not eligible for loading are simply ignored.

Use of the memory type and function code parameters can save you time, if you change a memory map term and only want to reload the addresses affected by the change. On the other hand, if you have a range of addresses that overlap, but have different function codes, then you *must* load the two address ranges separately, since the absolute file does not contain any function code information. This is illustrated in the following example.

Examples

Suppose you using two MMU mappings, one which is user address space from 1000 thru 1fff hex. The other is supervisor address space from 1000 thru 1fff hex. You have absolute files called userprog.x and supprog.x. You load these programs by selecting **M**emory **L**oad twice. The first time, you select **u** as the function code, and specify userprog.x as the file name. The second time, you use the arrow keys and **Tab** to select **s** as the function code, and specify supprog.x as the file name. The programs are loaded into the correct address spaces.

To store a program

1 Select **M**emory **S**tore.

2 **Tab** to select a file format for the stored program data, then press **<Enter>**.

3 Type in a memory range: **<lower>..<upper>**, then press **<Enter>**.

4 Type in the name for the new absolute file. Press **<Enter>**.

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 you may want to reload the patch later for future testing. The store command allows you to do this.

You can save memory to files in any of the following formats:

- Raw HP64000 absolute.
- Intel hexadecimal.
- Tektronix hexadecimal.
- Motorola S-records.

Example

To save the memory locations of the init_system routine in an absolute file named **new**, select **M**emory **S**tore. Then select the Raw HP64000 file format, and type in the range init_system..init_system end and press **<Enter>**. Type in the name **new** and press **<Enter>** to save the memory range in the file.

Using Symbols

Symbol handling adds power to your interaction with the emulator. You can use symbols in expressions involving addresses, which frees you from trying to remember the addresses associated with the symbols.

When you build HP64000 absolute files, an assembler symbol file (with the same base name as the source file and a ".A" extension) is created. The assembler symbol file contains local symbol information. Also, a linker symbol file (with the same base name as the absolute file and a ".L" extension) is created. The linker symbol file contains global symbol information and information about the relocatable assembly modules that combine to form the absolute file.

When you load a file using the HP64000 file format, the file format reader collects global symbol information from the linker symbol file and local symbol information from the assembler symbol files. It uses this information to create a single symbol database with the extension .HPS. The process is similar for the IEEE-695 file format, except that all symbol information is collected directly from the absolute file.

The symbol database is accessible only to the PC Interface, and allows you to use symbols in address expressions. For example, you can specify a symbol in a run address or in a memory display command.

Additional symbol capabilities are available when you transfer symbol data to the emulator from the PC Interface. After you transfer symbol data to the emulator, you can display memory locations and analyzer trace lists with symbols.

What is symbol scoping?

Every symbol is defined within some program module, and has a *scope* associated with it, determined by the semantics of the source language you are using. A symbol's scope can be global, or it can be local to the module defining it. A symbol also has a name associated with it. If the symbol has global scope, then it's full name has the form ":<symbol_name>". If the symbol has scope local to a module, then it's full name has the form "<module name>:<symbol name>".

To illustrate, suppose A, B and C are modules, and that symbol **XYZ** is defined as a local symbol in both A and B, but is defined as a global symbol in module C. Then there are actually three instances of **XYZ**, which may be distinguished by using their “full” names: **A:XYZ**, **B:XYZ**, and **:XYZ**. Note that the global instance is *not* called **C:XYZ**, even though it is defined in module C. The global instance **:XYZ** is best thought of as being common to all modules. This implies that the symbol **XYZ** can only be defined as a global symbol once, otherwise we would have two symbols whose full names are both **:XYZ**.

A symbol can always be referred to by its full name. However, this can be tedious to type if the module name happens to be something like Spurious_Interrupt_Handler. The PC Interface allows the most recently referenced module to be the current local symbol module. This means that the full name of a symbol local to the current module has the form “<symbol name>”. Notice that “<module name>:” does not have to be typed. The full name of a global symbol that is not also defined as a local in the current module also has the form “<symbol name>”.

To illustrate, if module A happened to be current, then **XYZ** would refer to the local symbol **A:XYZ**. If **XYZ** is not defined as a local symbol in the current module, then **XYZ** refers to the (single) global instance **:XYZ**. Suppose D is yet another module, where D doesn’t define the symbol **XYZ** locally. Then, when either C or D is the current module, **XYZ** refers to the global instance **:XYZ**.

When you first load an absolute file, there is no current local symbol module. Thus, **XYZ** refers to the global instance **:XYZ** initially. Referencing a local symbol by its full name causes that symbol’s module to become current. So if you refer first to **A:XYZ**, then to **XYZ**, both references are to the local instance **A:XYZ**. This is convenient for specifying a range of addresses, which some commands allow.

For example, the range of addresses **XYZ..XYZ+100** in module A could be entered simply as **A:XYZ..XYZ+100**, instead of **A:XYZ..A:XYZ+100**. A module also becomes current when you display its local symbols using the **S**ystem **S**ymbols **L**ocal **D**isplay command.

To load a symbol database

- Load an absolute file in HP64000 or IEEE-695 format using the **M**emory **L**oad command.

or

- Select **S**ystem **S**ymbols **G**lobal **L**oad, type the name of the linker symbol file (for HP64000 format) or the absolute file (for IEEE-695 format), and press **<Enter>**.

When you load HP64000 or IEEE-695 format absolute files into the emulator, the corresponding symbol database is also loaded. A symbol database also can be loaded separately with **S**ystem **S**ymbols **G**lobal **L**oad. Use this command when you load multiple absolute files into the emulator. You can load the various symbol databases corresponding to each absolute file. When you load a symbol database, information from any previously loaded symbol database is removed from memory. That is, only one symbol database can be in memory at a time.

After a symbol database is loaded, both global and local symbols from the database can be used when entering expressions. Initially, however, no local symbol module is current. Therefore, to access global symbol XYZ, you would simply enter XYZ; whereas, to access local symbol XYZ in module A, you would need to enter A:XYZ (making A the current local symbol module).

To display global symbols

- Select **S**ystem **S**ymbols **G**lobal **D**isplay.

The symbols window automatically becomes the active window because of this command.

The global symbols display has two parts. The first part lists all the modules that were linked to produce this object file. You use these module names when you want to refer to a local symbol. They are case-sensitive.

The second part of the display lists all global symbols in this module. These names can be used in measurement specifications, and are case-sensitive. For example, if you wish to make a measurement using the symbol **Cmd_A**, you must specify **Cmd_A**. The strings **CMD_A** and **cmd_a** are not valid symbol names here.

To display local symbols

1 To display local symbols loaded into the PC Interface, select **S**ystem **S**ymbols **L**ocal **D**isplay.

2 Type the name of the module you want to display (from the first part of the global symbols list) and press **<Enter>**.

After you display local symbols with the "**S**ystem **S**ymbols **L**ocal **D**isplay" command, you can enter local symbols as they appear in the source file or local symbol display, rather than entering their full names in the form **module_name:symbol**. That is, when you display local symbols in a given module, that module becomes the current local symbol module.

If you have not displayed local symbols, you can still enter a local symbol by including the name of the module (which also causes that module to become the current local symbol module):

```
module_name:symbol
```

Remember that the only valid module names are those listed in the first part of the global symbols display, and are case-sensitive for compatibility with other systems (such as UNIX). Furthermore, module names and local symbols must be found in the currently loaded symbols database (see "To load a symbol database," earlier in this chapter.)

Examples

To display the local symbols from the "update_sys" module in the demo program, select **S**ystem **S**ymbols **L**ocal **D**isplay, type in **update_sys**, and press **<Enter>**.

To transfer global symbols to the emulator

- Select **S**ystem **S**ymbols **G**lobal **T**ransfer.

You can use the emulator's symbol-handling capability to make various displays (particularly those containing addresses) more readable. Many displays, such as trace listings and memory displays in mnemonic form, are produced directly by the emulator itself, then sent to the PC Interface (which simply puts the results on your screen). Consequently, you must inform the emulator which symbols you wish to see in any displays which it might produce. You do this by transferring the symbol database information to the emulator.

To remove global symbols from the emulator

- Select **S**ystem **S**ymbols **G**lobal **R**emove.

If you work with large symbol databases, you can easily fill emulator system memory. This may cause problems if you try to define new equates or macros using the Terminal Interface, or if you want to load more local symbol information. You can free memory by removing groups of symbols from the emulator.

To transfer local symbols to the emulator

- Select **S**ystem **S**ymbols **L**ocal **T**ransfer **A**ll to transfer local symbol information for all modules in the symbol database.

or

- Select **S**ystem **S**ymbols **L**ocal **T**ransfer **G**roup and type in one or more module names, separated by commas.

As with global symbols, you can transfer local symbol information to the emulator. You can transfer information for all modules in the database at once. Or, to save space in the emulator's system memory, you may prefer to transfer only the symbols related to the module(s) used in your measurements.

Examples

To transfer local symbols for all modules in the demo program, select:

```
System Symbols Local Transfer All
```

To transfer local symbol information from the assembly module **update_sys** in the demo program, select:

```
System Symbols Local Transfer Group update_sys <Enter>
```

To display transferred local symbols

- Select **S**ystem **S**ymbols **L**ocal **L**oaded. This command operates in much the same way as the **S**ystem **S**ymbols **L**ocal **D**isplay command, except that it only displays those symbols which have been transferred from the symbol database over to the emulator. This display option is useful for determining whether you need to transfer other local modules, or remove any which are no longer useful to free up emulator memory.

To remove local symbols from the emulator

- Select **S**ystem **S**ymbols **L**ocal **R**emove **A**ll to remove local symbol information for all modules in the symbol database.

or

- Select **S**ystem **S**ymbols **L**ocal **R**emove **G**roup and type in one or more module names, separated by commas.

When you're finished making measurements using one group of local symbols, you may want to remove them from the emulator to make room for new symbol groups.

Examples

To remove local symbol information for all modules in the demo program, select:

```
System Symbols Local Remove All
```

To remove local symbol information for the assembly module main in the demo program, select:

```
System Symbols Local Remove Group main <Enter>
```

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 to see if a sequence of values was written correctly. Also, you may need to modify one or more 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.

The display/modify mode can be either bytes, words, or longs. Otherwise the mode specified in the last **M**emory **D**isplay command determines how data is displayed. If you selected **don't care** when you specified "Memory data access width?" as part of the emulation configuration, the size you specify here will be used to access memory for the modification you specify.

To display memory as byte values

1 Select **M**emory **D**isplay **B**yte.

2 Type in one or more individual addresses or address ranges. If you specify multiple addresses, separate them with semicolons. Press <**Enter**>.

The emulator automatically breaks into the monitor if there is a request to access non-dual-port memory.

Example

To display the demo program's average temperature array in byte format, enter:

```
Memory Display Byte aver_temp.. <Enter>
```

To display memory as word values

1 Select **M**emory **D**isplay **W**ord.

2 Type in one or more individual addresses or address ranges. If you specify multiple addresses, separate them with semicolons. Press <**Enter**>.

The emulator automatically breaks into the monitor if there is a request to access non-dual-port memory.

Example

To display the memory from e00 to e1f as word values, enter:

```
Memory Display Word 0e00..0e1f <Enter>
```

To display memory as long-word values

1 Select **M**emory **D**isplay **L**ong.

2 Type in one or more individual addresses or address ranges. If you specify multiple addresses, separate them with semicolons. Press <**Enter**>.

The emulator automatically breaks into the monitor if there is a request to access non-dual-port memory.

Example

To display the processor’s interrupt vector table in long-word format, enter:

```
Memory Display Long 0..3ff <Enter>
```

To display memory as instruction mnemonics

1 Select **M**emory **D**isplay **M**nemonic.

2 Type in one or more individual addresses or address ranges. If you specify multiple addresses, separate them with semicolons. Press **<Enter>**.

When you use the **M**nemonic option, 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.

The emulator automatically breaks into the monitor if there is a request to access non-dual-port memory.

Example

To display memory containing the write_hdwr routine in the demo program, enter:

```
Memory Display Mnemonic write_hdwr.. <Enter>
```

To display memory repetitively

1 Select **M**emory **D**isplay **R**epetitively to repeatedly display the contents of the most recently displayed memory locations.

2 When you're finished using the changing memory display, press **<Esc>** to return to the PC Interface command line.

Repetitive memory display can be useful if you want to watch the activity in a certain memory area while a target system program is executing. However, it continuously uses the communication channel from the PC to the emulator, so you can't enter other measurements while the repetitive display is in process.

The address range and display format (byte, word, mnemonic, etc.) are those specified in the most recently entered **M**emory **D**isplay command.

To modify memory by bytes

1 Select **M**emory **M**odify **B**yte.

2 To modify a single memory location to a single value, type: **<address>=<value>**

or

To modify a range of memory locations to a single value, type:
<lower>..<upper>=<value>

or

To modify a range of memory locations with a list of values, type:
<lower>..<upper>=<value1>,<value2>, . . .

3 Press **<Enter>** to start the modify.

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.

See next page for examples.

Examples

To modify the byte at e1f hex to 43, select:

```
Memory Modify Byte 0e1f=43 <Enter>
```

To modify the value at an address identified by a symbol, enter:

```
Memory Modify Byte mysymbol=43 <Enter>
```

To modify the range of locations from e00 through e38 to zero, enter:

```
Memory Modify Byte 0e00..0e38=0 <Enter>
```

To modify the range of locations from e00 through e38 to “ABC,” enter:

```
Memory Modify Byte 0e00..0e38="ABC" <Enter>
```

You can combine multiple address ranges and value lists. For example:

```
Memory Modify Byte 0e1f=43;0e00..0e38="ABC" <Enter>
```

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:

```
Memory Modify Byte 0f01=3 <Enter>
```

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

```
Memory Modify Word 0f00=3 <Enter>
```

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).

To modify memory by words

1 Select **M**emory **M**odify **W**ord.

2 To modify a single memory location to a single value, type: **<address>=<value>**

or

To modify a range of memory locations to a single value, type:
<lower>..<upper>=<value>

or

To modify a range of memory locations with a list of values, type:
<lower>..<upper>=<value1>,<value2>, . . .

3 Press **<Enter>** to start the modify.

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.

To modify memory by long words

1 Select **M**emory **M**odify **L**ong.

2 To modify a single memory location to a single value, type: **<address>=<value>**

or

To modify a range of memory locations to a single value, type:
<lower>..<upper>=<value>

or

To modify a range of memory locations with a list of values, type:
<lower>..<upper>=<value1>,<value2>, . . .

3 Press **<Enter>** to start the modify.

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.

To copy memory

1 Select **M**emory **C**opy.

2 Specify the source address range in the form **<lower>..<upper>**, and then press **<Enter>**.

3 Type in a destination address to which the first address of the source range should be copied.

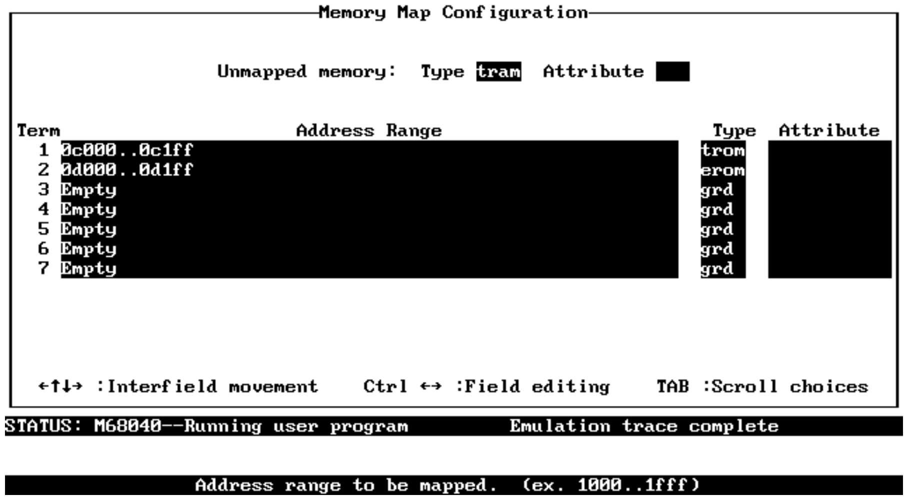
4 Press **<Enter>** to start the copy.

The **M**emory **C**opy command allows you to move blocks of code or data to different locations in memory. You can use it, along with memory remapping, to move a block of target system code into emulation memory.

<lower> and **<upper>** specify the lower and upper address ranges of the block that you want to move, while the **<destination>** address is the starting address of the range for the destination memory block.

Examples

In one situation, suppose you have relocatable ROM code in your target system in the range c000 through c1ff. You want to move this code into emulation memory for modification. You could set up a temporary memory map using **C**onfigure **M**emory **M**ap command as shown below:



```
─────────────────────Memory Map Configuration──────────────────────

          Unmapped memory:  Type tram  Attribute

Term                    Address Range                    Type  Attribute
  1 0c000..0c1ff                                         trom
  2 0d000..0d1ff                                         erom
  3 Empty                                                grd
  4 Empty                                                grd
  5 Empty                                                grd
  6 Empty                                                grd
  7 Empty                                                grd

  ←↑↓→ :Interfield movement    Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: M68040--Running user program        Emulation trace complete

             Address range to be mapped.  (ex. 1000..1fff)
```

Example Memory Mapping

Now copy the memory block from target ROM to emulation ROM:

`Memory Copy 0c000..0c1ff <Enter> 0d000 <Enter>`

In another situation, suppose you need to modify the exception vector table located in your target system ROM with the following conditions:

Initial Map:

Address Range	Memory Configuration
0000-ffff	64K bytes target ROM for the exception vector table and program code
10000-18fff	32K bytes target RAM for program data
19000-19fff	Other guarded memory
80000-80fff	Emulation RAM (foreground monitor)

Add Emulation Memory:

20000-203ff Emulation RAM (1K block for exception vector table).

Now copy the exception table from target ROM to emulation RAM:

```
Memory Copy 0..3ff <Enter> 20000 <Enter>
```

Change the VBR to point to the relocated exception vector table starting address:

```
Register Modify VBR=20000 <Enter>
```

To search memory

1 Select **M**emory **F**ind.

2 Type in the memory range to be searched in the form **<lower>..<upper>**, where **<lower>** and **<upper>** are the boundaries of the region to be searched. Press **<Enter>**.

3 Type in the expression or string for which you want to search. If you type in a character string, it must be delimited by single or double quote marks.

4 Press **<Enter>** to start the search.

Searching memory for values or character strings can help you determine whether a program is functioning correctly.

Sometimes you expect a data value to be written to a particular memory location during a program run. But, the program may accidentally write the value to the wrong location. You can search memory for the expression to see if the value was written to another location.

The **<lower>** and **<upper>** values are address expressions representing the lower and upper boundaries of the memory area to be searched.

If you're searching for character strings, **<string>** is an ASCII string delimited by ' (accent grave) or " (double quote). Remember that if one of the characters is part of the string, you should use the other character as a delimiter.

Example

To search an address range for the string "ommand," enter:

```
Memory Find  100..200 <Enter> "ommand" <Enter>
```

Using Processor Run Controls

Without the help of 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 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 your system after each program instruction, you can use the **P**rocessor **S**tep command to step through the program. You can break to the monitor program to examine on-chip resources such as RAM and registers using software breakpoints. And, you can reset the processor from the emulator.

To run a program

- To run the processor from the current program counter (PC) value, select **P**rocessor **G**o **P**C.

- To run the processor from a specific address value, select **P**rocessor **G**o **A**ddress and type in an address.

- To run the processor from reset, select **P**rocessor **G**o **R**eset.

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

The address is a 32-bit address expression. You can include supervisor or user function codes to specify the priviledge level for the run command. See <ADDRESS> in chapter 11 for more information.

The **P**rocessor **G**o **R**eset command pulses the processor reset line. The processor fetches the values at offsets 0 and 4 from the vector table and loads these values into the interrupt stack pointer and program counter registers. It then begins running from the program counter address value.

However, if you reset the emulator, break to the monitor, then run the processor from the current PC (or from a specified address), the processor does not fetch offsets 0 and 4 from the vector table. It simply uses the current stack pointer and program counter register values. Unless the stack pointer (and the PC, when doing a run from PC) have not been initialized to the same values as they would be by doing a **P**rocessor **G**o **R**eset, the run will fail. The **C**onfig **G**eneral form allows you to define initial values for the program counter and stack pointer, which are used whenever the monitor is entered from the reset state. See Chapter 7, “Configuring the Emulator,” for more information.

Examples

To run from the demo program’s starting location, select:

```
Processor Go Address ENTRY <Enter>
```

To run a program from a known address, such as 400h, select:

```
Processor Go Address 400 <Enter>
```

To break to monitor

- Select **P**rocessor **B**reak.

The emulation monitor is a program that provides various emulation functions, including register access and target system memory manipulation. You must break execution to the monitor before executing certain emulation commands, such as those accessing registers, emulation memory that is not dual-port, or target system memory. You also can use the break command to pause user program execution.

The emulator status changes to show that the processor is running in the monitor.

You can use either a foreground or background monitor. See the chapter titled "Configuring the Emulator" for more information. The book *Concepts of Emulation and Analysis* that was supplied with your product manuals also has an explanation of emulation monitors.

To step the processor

- To step from the current program counter value, select **P**rocessor **S**tep **P**c. Type in the number of instructions to step, and press **<Enter>**.

- To step from a specific address, select **P**rocessor **S**tep **A**ddress. Type in the number of instructions to step, and press **<Enter>**. Type in the starting address for the step, and press **<Enter>**.

The **P**rocessor **S**tep command lets you single-step the processor through program code. You can display registers after each step to help you locate the source of problems or verify correct operation. Or, you might want to modify a register, and then step the processor to check the result.

You can specify a step count (**<count>**) to step the processor more than one instruction. The default base is decimal. The default base for **<address>** is hex.

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, enter:

```
Processor Step Pc 1 <Enter>
```

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

```
Processor Step Pc 3 <Enter>
```

To step the processor five instructions from the **init_system** symbol in the demo program, enter:

```
Processor Step Address 5 <Enter> init_system <Enter>
```

To change the step function display

1 Select **P**rocessor **S**tep **E**vents.

2 **Tab** to select **y** if you want to display registers after each step, or **n** if you don’t want register display. Press **<Enter>**.

3 **Tab** to select **y** if you want the instruction mnemonic displayed at each step, or **n** if you don’t want mnemonic display. Press **<Enter>**.

4 Type the name of a command file to execute when stepping is complete. (This is optional).

5 Press **<Enter>** to accept the changes. Press **<Esc>** to discard the changes.

You can alter the way the step command works to give you only the information you need. This helps avoid display clutter for long step sequences.

Also, you can enhance the step command by specifying an optional command file to execute when stepping.

To reset the processor

- To reset the processor from the emulator and leave it reset, select **P**rocessor **R**eset **H**old.

- To reset the processor and begin executing in the monitor, select **P**rocessor **R**eset **M**onitor.

- To reset the processor 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. See Chapter 7 for more information.

Sometimes you may want to reset the emulation processor prior to a program run. The **P**rocessor **R**eset 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 behaves as if a **P**rocessor **G**o **R**eset 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 (as if a **P**rocessor **R**eset **M**onitor command had been given.)

Using Registers

The emulator allows you to display the processor's internal registers to determine the results of program execution. You can display a single register, or you can diaplay 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

- To display a register, select **R**egister **D**isplay **S**ingle, and then use the **Tab** key to select the register. Or, you can type in the register name. Press **<Enter>** to accept your selection.

- To display all registers from the processor's basic register set, select **R**egister **D**isplay **B**asic.

- To display all registers in a class of the processor's register set (such as the fpu), select **R**egister **D**isplay **C**lass. Use the **Tab** key to select the register class desired.

The available registers and register classes are as follows:

Register Class	**Register Names**
Basic	**pc, st, 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 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. Those registers are displayed in the dtt0/itt0 and dtt1/itt1 registers, respectively.

The processor must be running to allow register displays. If it is running in the monitor, the emulator does the display directly. If it's running the target 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 **R**egister **D**isplay command isn't allowed while you're running a user program. See Chapter 7, "Configuring the Emulator," for more information.

Examples

To display the processor's A0 register, enter:

```
Register Display Single a0 <Enter>
```

To modify registers

1 Select **R**egister **M**odify.

2 Use the **Tab** key to select a register name. Or, simply type the name of the register.

3 Press **<Enter>** to accept the name and move to the next field.

4 Type an integer or integer expression to which the register should be modified. (You cannot use symbols in these expressions.)

5 Press **<Enter>** to accept the value and modify the register.

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 processor must be running to allow modifying registers. See the previous instructions on "To display registers" for more information.

You can enter values into the three FPU control registers and the eight floating-point registers in hexadecimal values. You cannot use other numerical bases or floating-point values.

Examples

To modify the PC register to address 1000h, enter:

```
Register Modify pc <Enter> 1000 <Enter>
```

Notice that you can't use a symbol as part of the expression when modifying a register.

To modify the D3 register to 0, enter:

```
Register Modify d3 <Enter> 0 <Enter>
```

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 add breakpoints, set existing breakpoints, clear breakpoints but keep them available to be set again, or remove execution breakpoints from the emulator.

Add 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).

Adding or setting execution breakpoints in RAM

When you add or 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 clear or remove the breakpoint.

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

Adding or setting execution breakpoints in ROM

If you try to add or set an execution breakpoint at a location in ROM, the emulator will attempt to set the breakpoint instruction 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 contain 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.

To add an execution breakpoint

1 Select **B**reakpoints **A**dd.

2 Add one breakpoint by typing in a single address. Add several breakpoints by typing in several addresses separated by semicolons.

3 Press **<Enter>** to save the address selection.

When you add an execution breakpoint, it is automatically set. A breakpoint entry is also added in the system breakpoint table. The emulator uses the monitor to insert the breakpoint instruction. Therefore, you can't add a breakpoint when the emulator is reset. Select **P**rocessor **B**reak to begin running in the monitor.

Examples

To add an execution breakpoint at the symbol get_targets, enter:

```
Breakpoints Add get_targets <Enter>
```

To add an execution breakpoint at address 42a, enter:

```
Breakpoints Add 42a <Enter>
```

To set an execution breakpoint

- Select **B**reakpoints **S**et **S**ingle. Then type a single address to enable one breakpoint, or type in several addresses, separated by semicolons, to enable several breakpoints. Press **<Enter>** to save your selection.

or

- Select **B**reakpoints **S**et **A**ll to enable all currently defined breakpoints.

When a breakpoint is executed, it is disabled. If you want to reenable the breakpoint, use the **S**et option to the **B**reakpoints command. The emulator will search the breakpoint table for the address you specify. If there is a breakpoint entry for that location, the entry will be marked “enabled”. If the breakpoint is in RAM, the BKPT instruction will be written to memory at that location.

The emulator uses the monitor to enable the breakpoint. Therefore, you can’t enable a breakpoint when the emulator is reset. Use the **P**rocessor **B**reak command to begin running in the monitor.

Examples

To enable an existing execution breakpoint at target_temp, select:

```
Breakpoints Set Single target_temp <Enter>
```

To enable existing breakpoints at get targets and write_hdwr, select:

```
Breakpoints Set Single get_targets;write_hdwr <Enter>
```

To enable all existing breakpoints, select:

```
Breakpoints Set All
```

To set a ROM breakpoint in RAM

- Select **B**reakpoints **A**dd. Then type in the address where the breakpoint is to occur in the Address field. Now move the cursor to the field beside "Force hardware breakpoint:" and use the <Tab> key to select "yes". Press **<Enter>** to save your selection.

There may be times when you want to force 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 command ensures 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

- Select **B**reakpoints **C**lear **S**ingle. Then type a single address to disable one breakpoint, or type in several addresses, separated by semicolons, to disable several breakpoints. Press **<Enter>** to save your selection

or

- Select **B**reakpoints **C**lear **A**ll to disable all currently defined breakpoints.

Sometimes you will want to temporarily disable a breakpoint without removing it. The **C**lear option to the **B**reakpoints command lets you do this.

When you disable an execution breakpoint, the emulator replaces the BKPT instruction at the breakpoint address with the original instruction. It marks the breakpoint table entry as “disabled.” The processor won’t break to the monitor when the instruction at that location is executed.

The emulator uses the monitor to disable the breakpoint. Therefore, you can’t disable a breakpoint when the emulator is reset. Use the **P**rocessor **B**reak command to begin running in the monitor.

Examples

To disable an existing execution breakpoint at address 100, select:

```
Breakpoints Clear Single 100 <Enter>
```

To disable existing breakpoints at addresses 100 and 200, select:

```
Breakpoints Clear Single 100;200 <Enter>
```

To disable all existing breakpoints, select:

```
Breakpoints Clear All
```

To remove an execution breakpoint

- Select **B**reakpoints **R**emove **S**ingle. Then type a single address to remove one breakpoint, or type in several addresses, separated by semicolons, to remove several breakpoints. Press **<Enter>** to save your selection.

or

- Select **B**reakpoints **R**emove **A**ll to remove all currently defined breakpoints.

When you're finished using a particular breakpoint, you should remove the breakpoint table entry. The **R**emove option to the **B**reakpoints command lets you do this. In RAM, the original instruction is restored to memory, and the breakpoint table entry is removed.

The emulator uses the monitor to remove the breakpoint.

Examples

To remove an existing execution breakpoint at address 100, select:

```
Breakpoints Remove 100 <Enter>
```

To remove existing breakpoints at addresses 100 and 200, select:

```
Breakpoints Remove 100;200 <Enter>
```

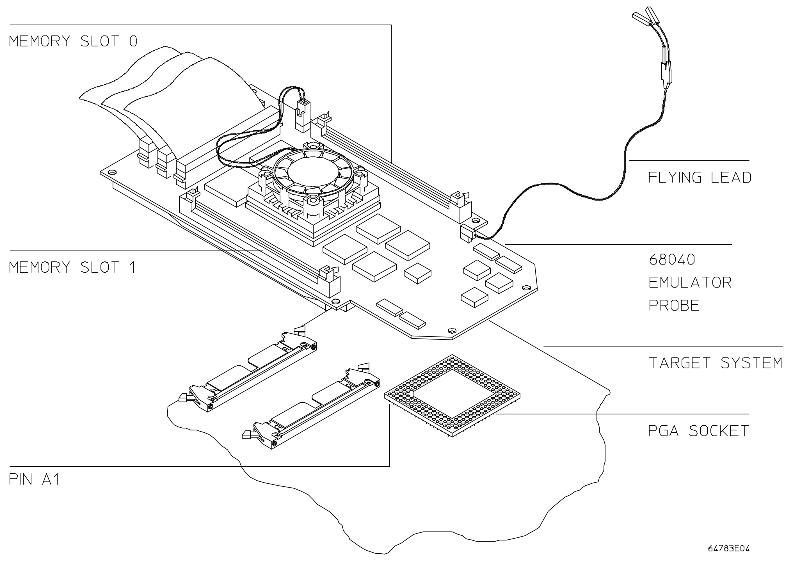
To remove all existing breakpoints, select:

```
Breakpoints Remove All
```

To display execution breakpoints

- Select **B**reakpoints **D**isplay. The status of each breakpoint is shown.

Using the Emulator In-Circuit



Emulator Probe Installation

As your target system design progresses, you'll want to test features of your target system program. Your program design will interact with real hardware instead of emulation memory locations.

You must connect the emulator probe to your target system to do in-circuit emulation. Then you can make analyzer measurements and use 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 the chapter titled "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 the chapter titled "Configuring the Emulator" for details of the emulation configuration.

To install the emulator probe in the target system

Caution *Possible damage to the emulator probe.* The emulation probe contains devices that are susceptible to damage by static discharge. You should 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 target system and emulator power are OFF before installing the emulator probe into the target system.

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 the 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 the target system socket. Make sure to align pin A1 of the emulator probe and the target system socket.

To power-on the emulator and target system

Caution *Possible damage to the emulator!* You must apply power to the emulator before you apply power to the target system. Otherwise, the emulator may be damaged.

1. Apply power to the emulator.

2. Apply power to the target system.

To shutdown the emulator and target system

1. Turn off power to the target system.
2. Turn off power to the emulator.

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 MC68040 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 emulator with the MMU enabled. The chapter titled, "Using Memory Management" provides detailed information to help you use the MMU most efficiently.

Disable the MMU unless you need it for address translation. You will still be able to use the transparent translation registers for defining cache modes, etc.

To enable the processor memory management unit

- To turn on the MMU in the emulation processor, obtain the emulation configuration with **C**onfig **G**eneral, and proceed as follows:
  Change the Monitor type to "foreground"
  Answer **n** to "Disable memory management unit?"

Refer to the chapter titled, "Configuring the Emulator" in this manual for details of setting up the emulation configuration.

Once enabled, the MMU can be set up by the operating system to manage logical (virtual) memory in physical address space. The selection of a root pointer and the value in the translation control register determine how the MMU will manage memory. The MMU must be enabled in the emulation configuration before the operating system can establish the MMU control values.

The target system will control the MMU during program execution by using the MDIS signal. The target system can disable the MMU, even if it is enabled in the emulation-configuration question.

A foreground monitor must be used when the MMU is enabled. Before you can enable the MMU in the emulation configuration, you must select "foreground" for the monitor type.

Note

Make sure the foreground monitor is mapped to memory space that has a 1:1 translation and is not write protected. Refer to the chapter titled "Configuring the Emulator" for instructions on how to map the foreground monitor to 1:1 address space in the MMU.

Examples

To enable the MMU so that the operating system can set it up to manage memory, perform the following:

Access the general configuration screen:

```
Config General
```

Use the arrow keys to move to the Monitor type field and enter:

Tab (to select **foreground**)

Use the arrow keys to move to the Disable memory management unit? field and enter:

Tab (to select **n**)

Save the changes and exit the general configuration screen:

<End><Enter>

Refer to the chapter titled "Configuring the Emulator" for further details on setting up the emulation configuration.

To disable the MMU, do the following:

Access the general configuration screen:

```
Config General
```

Use the arrow keys to move to the Disable memory management unit? field and enter:

Tab (to select **y**)

You can use the arrow keys to move to the Monitor type field and select foreground, background, or none, as desired.

To obtain additional information about the MMU, refer to the chapter titled "Using Memory Management" in this manual.

To view the present logical-to-physical mappings

- Select **P**rocessor **M**MU **M**appings. Then enter the logical address range for which you want the mappings displayed. Finally, specify whether to use the present TC register value or an alternate value you specify, the present URP value or a value you specify, and the present SRP value or a value you specify.

If the present TC register value disables the MMU, you must override it with an enable value before the emulator can read the MMU mappings.

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 thru 0ffffffffh), enter:

```
Processor MMU Mappings <End><Enter>
```

To see all of the logical-to-physical mappings for logical addresses from 0 through 0ffffh (when the URP root pointer is enabled), enter the command:

```
Processor MMU Mappings
```

Then use the arrow keys to enter in the Address(range): field, and type in:

```
0..0ffff  <End><Enter>
```

To see the logical-to-physical mappings for the pages that contain logical address 40f0h, enter the command:

```
Processor MMU Mappings
```

Then use the arrow keys to enter in the Address(range): field, and type in:

`40f0` **`<End><Enter>`**

To see only the mappings for supervisor function code in the address range from 0 through 0ffffh, enter the command:

```
Processor MMU Mappings
```

Then use the arrow keys to enter in the Address(range): field, and type in:

0..0ffff@s **<End><Enter>**

To see only the mappings for user function code in the address range from 0 through 0ffffh, enter the command:

```
Processor MMU Mappings
```

Then use the arrow keys to enter in the Address(range): field, and type in:

0..0ffff@u **<End><Enter>**

To show all of the valid mappings in the mapping tables for selected values of the TC, URP, and SRP registers, ignoring the present values of those registers, enter:

```
Processor MMU Mappings
```

Place the cursor in the "Use TC reg:" field and press the **Tab** key to obtain "no". A Value: field will open. Specify the value you want to be used in place of the present value of the TC register.

Use TC reg:no **Value:**0C000

Place the cursor in the "Use URP reg:" and "Use SRP reg:" fields and use the **Tab** key to select "no". Then enter the desired values for those registers in the associated "Value:" fields.

Finally, press **<End><Enter>**

Note that the hexadecimal number base is assumed for the TC, URP, and SRP registers.

To see translation details for a single logical address

- Select **P**rocessor **M**MU **T**ables. Then enter the logical address for which you want the translation details displayed. In the Table level: field, use the **Tab** key to select all tables used by your logical address. Finally, specify whether to use the present TC register value or an alternate value you specify, the present URP value or a value you specify, and the present SRP value or a value you specify.

If the present TC register value disables the MMU, you must override it with an enable value before the emulator can read the MMU mappings.

Examples

To see how logical address 40f0h is mapped through the translation tables to its corresponding physical address, enter:

```
Processor MMU Tables
```

In the Address: field, type in 40f0

Finally, press **<End><Enter>**

To see how logical address 1000h in user space is mapped through the translation tables, enter the command:

```
Processor MMU Mappings
```

Then use the arrow keys to enter in the Address(range): field, and type in:

1000@u

Finally, press **<End><Enter>**

To see details of a translation table used to map a selected logical address

- Select **P**rocessor **M**MU **T**ables. Then enter the logical address for which you want the table details displayed. In the Table level: field, use the **Tab** key to select the desired translation table whose details you want displayed. Finally, specify whether to use the present TC register value or an alternate value you specify, the present URP value or a value you specify, and the present SRP value or a value you specify.

If the present TC register value disables the MMU, you must override it with an enable value before the emulator can read the MMU mappings.

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 all tables in the Table level: field.

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, enter the command:

```
Processor MMU Tables
```

In the Address: field, type in 1250

In the Table level: field, use the **Tab** key to select **a**.

Finally, press **<End><Enter>**

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.

5

Using the Analyzer

How to record program execution in real-time

The *emulation-bus analyzer* is a powerful tool that allows you to view the execution of your program in real-time, by monitoring all activity on the system bus. The bus is a group of lines carrying address, data, and status information between various parts of the target system and the emulator. Any time the digital values on the bus line are stable, the bit values on the bus lines are said to comprise a *bus state.* The bus state changes continually as information is transferred from place to place in the system. When the bus state changes, it marks the completion of one *bus cycle*, and the beginning of another.

You monitor bus activity by capturing a *trace* of bus states in a special emulator buffer called the trace buffer. By examining the trace list you can tell, for example, whether a certain address was accessed (as in an instruction fetch), and can even tell which accesses were reads and which were writes. Furthermore, if you want, you can tell how much time elapsed between one state and the next. Because there is usually far more information carried over the bus than you care to see, powerful triggering and sequencing capabilities ensure that the analyzer captures only the information you need. That way, you don't waste time searching through long trace lists for the information you're looking for.

This chapter explains how to use analyzer capabilities.

Making Basic Analyzer Measurements

Most measurements can be made using just a few analyzer commands. These commands allow you to:

- Start or stop a trace measurement.
- Display the trace status.
- Display the trace list.
- Define a simple trigger qualifier.
- Define a simple storage qualifier.
- Set the trigger position in trace memory.

In addition to these essential functions, the analyzer has powerful triggering, storage and trace list display capabilities, that you can use to make complex measurements. These features are described in other sections of this chapter.

To begin a trace measurement

- Begin an emulation-bus analyzer trace by selecting: **A**nalysis **B**egin

When you start a trace, the analyzer begins recording data according to your trigger and storage specifications. When the trace is complete, or halted (by you), you can display the data.

More information on trigger and storage specifications is included later in this chapter.

To halt a trace measurement

- Halt an emulation-bus analyzer measurement by selecting: **A**nalysis **H**alt

You may occasionally need to halt a trace because you determine that the analyzer isn’t capturing the data you expect (the trace buffer isn’t filling). That is, the analyzer status shows it is still “running” when you think it should be “complete.” Halt the trace currently in progress using **A**nalysis **H**alt. You may also need to respecify your trigger and storage terms (later in this chapter), and then restart the trace.

To view the trace status

- Check the emulation-bus analyzer trace status by looking at the PC Interface status line, near the bottom of the screen. The analyzer status is shown on the right half of the line.

The trace status display shows whether the trace is “running,” “halted,” or “complete.”

To display the trace list

1 Display the trace list using the default parameters by selecting: **A**nalysis **D**isplay

2 Press **<End><Enter>** to accept the default display.

The trace list is a 1024 state deep buffer—if the analyzer has been set up to count states or time, and a 512 state deep buffer otherwise. You can selectively display portions of the buffer by setting the fields in the Analysis Display form. See the section “The Trace Display” (later in this chapter) for more information.

To define a simple trigger qualifier

1. Define a simple trigger on an address value by selecting: **A**nalysis **T**race **M**odify

2. Use the arrow keys to move to the **Trigger on** field and type **a**. Press **<Enter>**.

3. Now you see another form. Use the arrow keys to move to the address field next to the **a=** label. Type in the address value, then press **<Enter>**.

4. Press **<End><Enter>** *twice* to save the trigger specification.

Typically you'll want to trigger the analyzer (begin storing data) once a certain program location is reached. You can use either a simple address expression, or one that includes symbols.

Example

Trigger the analyzer when the demo program reaches the location write_hdwr in the update_sys module:

```
Analysis Trace Modify
```

Use the arrow keys to move to the **Trigger on** field and type **a**. Press **<Enter>**.

Now you see another form. Move to the address field next to **a=**. Type in the address value **write_hdwr**.

Press **<End><Enter>** twice to save the trigger specification.

To define a simple storage qualifier

1 Store only bus states that reference a particular address by selecting: **A**nalysis **T**race **M**odify

2 Use the arrow keys to move to the **Store** field. Type **a**, then press **<Enter>**.

3 Now you see another form. Use the arrow keys to move to the field next to the **a=** label. Type in an address, then press **<Enter>**.

4 Press **<End><Enter>** *twice* to save the new trace specification.

If you want to store only the accesses to a specific location, you can use the trace storage qualifier. The trigger qualifier (previous section) effectively ignores all bus states prior to the trigger. The storage qualifier then filters all subsequent states after the trigger, storing only those that match the qualifier. As described later in this chapter, the storage qualifier can specify a pattern that matches a number of states, instead of matching just one.

Example

The Analysis Display window displays timing information. If you want to measure how often the demo program accesses the write_hdwr symbol, set the trigger pattern to **a=write_hdwr** (see the previous section). Then set the storage qualifier to **a** (step 2 above). When you get to the next form (step 3), the field next to the **a=** label already contains the address **write_hdwr**, because you entered it previously as the trigger qualifier. So, press **<End><Enter>** twice to accept this value.

Start the trace:

```
Analysis Begin
```

Then run the program:

```
Processor Go Address ENTRY <Enter>
```

Display the trace:

```
Analysis Display <End><Enter>
```

To set the trigger position

1 Position the trigger term by selecting: **A**nalysis **T**race **M**odify

2 Press the **<End>** key to move to the last field in the trace specification.

3 Use the **<Tab>** key to select **start, center** or **end** to position the trigger at the start, center or end of the trace. Or type in a value from 1 through 512 (or 1 through 1024 if counting is off) to position the trigger at that trace state.

4 Press the **<Enter>** key to save the new trace specification.

The previous sections of this chapter indicated that the analyzer begins capturing data when the trigger qualifier is matched. Actually, the analyzer captures data continuously before the trigger as well, but overwrites this data when the trigger is matched. By setting the trigger position, you can keep some (or all) of the states that were captured prior to the trigger. For example, you might want to see all the program events leading to a particular access. So, you can define that access as the trigger term, and then position the trigger at the end of the trace.

Example

To position the trigger 10 states after the beginning of the trace (so that 10 states will appear before the trigger in the trace list), select:

```
Analysis Trace Modify
```

Press the **<End>** key to move to the last field. Type **10** and press **<Enter>** to save the trace specification.

Displaying the Trace List

The *trace list* is your view of the analyzer's record of processor bus activity. You can control how the information in the trace list is presented, 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.

To change the trace format

1 Select **A**nalysis **F**ormat.

2 Use the arrow keys to move to one of the **label** fields.

3 **Tab** or **Shift-Tab** to change the label to one of the following:

addr (for address lines)
data (for data lines)
mne (for instruction mnemonics)
stat (for processor status information)
count (state and time counts)
seq (sequencer state change indicator)
—OFF— (no label in this position)

Press **<Enter>** to accept the new label definition.

4 If a **base** specification is available for the label, you can move to it with the arrow keys. Press **<Tab>** to select one of the following bases:

hex (hexadecimal)
bin (binary)
oct (octal)
dec (decimal)
asc (ascii)

Press **<Enter>** to accept the base specification.

5 For the addr label, you may specify a **width** parameter in the range 4..50. Type in the desired value and press **<Enter>**.

6 For the count label, you may specify **rel** to show the count (or time) relative to the previous state , or **abs** to show the absolute count between this state and the trigger state. Press **Tab** to select the desired setting and press **<Enter>**.

7 Press **<End>**, then **<Enter>** to save the format changes and exit the format screen. Press **<Esc>** to exit the format screen and discard your changes.

The **A**nalysis **F**ormat form specifies how data is arranged on the screen when you display the trace list. You can specify multiple options on the form. The sequence of the options in the form determines the sequence of the columns in the trace list display. Each row of the form specifies what will appear in one column of the output. The rows on the form—from top to bottom, correspond to the columns—from left to right, of the trace list display.

When you enter an **A**nalysis **F**ormat command with a new set of options, the previous trace format is destroyed, and the new options determine the new format. So if you display the trace list, then change the format and display the trace list again, you will see the same data as before, but formatted differently.

Example

If you want to set counting relative to the trigger state, perform the following:

Access the format screen:

`Analysis Format`

Use the arrow keys to move to a **label** field and enter:

`Tab` `(to select` **`count`**`)` **`<Enter>`**

Use the arrow keys to move to the count qualifier field and enter:

`Tab` `(to select` **`abs`**`)` **`<Enter>`**

Save the changes and exit the format screen:

<End><Enter>

To display the trace list

1 Select: **A**nalysis **D**isplay.

2 The cursor is in the “Dequeuing” field. **Tab** to select **on** if you want software dequeuing of the instruction stream (see next page), or **off** to disable dequeuing. Press **<Enter>**.

3 Now the cursor is in the “Start” field. Type in a number to specify the first state to be displayed (0 is the trigger state) from the trace list. Press **<Enter>**.

4 Now the cursor is in the “End” field. Type in a number to specify the last state to be displayed from the trace list. Press **<Enter>**.

5 If you enabled software dequeuing in step 2, the cursor will be in the “Operand” field. (This field appears only the first time that you display the most recent trace, or if you changed the “start” state in step 3.) Type in a number that represents the operand cycle associated with the first disassembled instruction. (You may need to examine the trace list first before specifying this value). Press **<Enter>**.

6 The cursor is now in the “Cycles” field. **Tab** to select **all** if you want to see all instruction and operand cycles as they were captured by the analyzer, or select **instr only** if you want to see only the instruction cycles. Press **<Enter>**.

7 The cursor is now in the “Address mode” field. **Tab** to select **Absolute** if you want to see numeric addresses in the address field, **Symbols** if you want to see only symbols in that field (when available), or **Both** if you want to see a mix of the numeric addresses and symbols. Press **<Enter>**.

8 The cursor is now in the “Start on” field. (This field appears only the first time that you display a trace, or if you entered a start state other than the default.) **Tab** to select **high word** if you want the analyzer to disassemble instructions beginning with the upper 16 bits of captured data, or **low word** to disassemble instructions beginning with the lower 16 bits of captured data.

9 Press **<End><Enter>** to accept your choices and display the trace list. Press **<Esc>** to abort the **A**nalysis **D**isplay command.

The trace list is a 512 or 1024 state deep buffer. You can selectively display portions of the buffer using the Analysis Display form. The trigger state is always state 0, so a negative number indicates a state prior to the trigger (if any).

The MC68040 PC Interface presents trace lists with the trace data disassembled into an instruction stream. You can choose to show all bus cycles (instructions and operands) or only the instruction cycles.

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

A dequeued trace list is available through the disassembly options. In the 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 the branch if the dequeuer determines that the branch was taken. NOT TAKEN is shown if the dequeuer determines that the branch was not taken. ?TAKEN? means the dequeuer was not able to definitely determine whether or not the branch was taken. If you read down the trace list and see that the branch was taken, restart disassembly at the trace list line number of the branch destination. You will need to use the **low word** option if the destination opcode is in the low word at the destination address.

Example

Suppose you want to display ten states from the trace list, starting at the trigger, without dequeuing. Access the trace list display screen and set up the display using the "Dequeuing," "Start," and "End" fields:

```
Analysis Display
Tab (to select off) <Enter>
Tab (to select 0) <Enter>
Tab (to select 10) <Enter>
```

Save the changes and exit the display screen:

```
<End><Enter>
```

To change the trace depth

1 Select **A**nalysis **T**race **M**odify.

2 Use the arrow keys to move the cursor to the field next to "Count" and press the **Tab** key to select **off** to disable counting or **on** to enable counting. Press **<Enter>**.

3 Press **<End>**, then **<Enter>** to save your changes and exit the trace specification form. Press **<Esc>** if you want to discard your changes and exit the trace specification form.

The analyzer's state/time counter uses half of the analyzer's state memory resources. Therefore, when you use the time or state counter, the analyzer's trace depth is 512 states. You can double the trace depth to 1024 states by setting the state/time counter to off.

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 9 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 with the command:

```
Processor Break
```

3 Load the deMMUer with the command:

```
Processor DeMMU Load or Verbose_Load
```

4 Enable the deMMUer with the command:

```
Processor DeMMU Enable
```

5 Continue execution of your target program with the command:

```
Processor Go Pc, or Address, or 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 be able to reverse translate the physical addresses according to the MMU setup at the time you issued the **P**rocessor **D**eMMU **L**oad 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 **P**rocessor **D**eMMU **L**oad 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

- Store the deMMUer setup in a deMMUer setup file with the command:

```
Processor DeMMU File Store <file>
```

Stores a deMMUer setup file (with a ".ED" suffix) by reading the present content of the MMU registers and the MMU tables. You can edit this file to remove address ranges that do not need to be reverse translated. Then you can load this file to set up the deMMUer to reverse translate only those ranges that need to be reverse translated during your next test.

To load the deMMUer from a deMMUer setup file

- Load the deMMUer from a deMMUer setup file with the command:

  ```
  Processor DeMMU File Load or Verbose_Load <file>.ED
  ```

  Files that store setup information for the deMMUer have filenames that end in ".ED".

  The <file>.ED should contain only those address ranges that need to be reverse translated. When this file is loaded, the deMMUer creates a set of reverse translations for it, ignoring the present content of the MMU registers and the MMU tables in the emulator.

To trace program execution in physical address space

- Disable the deMMUer with the command:

  ```
  Processor DeMMU Disable
  ```

  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 Measurements

Earlier in this chapter the idea of a trigger and storage qualifier were introduced, allowing you to selectively capture only the states you want to see. However, the analyzer is actually capable of much more. You may find that:

- the trace buffer fills before reaching the states you want to see;
- you must search through a long trace list to find only a few states pertinent to your measurement problem.

The HP 64700 analyzer has triggering and sequencing capabilities that help solve these problems. A *simple trigger* tells the analyzer to start storing data when a certain bus state is encountered. A *sequence* is a more complex specification that specifies a series of bus states that must be matched before the trigger is reached. A triggering sequence consists of *sequence terms*, each of which specifies which state(s) to match next, and which states to store in the trace buffer while looking for that state.

So far in this chapter, triggers and storage qualifiers have been single addresses. But each trigger and storage qualifier should really be thought of as an expression describing a *pattern* that can match one or more states. For example, an expression could select all states in which the address lines carry a value ending in 1111 (binary). An address is just a special case of a pattern that only matches a single address. Throughout this chapter, such expressions are represented by the <expression> symbol. Expressions are described more fully, later in this chapter.

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

To insert a sequence term

1. Select **A**nalysis **T**race **M**odify.

2. Use the arrow keys to move the cursor to the term number of the sequence term where you want to insert a new term. The new term will be inserted before that term.

3. **Tab** until the term number field displays **I**, and press **<Enter>**. A new sequence term is inserted.

4. Make any changes needed to qualifiers (see other parts of this chapter for details.)

5. Press **<End>**, then **<Enter>**, to save your changes and exit the trace specification form. Press **<Esc>** if you want to discard your changes and exit the trace specification form.

You can specify up to eight sequence terms. At least two are always required, because the first term specifies the trigger and the analyzer triggers on entry into the succeeding term.

When you start a trace, the analyzer searches for the “Find” state of the first sequence term, then searches for the “Then find” state of the second term, and so on until it reaches the “Trigger on” term.

Any one of the eight sequence terms may be specified as the trigger level. The trigger state is always line 0 in the trace list.

When you insert a new sequence term in the middle of the sequence, succeeding terms are incremented.

Example

Assume that you have four sequence terms and you want to insert a trigger as the third sequence term.

Select **A**nalysis **T**race **M**odify.

Insert the new sequence term by using the arrow keys to position your cursor on the term number of sequence term 3.

Use the **Tab** key to cycle through the term number options until the field displays **I**. Press **<Enter>**.

A new sequence term will now be displayed. Sequence terms 3 and 4 have been moved down the screen and renumbered 4 and 5.

Make the new term the trigger level by placing the cursor on the term number of sequence term 3 and pressing **Tab** until the field displays **T**. Press **<Enter>**. The “Find” or “Then find” label changes to “Trigger on.”

To remove a sequence term

1 Select **A**nalysis **T**race **M**odify.

2 Use the arrow keys to move the cursor to the term number of the sequence term you want to remove.

3 **Tab** until the term number field displays **D**, and press **<Enter>**. The sequence term is deleted.

4 Make any changes needed to qualifiers (see other parts of this chapter for details.)

5 Press **<End>**, then **<Enter>**, to save your changes and exit the trace specification form. Press **<Esc>** if you want to discard your changes and exit the trace specification form.

You may want to delete sequence terms to remove unneeded qualifications from the sequence specification. When you delete a sequence term, any terms following it are shifted up to fill the gap.

Example

Suppose that there are 4 sequence terms on the screen and you want to remove term number 3.

Select **A**nalysis **T**race **M**odify.

Use the arrow keys to move to the term number field that holds a 3. **Tab** until the field displays **D** and press **<Enter>**.

Sequence term number 4 will be shifted up and will now be term number 3.

Save the changes and exit the format screen by entering **<End><Enter>**.

To assign the trigger term

1 Select **A**nalysis **T**race **M**odify.

2 Use the arrow keys to move the cursor to the term number of the sequence term that you want to use as the trigger level.

3 **Tab** until the term number field displays **T**, and press **<Enter>**. The “Find” or “Then find” label changes to “Trigger on.”

4 Make any changes needed to qualifiers (see other parts of this chapter for details.)

5 Press **<End>**, then **<Enter>**, to save your changes and exit the trace specification form. Press **<Esc>** if you want to discard your changes and exit the trace specification form.

Any but the last of the eight sequence terms may be specified as the trigger level. The trigger state is always line 0 in the resultant trace list.

The selected term specifies the trigger qualifier. The analyzer triggers on entry into the succeeding term.

Example

Suppose that there are six sequence terms on the screen and you want to make term number 4 the trigger term.

Select **A**nalysis **T**race **M**odify.

Use the arrow keys to move to the term number field that holds a 4. **Tab** until the field displays **T** and press **<Enter>**.

The “Find” or “Then find” label changes to “Trigger on.”

Save the changes and exit the trigger screen by entering **<End><Enter>**.

To reset the analyzer

- Select **A**nalysis **T**race **R**eset.

When you reset the analyzer trace specification, the sequencer is set to the default two-term sequence in which it stores any state and triggers on any state. All pattern and range specifications are also reset. (See “To define trace patterns” and “To define a range” for more information.)

To define a qualifier

1. Select **A**nalysis **T**race **M**odify.

2. Use the arrow keys to move to the qualifier field for which you want to specify an expression. (The analyzer searches for the expression you specify before taking action, such as counting a state, moving to a new sequence term, and so on.)

3. **Tab** to select a state qualifier from the following list of states, ranges and patterns:

   any state, no state, r, !r, a, b, c, d, e, f, g, h, arm

   Or, you may enter a complex expression. Type in the complex expression using patterns and ranges according to the following rules:

   - The patterns, range and arm qualifier are divided into two disjoint sets.

     <SET1>={a,b,c,d,r,!r}
     <SET2>={e,f,g,h,arm}

     (The **arm** qualifier is discussed in the section on coordinated measurements.)

- You can form expressions by inserting intraset operators between members of the same set. The operators are:

  ~ (intraset logical NOR)
  | (intraset logical OR)

  If you form an expression using these operators, the operator must remain the same for all members of the same set. (See the examples).

- You can form expressions by inserting interset operators between members of **<SET1>** and **<SET2>**. The operators are:

  and (logical and)
  or (logical or)

  The order in which you put the sets does not matter.

See the chapter titled “Expression Syntax” for a more complete description.

4 Press **<Enter>** if you need to define the expressions associated with the patterns and ranges in the expression. (See “To define trace patterns.”) Otherwise, skip to step 5.

5 Press **<End>**, then **<Enter>**, to save your changes and exit the trace specification. Press **<Esc>** to discard your changes and exit the trace specification.

Complex expressions allow you to build more complicated trace qualifiers with multiple conditions. Since the complex expressions are built from the trace patterns, which contain simple expressions, you can build qualifiers with multiple logical operators.

Examples

These are valid complex expressions:

```
a~b and e|arm
a or e
b and e and c and g
```

These are invalid complex expressions:

```
a~c
c|f and g
```

To create a numeric expression

- Form logical expressions by combining numeric values and logical operators to produce a numeric result.

The simplest numeric expressions consist of numbers and radix indicators. The radix indicators are:

Y y (binary)
Q q O o (octal)
T t (decimal)
H h (hexadecimal (default))

See Chapter 11, “Expression Syntax,” for more details on numeric expressions and the available logic operators.

Examples

The following are valid numeric expressions:

```
1XXX0Y<<3
(340q*7)/2
0ffa^32T
52T*7a
```

To define trace patterns

1 Select **A**nalysis **T**race **M**odify.

2 Use the arrow keys to move to the qualifier you want to change and enter a pattern, range or complex expression. (See “To define a qualifier” for more details.) Press **<Enter>**.

3 Now you are in the pattern modification screen. Use the arrow keys to move to the equality field next to the desired pattern. To set the pattern equal to an expression, **Tab** until this field displays **=**. To set the pattern not equal to the expression, **Tab** until the field displays **!=**. Press **<Enter>** to move to the **addr** field for that pattern.

4 In the **addr** field, you can enter an address expression, using either numeric expressions, symbols, or a combination thereof. Or, you can leave the field blank. Press **<Enter>** to move to the **data** field.

5 In the **data** field, you can enter data expressions using numeric expressions, or leave the field blank. Press **<Enter>** to move to the **stat** field.

6 In the **stat** field, you can enter a status expression using numeric values. Or, you can use the **Tab** key to select from different predefined status qualifiers. (See STATUS in the "Expression Syntax" chapter for a list.) You can use logical operators to combine the status qualifiers.

7 When you finish modifying patterns, press **<End>** to move to the last field in the form. Press **<Enter>** to accept your changes and return to the sequencer definition screen. You can exit that screen by pressing **<End>**, then **<Enter>**.

If you want to discard your changes, press **<Esc>**.

The analyzer provides eight pattern variables, to which you assign simple expressions. Then you use these patterns to build more complicated qualifiers for the primary and secondary branch qualifiers.

By specifying the address, data and status values for a particular pattern, you can use a pattern to specify a complete processor bus state.

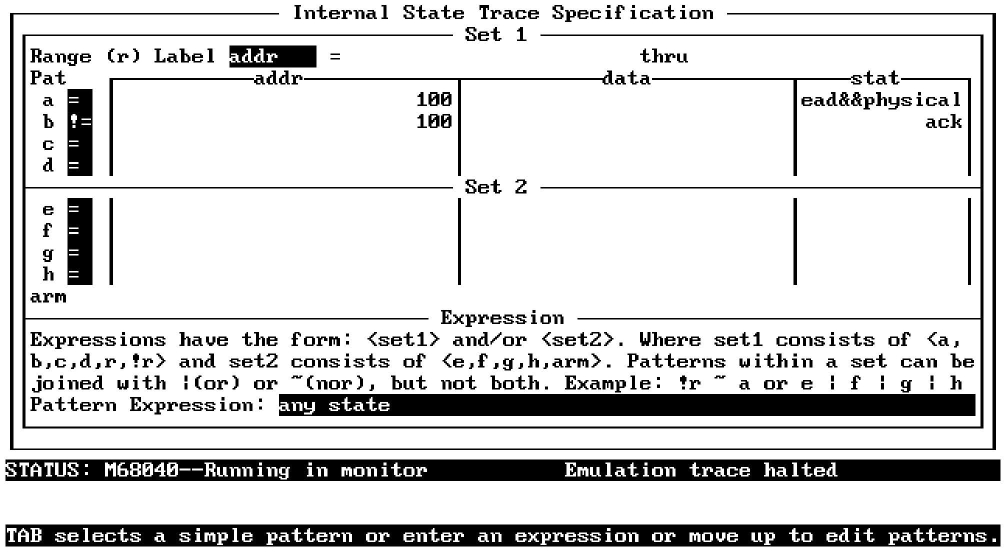
The equality setting affects the logical connectives between the address, data and status values. If you set the equality to =, the logical connective is **and**. If you set the equality to **!=**, the logical connective is **or**. For example, you could have the following relationships of address 100h and selected status values:

```
addr=100 and status=read from physical memory
addr!=100 or status!=acknowledge access
```

You can combine the predefined status qualifiers. For example, if you want to see only read cycles from physical memory, enter the following in the stat field:

```
read&&physical
```

See the following display for an example of how these are entered.



```
───────────── Internal State Trace Specification ─────────────
─────────────────────────── Set 1 ────────────────────────────
Range (r) Label addr     =                    thru
Pat     ─────────addr─────────┬─────────data─────────┬───stat───
 a =                       100│                      │ead&&physical
 b !=                      100│                      │        ack
 c =                          │                      │
 d =                          │                      │
─────────────────────────── Set 2 ────────────────────────────
 e =                          │                      │
 f =                          │                      │
 g =                          │                      │
 h =                          │                      │
arm
───────────────────────── Expression ─────────────────────────
Expressions have the form: <set1> and/or <set2>. Where set1 consists of <a,
b,c,d,r,!r> and set2 consists of <e,f,g,h,arm>. Patterns within a set can be
joined with |(or) or ~(nor), but not both. Example: !r ~ a or e | f | g | h
Pattern Expression: any state

STATUS: M68040--Running in monitor          Emulation trace halted

TAB selects a simple pattern or enter an expression or move up to edit patterns.
```

Analysis Trace Modify Form - Level 2

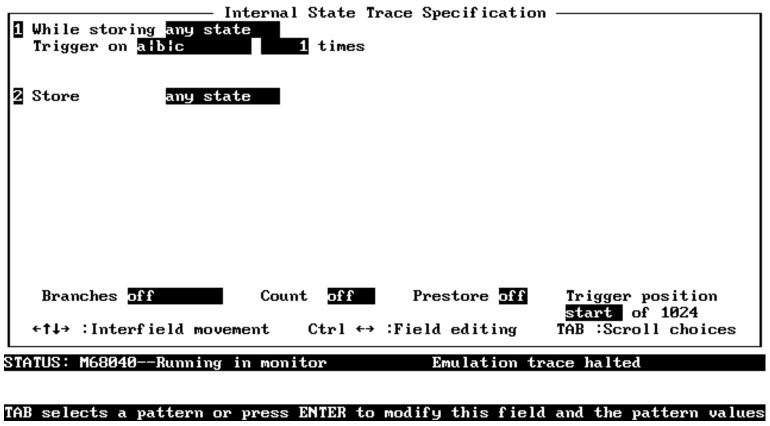
Example

Suppose that you want to trace and trigger on an access to either address 100, 200, or 300 in your program.

Set up the analyzer:

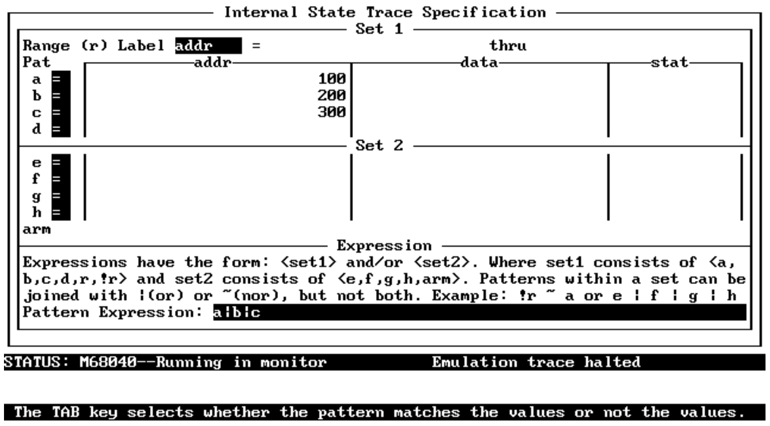
```
Analysis Trace Modify
```

Modify the trace forms as shown:



```
                 ──── Internal State Trace Specification ────
1 While storing any state
  Trigger on a|b|c          1 times

2 Store         any state




   Branches off        Count off     Prestore off   Trigger position
                                                    start  of 1024
  ←↑↓→ :Interfield movement    Ctrl ↔ :Field editing    TAB :Scroll choices
STATUS: M68040--Running in monitor          Emulation trace halted

TAB selects a pattern or press ENTER to modify this field and the pattern values
```

Analysis Trace Modify Form - Level 1



```
                 ──── Internal State Trace Specification ────
                              ──── Set 1 ────
 Range (r) Label addr   =                      thru
 Pat    ─────────addr──────────────────data──────────────stat──
  a =                   100
  b =                   200
  c =                   300
  d =
                              ──── Set 2 ────
  e =
  f =
  g =
  h =
 arm
                            ──── Expression ────
 Expressions have the form: <set1> and/or <set2>. Where set1 consists of <a,
 b,c,d,r,!r> and set2 consists of <e,f,g,h,arm>. Patterns within a set can be
 joined with |(or) or ~(nor), but not both. Example: !r ~ a or e | f | g | h
 Pattern Expression: a|b|c
STATUS: M68040--Running in monitor          Emulation trace halted

 The TAB key selects whether the pattern matches the values or not the values.
```

Analysis Trace Modify Form - Level 2

Start the analyzer and the program:

```
Analysis Begin
Processor Go Pc
```

Now you can run your program. When you list the trace, the analyzer will have triggered on the first access to any of the three addresses: 100h, 200h, or 300h.

To define a range

1 Select **A**nalysis **T**race **M**odify.

2 Use the arrow keys to move to the qualifier you want to change and enter an expression using the range expression (**r** or **!r**). (See “To define a qualifier” for more details.) Press **<Enter>**.

3 Now you are in the pattern modification screen. Use the arrow keys to move to the label field next to “Range (r) Label.” **Tab** to select **addr** (address) or **data** (data) as the group of processor lines for which you want to define a range. Press **<Enter>** to move to the next field.

4 Now you can specify the lower bound of the range. If you selected addr, you can enter an address expression, using either numeric expressions, symbols, or a combination thereof. If you selected data, you can enter data expressions using numeric expressions. Press **<Enter>**.

5 Specify the upper bound of the range, using the same rules as in step 4.

6 When you finish, press **<End>** to move to the last field in the form. Press **<Enter>** to accept your changes and return to the sequencer definition screen. You can exit that screen by pressing **<End>**, then **<Enter>**.

If you want to discard your changes, press **<Esc>**.

The range qualifier **r** can be used in analyzer storage and complex branch qualifiers. For example, you might have a lookup table in your program, and want to record accesses to that table in the trace list. You can define the range qualifier as the lower and upper boundaries of the lookup table.

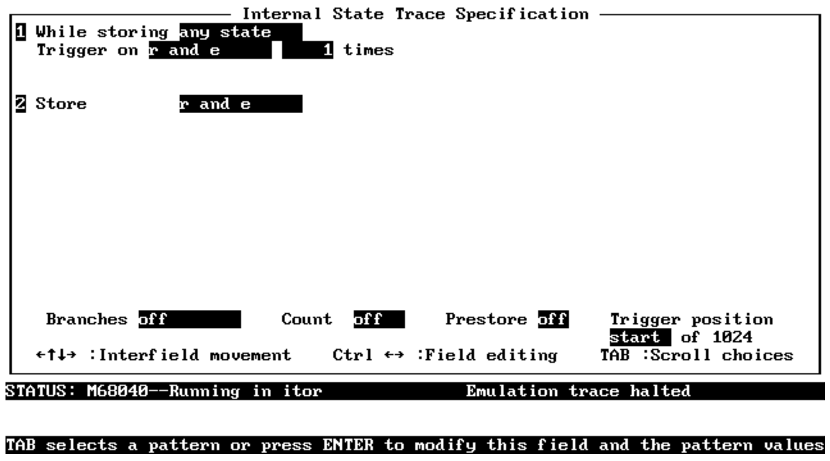
Examples

Suppose you want to trigger on reads from a data area (addresses 5000h through 5100) and store only those reads.

Use **A**nalysis **T**race **F**ormat to set an **addr** and **data** label to **hex**. Set up the analyzer as shown.

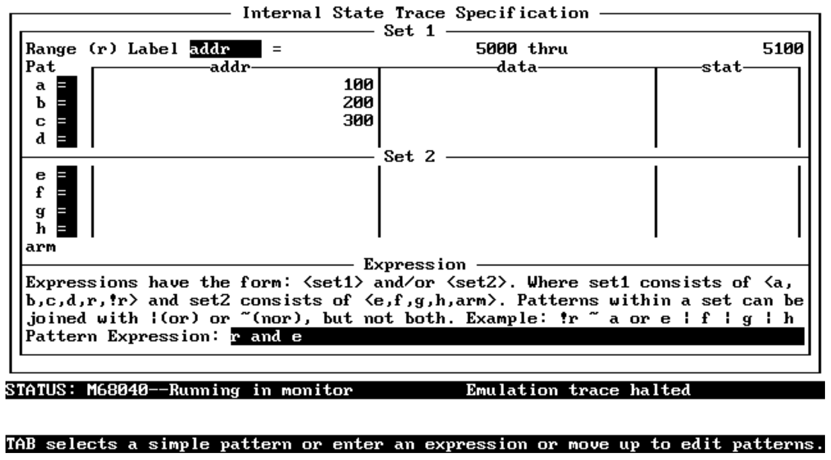
`**A**nalysis **T**race **M**odify`

Modify the screens as shown:



```
                    Internal State Trace Specification
1 While storing any state
  Trigger on r and e          1 times

2 Store         r and e






   Branches off           Count  off      Prestore off    Trigger position
                                                          start  of 1024
  ←↑↓→ :Interfield movement     Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: M68040--Running in itor               Emulation trace halted

TAB selects a pattern or press ENTER to modify this field and the pattern values
```

Analysis Trace Modify Form - Level 1



```
─────────────── Internal State Trace Specification ───────────────
──────────────────────────── Set 1 ───────────────────────────────
Range (r) Label addr     =                 5000 thru          5100
Pat   ───────addr──────────┬──────────data──────────┬────stat────
 a =                    100│                        │
 b =                    200│                        │
 c =                    300│                        │
 d =                       │                        │
──────────────────────────── Set 2 ───────────────────────────────
 e =                       │                        │
 f =                       │                        │
 g =                       │                        │
 h =                       │                        │
arm
────────────────────────── Expression ────────────────────────────
Expressions have the form: <set1> and/or <set2>. Where set1 consists of <a,
b,c,d,r,!r> and set2 consists of <e,f,g,h,arm>. Patterns within a set can be
joined with |(or) or ~(nor), but not both. Example: !r ~ a or e | f | g | h
Pattern Expression: r and e

STATUS: M68040--Running in monitor          Emulation trace halted

TAB selects a simple pattern or enter an expression or move up to edit patterns.
```

Analysis Trace Modify Form - Level 2

Start the analyzer and the program.

```
Analysis Begin
Processor Go PC
```

When your program has run long enough, or you have taken steps to cause the reads to occur:

Display the trace

```
Analysis Display
```

To set the storage qualifier

1 Select **A**nalysis **T**race **M**odify.

2 Enter a qualifier in the qualifier field next to the “While Storing” or “Store” label for each sequence term. (See “To define a qualifier” if you need more information.)

3 Press **<End>**, then **<Enter>**, to save your changes and exit the trace specification form. Press **<Esc>** if you want to discard your changes and exit the trace specification form.

There are eight storage qualifiers, one for each sequence term. This allows you to store only the states of interest at each level of the sequence, which uses the trace memory more efficiently and makes the trace display easier to read.

Example

Suppose you want to trace execution in a program, but only after a sequence of events have occurred in a specific order. In this example, the analyzer will wait to find the symbol "main", and then the symbol "interrupt_sim", and then the symbol "gen_ascii_data" in the ecs demo program. The analyzer will capture the occurrences of each of these symbols (addresses), and then it will store all of the program activity that follows. This type of measurement is useful when you are analyzing a program that may take several paths, and you want to capture activity only after the program takes one particular path.

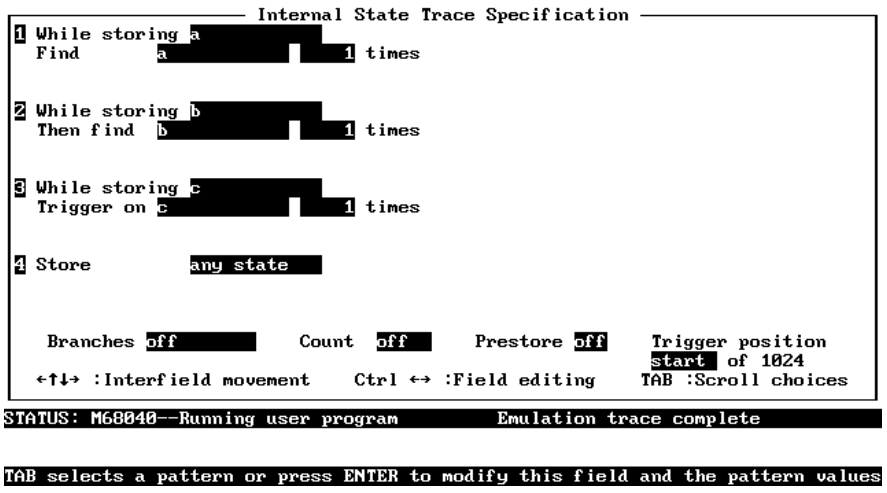
Start this example with the emulation processor reset.

```
Processor Reset Hold
```

Set up the trigger and pattern specifications for the measurement.

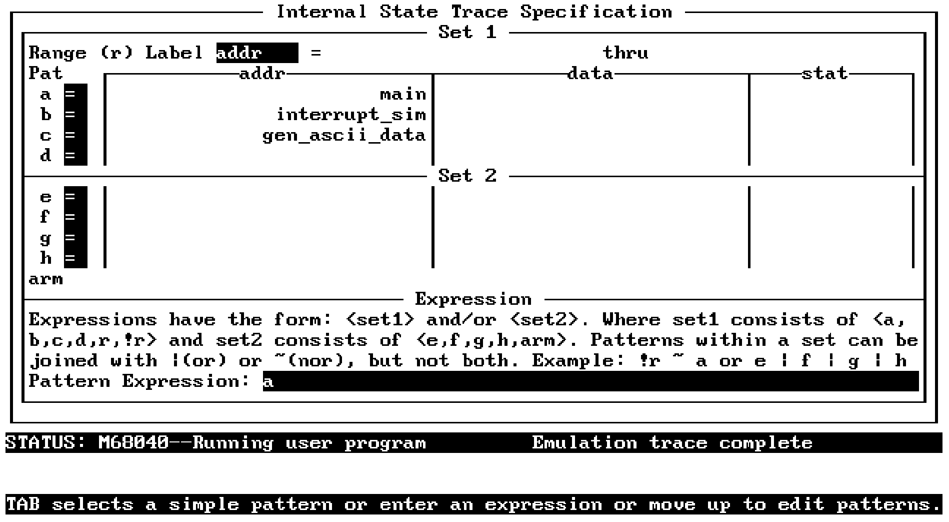
`Analysis Trace Modify`

The Analysis Trace Modify Form (Level 1) will be on screen. Modify the Level 1 and Level 2 forms as shown. Obtain the Level 2 form by pressing <Enter> while the cursor is in one of the "While storing" fields of the Level 1 form.



```
──────────── Internal State Trace Specification ────────────
1 While storing a
  Find        a                  1 times

2 While storing b
  Then find   b                  1 times

3 While storing c
  Trigger on  c                  1 times

4 Store          any state


   Branches off        Count  off     Prestore off    Trigger position
                                                      start  of 1024
  ←↑↓→ :Interfield movement    Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: M68040--Running user program         Emulation trace complete

TAB selects a pattern or press ENTER to modify this field and the pattern values
```

Analysis Trace Modify Form - Level 1



```
                  Internal State Trace Specification
                                          Set 1
Range (r) Label addr    =                      thru
Pat          addr                   data                 stat
 a =                main
 b =        interrupt_sim
 c =       gen_ascii_data
 d =
                                          Set 2
 e =
 f =
 g =
 h =
arm
                                       Expression
Expressions have the form: <set1> and/or <set2>. Where set1 consists of <a,
b,c,d,r,!r> and set2 consists of <e,f,g,h,arm>. Patterns within a set can be
joined with !(or) or ~(nor), but not both. Example: !r ~ a or e ! f ! g ! h
Pattern Expression: a

STATUS: M68040--Running user program         Emulation trace complete

TAB selects a simple pattern or enter an expression or move up to edit patterns.
```

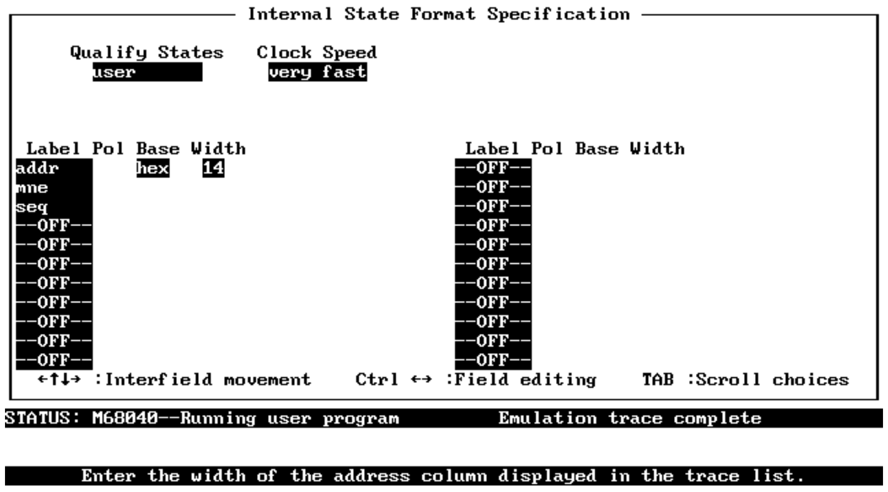
Analysis Trace Modify Form - Level 2

When the Level 2 form is set up as shown above, press <End> and then <Enter> to save your changes and return to the Level 1 form.

When the Level 1 form is set up as shown on the preceding page, press <End> and then <Enter> to save your changes and return to the display of the MC68040 PC Interface windows.

Set up the trace display format.

`Analysis Format`



```
─────────────── Internal State Format Specification ───────────────

     Qualify States   Clock Speed
       user             very fast


 Label Pol Base Width                    Label Pol Base Width
addr       hex   14                     --OFF--
mne                                     --OFF--
seq                                     --OFF--
--OFF--                                 --OFF--
--OFF--                                 --OFF--
--OFF--                                 --OFF--
--OFF--                                 --OFF--
--OFF--                                 --OFF--
--OFF--                                 --OFF--
--OFF--                                 --OFF--
--OFF--                                 --OFF--
  ←↑↓→ :Interfield movement    Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: M68040--Running user program         Emulation trace complete

       Enter the width of the address column displayed in the trace list.
```

Analysis Format Form

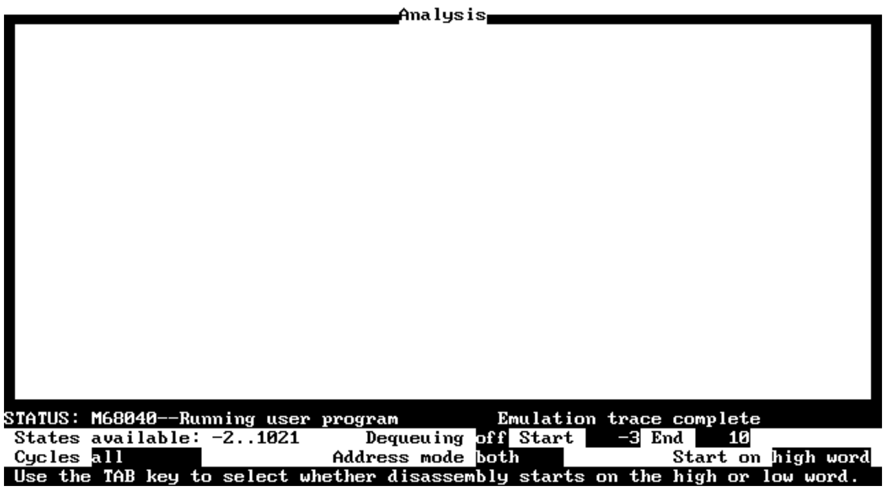
The display width of 14 was selected to allow the full symbol names to be shown in the address column of the trace list.

Start the analyzer first, and then start the demo program.

```
Analysis Begin
Processor Go Address ENTRY
```

The trace will be completed. Then you can view the tracelist. Format the tracelist as follows:

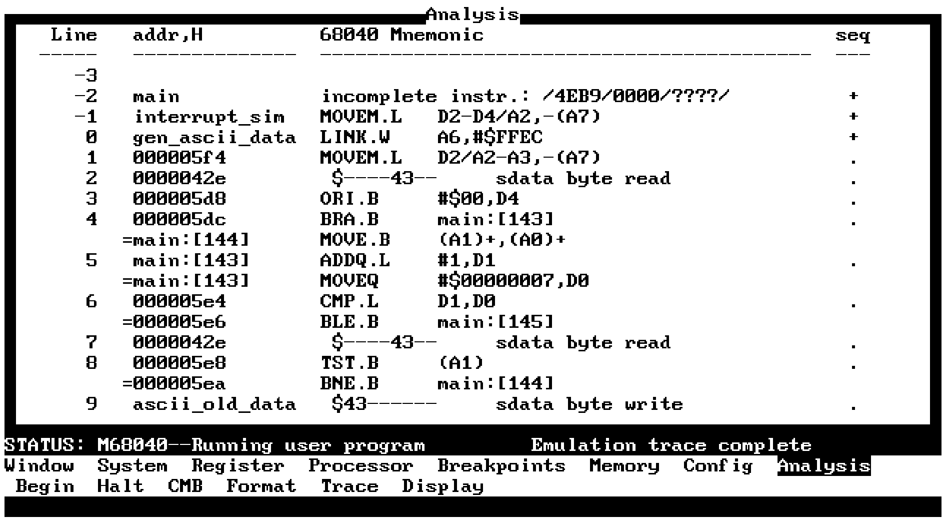
```
Analysis Trace Display
```



```
Analysis

STATUS: M68040--Running user program         Emulation trace complete
 States available: -2..1021      Dequeuing off Start    -3 End    10
 Cycles all               Address mode both            Start on high word
 Use the TAB key to select whether disassembly starts on the high or low word.
```

Analysis Display Trace Form

The above form uses the default analyzer display format, except that the trace list is started on line -3 and ended on line 10. Make these selections in the form and press <Enter>.



```
                                  Analysis
  Line    addr,H          68040 Mnemonic                                    seq
  -----   --------------  ------------------------------------------       ---
     -3
     -2   main            incomplete instr.: /4EB9/0000/????/                +
     -1    interrupt_sim  MOVEM.L   D2-D4/A2,-(A7)                           +
      0   gen_ascii_data  LINK.W    A6,#$FFEC                                +
      1   000005f4        MOVEM.L   D2/A2-A3,-(A7)                           .
      2   0000042e         $----43--     sdata byte read                     .
      3   000005d8        ORI.B     #$00,D4                                  .
      4   000005dc        BRA.B     main:[143]                               .
         =main:[144]      MOVE.B    (A1)+,(A0)+
      5   main:[143]      ADDQ.L    #1,D1                                    .
         =main:[143]      MOVEQ     #$00000007,D0
      6   000005e4        CMP.L     D1,D0                                    .
         =000005e6        BLE.B     main:[145]
      7   0000042e         $----43--     sdata byte read                     .
      8   000005e8        TST.B     (A1)                                     .
         =000005ea        BNE.B     main:[144]
      9   ascii_old_data   $43------     sdata byte write                    .

STATUS: M68040--Running user program         Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Begin  Halt  CMB  Format  Trace  Display
```

Analysis Display Screen

The analysis display screen shows that the first state that was captured was the symbol "main". The second state that was captured was the symbol "interrupt_sim". Trigger was recognized on symbol "gen_ascii_data", and after that, all states were stored. This has given a trace of activity that occurs only after a specific series of events have been executed. The trace memory only contains the states of interest at each step in the sequence.

To define a primary branch term

1 Select **A**nalysis **T**race **M**odify.

2 For each term that you want to modify, enter a state qualifier in the qualifier field next to the “Find” or “Then find” labels. (See “To define a qualifier” if you need more information.) Press **<Enter>** if you need to define patterns, or use the arrow keys to move to the “times” field.

3 Now the cursor is in the field next to the word “times.” This field sets the occurrence count. Type in a number between 1 and 65535 that specifies how many times the qualifier must be found before the sequencer advances to the next state.

4 Press **<End>**, then **<Enter>**, to save your changes and exit the trace specification form. Press **<Esc>** if you want to discard your changes and exit the trace specification form.

The primary branch qualifier defines the main path from a given sequencer term to another term (or the same term). If both the primary and secondary branch qualifiers are satisfied simultaneously, the primary branch is taken.

Usually, you’ll use the primary branch qualifiers to define a sequence of states that must be satisfied to reach the trigger condition.

Example

See the example under "To set the storage qualifier" for an example use of the primary branch qualifier.

To define a global restart term

1 Select **A**nalysis **T**race **M**odify.

2 Use the arrow keys to move the cursor to the field next to the label “Branches.” **Tab** to select **restart on** if you want a global restart, or **off** if you want to disable the global restart. Press **<Enter>**.

3 If you selected **restart on** in step 2, enter a state qualifier in the qualifier field below the “Branches” field. (See “To define a qualifier” if you need more information.)

4 Press **<End>**, then **<Enter>**, to save your changes and exit the trace specification form. Press **<Esc>** if you want to discard your changes and exit the trace specification form.

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.

You can also have the analyzer take a unique secondary branch for each sequencer level. See “To define a secondary branch term.”

Example

Suppose you want to trace execution in a program, but only if it executes abnormally. In this example, the analyzer is set up to find the symbol "main", and then the symbol "interrupt_sim", and then trigger when it finds the symbol "gen_ascii_data" in the ecs demo program. In the normal sequence of events, the system is updated (symbol "update_system" occurs) before the "interrupt_sim" symbol occurs. You only want to trace activity if "interrupt_sim" and then "gen_ascii_data" occur without the system being updated.

The analyzer will capture the occurrences of each of the above symbols, but will restart its search from the beginning of the sequence (symbol "main") each time it finds "update_system". The trace will only be triggered and completed (with storage of all activity) if "gen_ascii_data" is found during a trace of the sequence that does not include "update_system".

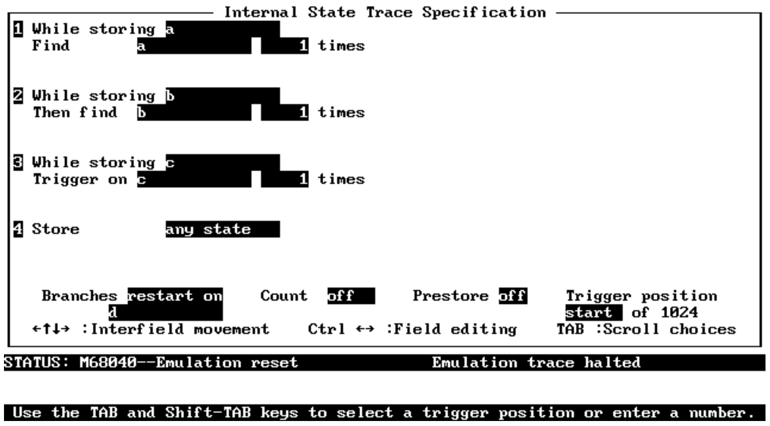
This type of measurement is useful when you are analyzing a program that takes a normal path, but occasionally fails to make a required call. You want to capture activity only if the program fails to make that required call.

Load the demo program and transfer symbol data to the emulator (see Chapter 1 for instructions)

Set up the trigger and pattern specifications for the measurement:

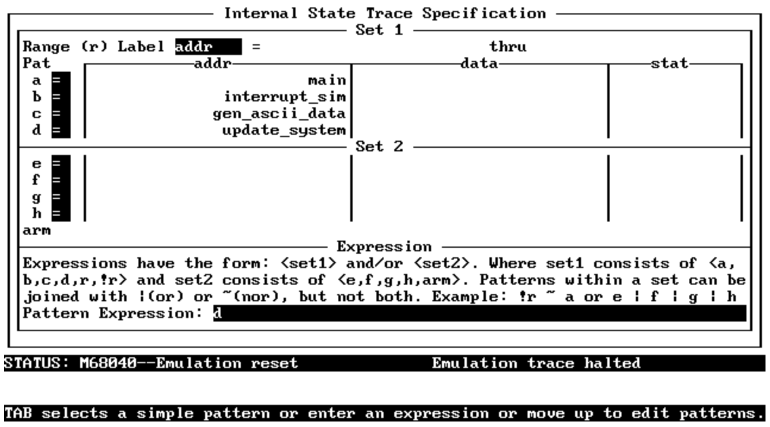
```
Analysis Trace Modify
```

The Analysis Trace Modify Form (Level 1) will be on screen. Modify the Level 1 and Level 2 forms as shown. Obtain the Level 2 form by pressing <Enter> while the cursor is in one of the "While storing" fields of the Level 1 form.



```
─────────────── Internal State Trace Specification ───────────────
1 While storing a
  Find       a                 1 times

2 While storing b
  Then find  b                 1 times

3 While storing c
  Trigger on c                 1 times

4 Store          any state

   Branches restart on    Count  off     Prestore off    Trigger position
            d                                            start   of 1024
  ←↑↓→ :Interfield movement    Ctrl ↔ :Field editing    TAB :Scroll choices
STATUS: M68040--Emulation reset             Emulation trace halted

 Use the TAB and Shift-TAB keys to select a trigger position or enter a number.
```

Analysis Trace Modify Form - Level 1



```
─────────────── Internal State Trace Specification ───────────────
──────────────────────────────── Set 1 ────────────────────────────
Range (r) Label addr    =                    thru
Pat    ┌──────────addr──────────┬──────────data──────────┬──────stat──────┐
 a =   │                   main │                        │                │
 b =   │          interrupt_sim │                        │                │
 c =   │         gen_ascii_data │                        │                │
 d =   │          update_system │                        │                │
──────────────────────────────── Set 2 ────────────────────────────
 e =   │                        │                        │                │
 f =   │                        │                        │                │
 g =   │                        │                        │                │
 h =   │                        │                        │                │
arm
────────────────────────────── Expression ─────────────────────────
Expressions have the form: <set1> and/or <set2>. Where set1 consists of <a,
b,c,d,r,!r> and set2 consists of <e,f,g,h,arm>. Patterns within a set can be
joined with |(or) or ~(nor), but not both. Example: !r ~ a or e | f | g | h
Pattern Expression: d
STATUS: M68040--Emulation reset             Emulation trace halted

TAB selects a simple pattern or enter an expression or move up to edit patterns.
```

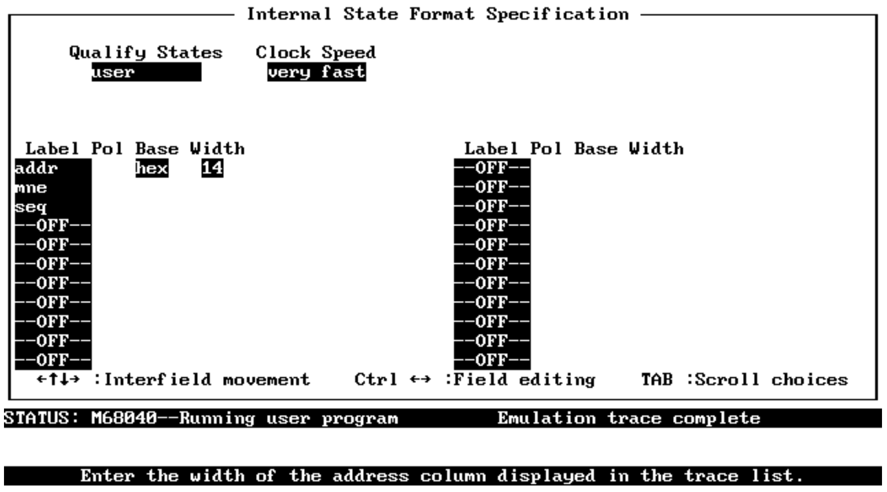
Analysis Trace Modify Form - Level 2

When the Level 2 form is set up as shown above, press <End> and then <Enter> to save your changes and return to the Level 1 form.

When the Level 1 form is set up as shown above, press <End> and then <Enter> to save your changes and return to the display of the MC68040 PC Interface windows.

Set up the trace display format.

`Analysis Format`



```
─────────────── Internal State Format Specification ───────────────

     Qualify States   Clock Speed
       user             very fast


 Label Pol Base Width                        Label Pol Base Width
addr        hex   14                        --OFF--
mne                                         --OFF--
seq                                         --OFF--
--OFF--                                     --OFF--
--OFF--                                     --OFF--
--OFF--                                     --OFF--
--OFF--                                     --OFF--
--OFF--                                     --OFF--
--OFF--                                     --OFF--
--OFF--                                     --OFF--
--OFF--                                     --OFF--
  ←↑↓→ :Interfield movement    Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: M68040--Running user program         Emulation trace complete

       Enter the width of the address column displayed in the trace list.
```

Analysis Format Form

The display width of 14 was selected to allow the full symbol names to be shown in the address column of the trace list.

Start the analyzer first, and then start the demo program.

```
Analysis Begin
Processor Go Address ENTRY
```

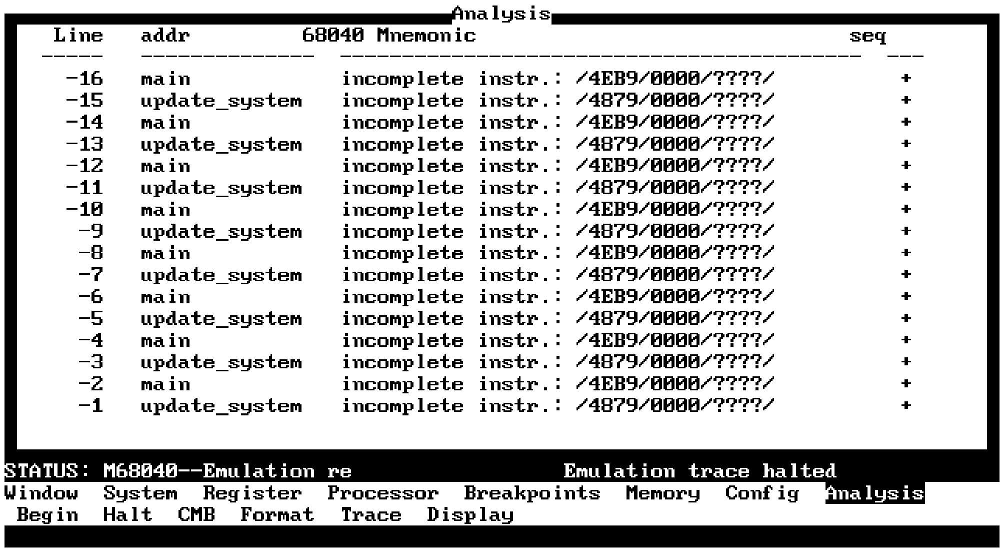
The trace will not be completed. You will have to halt it.

```
Analysis Trace Halt
```

Then you can view the tracelist:

```
Analysis Trace Display
```

Accept the default format of the trace list offered in the Analysis Display Trace Form. Simply press <End> and then <Enter>.



```
                                  Analysis
  Line   addr         68040 Mnemonic                                  seq
 -----   --------------  ----------------------------------------  ---
   -16   main            incomplete instr.: /4EB9/0000/????/          +
   -15   update_system   incomplete instr.: /4879/0000/????/          +
   -14   main            incomplete instr.: /4EB9/0000/????/          +
   -13   update_system   incomplete instr.: /4879/0000/????/          +
   -12   main            incomplete instr.: /4EB9/0000/????/          +
   -11   update_system   incomplete instr.: /4879/0000/????/          +
   -10   main            incomplete instr.: /4EB9/0000/????/          +
    -9   update_system   incomplete instr.: /4879/0000/????/          +
    -8   main            incomplete instr.: /4EB9/0000/????/          +
    -7   update_system   incomplete instr.: /4879/0000/????/          +
    -6   main            incomplete instr.: /4EB9/0000/????/          +
    -5   update_system   incomplete instr.: /4879/0000/????/          +
    -4   main            incomplete instr.: /4EB9/0000/????/          +
    -3   update_system   incomplete instr.: /4879/0000/????/          +
    -2   main            incomplete instr.: /4EB9/0000/????/          +
    -1   update_system   incomplete instr.: /4879/0000/????/          +

STATUS: M68040--Emulation re               Emulation trace halted
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Begin  Halt  CMB  Format  Trace  Display
```

Analysis Display Screen

Notice that the sequence is never completed because after "main", "update_system" is found. When "update_system" occurs, the sequence is restarted. If the sequence were ever completed without "update_system" occurring, a trace would be triggered on "gen_ascii_data" and the trace memory would store the activity that followed.

To define a secondary branch term

1. Select **A**nalysis **T**race **M**odify.

2. Use the arrow keys to move the cursor to the field next to the label “Branches.” **Tab** to select **per level** if you want to specify individual secondary branch qualifiers, or **off** if you want to disable the secondary branch qualifiers. Press **<Enter>**.

3. If you selected **per level** in step 2, enter a state qualifier in the qualifier field next to the “Else on” label for each sequencer term. (See “To define a qualifier” if you need more information.) Press <Enter> to define patterns or use the arrow keys to move to the “goto level” field. Type in a sequence level (only the sequence levels currently displayed are valid).

4. Press **<End>**, then **<Enter>**, to save your changes and exit the trace specification form. Press **<Esc>** if you want to discard your changes and exit the trace specification form.

The secondary branch qualifier defines an alternate path from a given sequencer term to another term (or the same term). If both the primary and secondary branch qualifiers are satisfied simultaneously, the primary branch is taken. You can use the secondary branch as an alternate path to the trigger if more than one sequence of conditions is acceptable.

Because the secondary branch qualifier is unique for each sequence term, it is more flexible than a global restart qualifier. You can have each secondary branch qualifier cause a transition to different term numbers. See the examples under “To define a primary branch term” and “To set the storage qualifier.”

When you make all the secondary branch qualifiers identical, you have a global restart qualifier. See “To specify a global restart qualifier.”

Setting Analyzer Clocks

The HP 64700 Series emulators allow up to five clock signals for emulation and external analysis. These are J, K, L, M, and N clocks. The HP 64783 emulator generates the L clock to drive the emulation analyzer. The other clocks are not used.

The PC Interface provides fields in the **A**nalysis **F**ormat form to configure the clock signals. You can use the **A**nalysis **F**ormat command to qualify capture of either target program execution (called user), which includes execution of the foreground monitor, or execution of the background monitor. You also use this form to specify the maximum data rate that the analyzer will see, which affects the state/time counter.

To trace user/background code execution

1 Select **A**nalysis **F**ormat.

2 Use the arrow keys to move the cursor to the field next to the label “Qualify.” **Tab** to select **user** if you want the analyzer to record only your target program (user) states. Select **background** if you want the analyzer to record only background monitor program states. Select **all** to have the analyzer record execution of your target program and background monitor program states.

3 Press **<End>**, then **<Enter>**, to save your changes and exit the format form. Press **<Esc>** if you want to discard your changes and exit the format form.

The emulation-bus analyzer has built-in qualifiers that allow you to select whether the analyzer captures execution of the target program (which includes execution of the foreground monitor), background monitor code, or both. Usually, you’ll want to trace only your target program. If you’re trying to solve a problem with emulator and target system interaction, you may want to trace both the target program and background monitor code.

To configure the analyzer clock

1. Select **A**nalysis **F**ormat.
2. Use the arrow keys to move the cursor to the field next to the label “Clock Speed.” **Tab** to select **slow** if the analyzer data rate is less than or equal to 16 MHz. Select **fast** if the analyzer data rate is between 16 and 20 MHz. Select **very fast** if the analyzer data rate is greater than 20 MHz.
3. Press **<End>**, then **<Enter>**, to save your changes and exit the format form. Press **<Esc>** if you want to discard your changes and exit the format form.

The MC68040 analyzer clock is set to **very fast** by default. The 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 setting for the “Clock Speed” field in the MC68040 emulator, 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. (See the chapter titled "Configuring the Emulator" for more information.) Then choose the clock speed setting according to the calculated clock rate. If no burst cycles are performed, the analyzer clock speed can be set **slow**.

The trace state and time count qualifiers are limited by the analyzer clock rate settings as follows:

Analyzer clock rate	**Clock Speed setting**	**Valid Count Qualifier options**
clock ≤ 16 MHz	slow	Count <state> Count time
clock ≤ 20 MHz	fast	Count <state>
clock ≥ 20 MHz	very fast	Count none

Example

Suppose you are running the MC68040 processor at 40 MHz. You have enabled a wait state for target memory because target memory requires one wait state for synchronous/burst accesses over 25 MHz. The resulting data rate is 20 MHz, so you modify the “Clock Speed” field in the **A**nalysis **F**ormat form to **fast**. You are limited to counting states in the trace specification.

Because the analyzer clock rate is between 16 and 20 MHz, you can choose to count states during the trace. However, you cannot choose to count time during the trace.

Using Other Analyzer Features

The analyzer has other features that can be used in all configurations to make trace measurements easier to interpret or add additional information.

- *Prestore* allows you to save specific trace states that are related to other events in your trace list. For example, you might want to save all the callers of a subroutine.
- *Count* qualifiers allow you to count states or time.

To define a prestore qualifier

1 Select **A**nalysis **T**race **M**odify.

2 Use the arrow keys to move the cursor to the field next to “Prestore” and press the **Tab** key to select **on**. (Select **off** if you want to disable a previous prestore specification, and skip to step 4.) Press **<Enter>**.

3 Type in a state qualifier expression in the field below “Prestore.” (See “To define a qualifier,” in this chapter.)

4 Press **<End>**, then **<Enter>** to save your changes and exit the trace specification form. Press **<Esc>** if you want to discard your changes and exit the trace specification form.

You use the prestore qualifier to save states that are related to other states specified by storage qualifiers.

To count states or time

1 Select **A**nalysis **T**race **M**odify.

2 Use the arrow keys to move the cursor to the field next to "Count." Press the **Tab** key to select **time** if you want to count time intervals, **state** if you want to count bus states, or **off** if you want to disable counting. Press **<Enter>**.

3 If you selected **state** in step 2, type in a state qualifier expression in the field below "Count." (See "To define a qualifier," in this chapter.)

4 Press **<End>**, then **<Enter>** to save your changes and exit the trace specification form. Press **<Esc>** if you want to discard your changes and exit the trace specification form.

The trace count qualifier can be used to measure time for each stored state or occurrence counts of a particular state. You can display these values either relative to the last stored state (relative mode) or relative to the trigger state (absolute mode). You change the display using the **A**nalysis **F**ormat command. Refer to "To change the trace format," earlier in this chapter.

The MC68040 emulator defaults to **off**. Refer to "To configure the analyzer clock" to see the counts that can be made at different analyzer clock speeds.

When you select **off**, the analyzer trace depth is increased from 512 states to 1024 states.

6

Making Coordinated Measurements

Use the Coordinated Measurement Bus to start and stop multiple emulators and analyzers

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.

Three signal lines are used to control interaction over the CMB.

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.

There are two lines internal to the emulator that are used for coordinated analyzer measurements. These are TRIG1 and TRIG2. The analyzer can drive or receive either of these signals. Also, the rear-panel BNC and the CMB TRIGGER signal can drive or receive either of these signals.

Several different commands and forms control these signals. By using these commands, you can make the following types of measurements:

- Start a program run or analyzer trace when the CMB EXECUTE signal is driven.
- Use either the BNC trigger or CMB TRIGGER to arm (and potentially trigger) the analyzer.
- Have the analyzer drive the BNC trigger or CMB TRIGGER to trigger other instruments or emulators.
- Break the emulator into the monitor when a BNC trigger, CMB TRIGGER or analyzer trigger occurs.

The commands used to make coordinated measurements are as follows:

Command	**Function**
Config **G**eneral	Sets drivers and receivers of internal trigger signals
Config **T**rigger	Configuration item enables/disables CMB interaction
Processor **C**MB **G**o	Sets run at CMB EXECUTE address
Analysis **C**MB **B**egin	Enables/disables trace on CMB EXECUTE
Processor **C**MB **E**xecute or **A**nalysis **C**MB **E**xecute	Starts a coordinated CMB measurement

This chapter shows some of the common measurements that you may want to make. By combining the above commands in different ways, you can make more complex measurements involving several test instruments. This can be useful for troubleshooting multiprocessor systems or problems where the emulator isn’t capable of making the whole measurement.

The MC68040 emulator supports CMB interaction whether it is configured to use a background monitor or a foreground monitor. To connect emulators using the CMB, see the *HP 64700 Series Card Cage Installation/Service Guide*.

To start a simultaneous program run on two emulators

1 Enable the CMB on each emulator (use the **C**onfig **G**eneral command to access the "Enable CMB interaction?" configuration item, then select **yes**).

2 Reset each emulator using the **P**rocessor **R**eset **H**old command.

3 Set the run address for the first emulator by selecting **P**rocessor **C**MB **G**o **<address> <Enter>**.

4 Set the run address for the second emulator by selecting **P**rocessor **C**MB **G**o **<address> <Enter>**.

5 Start program execution on both emulators by selecting **P**rocessor **C**MB **E**xecute on the emulator driving the CMB bus.

Before you do this procedure, both emulators must be connected via the CMB. To connect the CMB, see the *HP 64700 Series Card Cage Installation/Service Guide.*

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 use the **A**nalysis **C**MB **B**egin command to enable trace on execute for each emulator. When the EXECUTE signal is received, both emulators will begin running as specified by the **P**rocessor **C**MB **G**o command, and will start a trace according to the given trigger specification.

Example

Assume that you have two MC68040 emulators. The same program is loaded in each emulator. The first emulator is driving the CMB bus. The following example will start a simultaneous program run on both emulators.

Enable CMB interaction by selecting in each emulator:

```
Config General
```

and set the "Enable CMB Interaction?" configuration item to **yes**.

Select in each emulator:

```
Processor Reset Hold
```

Set the run address for the first emulator by selecting:

```
Processor CMB Go
```

and entering the start address of the program (example: 100h)

Now set the run address for the second emulator by selecting:

```
Processor CMB Go
```

and entering 100

Start program execution on both emulators by selecting the following in the first emulator:

```
Processor CMB Execute
```

To trigger one emulation-bus analyzer with another

1 Enable the CMB on each emulator (use the **C**onfig **G**eneral command to access the “Enable CMB interaction?” configuration item, then select **yes**).

2 Reset each emulator using the **P**rocessor **R**eset **H**old command.

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 emulator using the **A**nalysis **B**egin command.

6 Start a run on each emulator using the **P**rocessor **G**o command.

Before you do this procedure, both emulators must be connected via the CMB. To connect the CMB, see *HP 64700 Series Card Cage Installation/Service Guide.*

In the above procedure, 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.

Example

Assume you have two MC68040 emulators. The same program is loaded in each emulator. The following example will trigger the analyzers in both emulators.

Enable CMB interaction by selecting:

```
Config General
```

and set the “Enable CMB Interaction?” configuration item to **yes**.

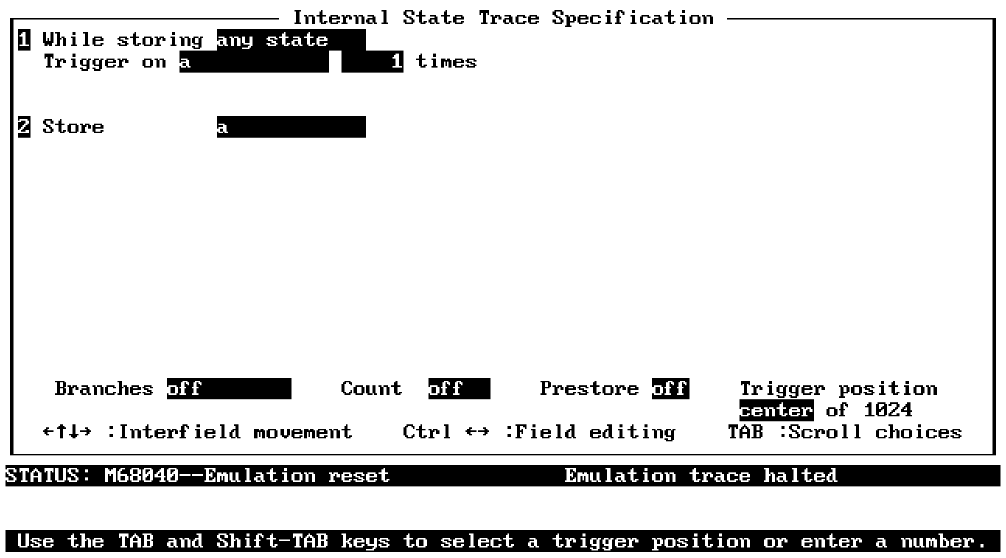
Select :

```
Analysis Trace Modify
```

Set the analyzer to trigger on the pattern:

```
addr=100 and stat=write
```

and set the trigger position to the center of the trace. See lower right corner of the following figure.



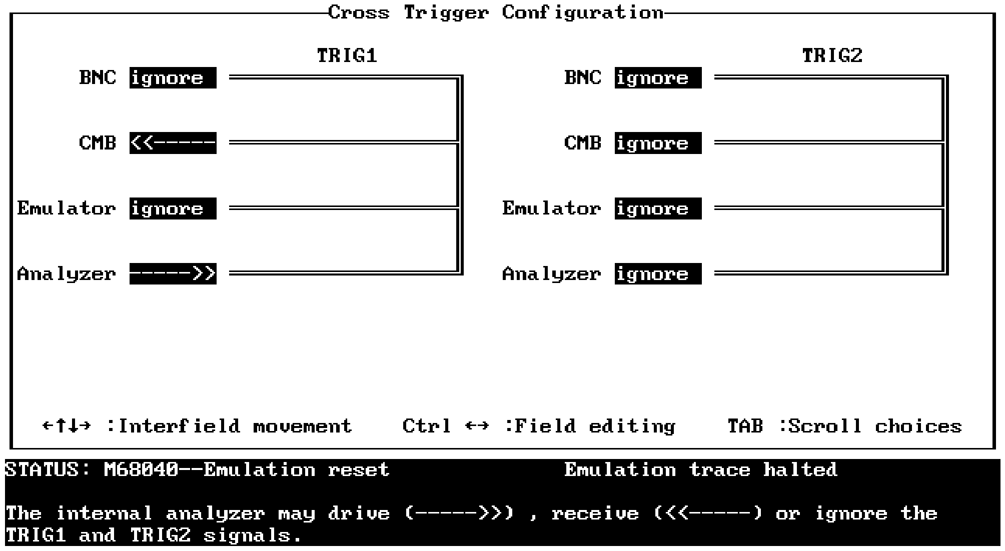
```
─────────────── Internal State Trace Specification ───────────────
1 While storing any state
  Trigger on a              1 times

2 Store         a










   Branches off        Count  off     Prestore off    Trigger position
                                                     center of 1024
  ←↑↓→ :Interfield movement    Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: M68040--Emulation reset              Emulation trace halted

 Use the TAB and Shift-TAB keys to select a trigger position or enter a number.
```

Select "Center" of Trace

Now set up the trigger signal configuration :

Config **T**rigger

Use the arrow keys to move to the analyzer and CMB fields for TRIG1, and use **<Tab>** to set the drivers and receivers as shown in the following display:


```
Cross Trigger Configuration

                              TRIG1                                        TRIG2
      BNC ignore                                     BNC ignore

      CMB <<-----                                    CMB ignore

 Emulator ignore                                Emulator ignore

 Analyzer ----->>                               Analyzer ignore

  ←↑↓→ :Interfield movement    Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: M68040--Emulation reset              Emulation trace halted

The internal analyzer may drive (----->>) , receive (<<-----) or ignore the
TRIG1 and TRIG2 signals.
```

First Emulation-bus Analyzer Trigger Configuration

Start the trace and break the emulator to the monitor:

```
Analysis Begin
Processor Break
```

Load the program in the second emulator. Then enable CMB interaction by selecting:

```
Config General
```

and set the “Enable CMB Interaction?” configuration item to **yes**.

Select:

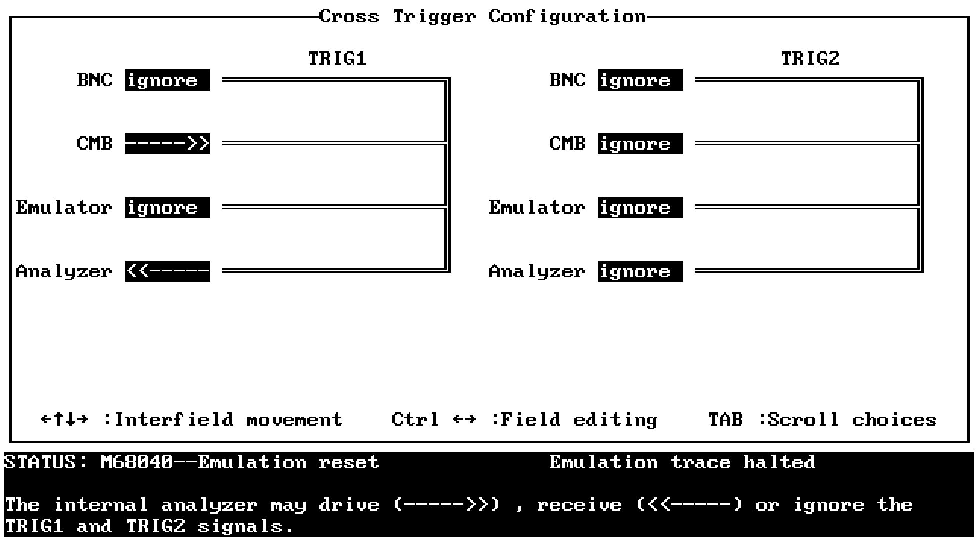
```
Analysis Trace Modify
```

Set the second emulation-bus analyzer to trigger on **arm**, and position the trigger in the center of the trace.

Now set up the second emulation-bus analyzer trigger signal:

```
Config Trigger
```

Use the arrow keys to move to the analyzer and CMB fields for TRIG1, and use **<Tab>** to set the drivers and receivers as shown in the following display:



```
Cross Trigger Configuration

                              TRIG1                                  TRIG2
      BNC ignore                                  BNC ignore

      CMB ----->>                                 CMB ignore

 Emulator ignore                             Emulator ignore

 Analyzer <<-----                            Analyzer ignore


  ←↑↓→ :Interfield movement    Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: M68040--Emulation reset              Emulation trace halted

The internal analyzer may drive (----->>) , receive (<<-----) or ignore the
TRIG1 and TRIG2 signals.
```

Second Emulation-bus Analyzer Trigger Configuration

Start the trace and break the emulator to the monitor:

```
Analysis Begin
Processor Break
```

Start the first emulator:

```
Processor Go Pc
```

On the second emulator, press **<Enter>**. Note that the second emulator is now running.

Enter a command for the program on the first emulator:

```
Memory Modify Bytes CMD_INPUT=z <Enter>
```

Display the trace on the first emulator:

Analysis **D**isplay **both <Enter> -1 <Enter> 14 <Enter>**

Display the trace on the second emulator:

Analysis **D**isplay **both <Enter> -1 <Enter> 14 <Enter>**

To break to the monitor on an analyzer trigger signal

1 Select **C**onfig **T**rigger.

2 Use the arrow keys to move to the field labeled "Emulator" under the TRIG1 signal. Use the **<Tab>** key to change this field to an arrow pointing toward "Emulator."

Use the arrow keys to move to the field labeled "Analyzer" under the TRIG1 signal. Use the **<Tab>** key to change this field to an arrow pointing away from "Analyzer."

3 Specify the trigger conditions for the trace. (See Chapter 5).

4 Select **A**nalysis **B**egin to start the trace.

5 Select **P**rocessor **G**o to start the program run.

The trigger signals and the analyzer trigger capabilities allow you to specify hardware 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 but before further program execution.

You can use a similar process to break to monitor when a BNC trigger or CMB trigger is received.

Example

The following example will cause execution to break to the monitor when your program reaches address 5000.

Select:

```
Config Trigger
```

Use the arrow keys to move to the "Emulator" field under the TRIG1 signal. **<Tab>** until this field displays an arrow pointing toward "Emulator."

Use the arrow keys to move to the "Analyzer" field under the TRIG1 signal. **<Tab>** until this field displays an arrow pointing away from "Analyzer."

Specify the trigger condition by selecting:

```
Analysis Trace Modify
```

and using the arrow keys to move to the "Trigger on" field. Type **a** and press **<Enter>**.

Use the arrow keys to move to the address field next to the **a=** label. Enter **5000** and press **<Enter>**.

Press **<End><Enter>** *twice* to save the trigger specification.

To start the trace select:

```
Analysis Begin
```

To start the program run select:

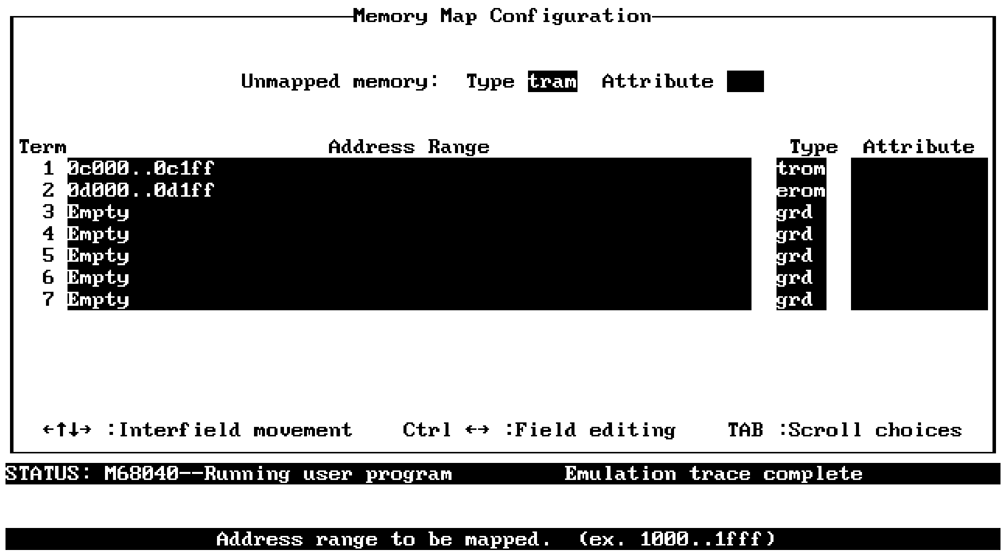
```
Processor Go
```

7

Configuring the Emulator

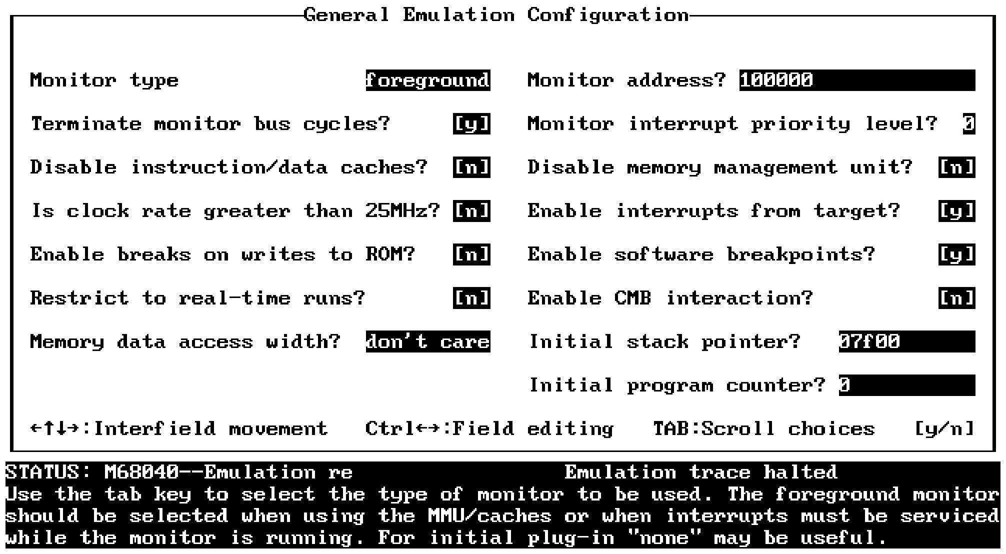
How to adapt the emulator to your system

Each target system differs in the way it uses memory and memory mapped I/O devices, and configures the processor. During system development, your needs for emulator resources may change as your target system design matures. You can allocate emulator resources using PC Interface commands. This resource allocation is called the *emulator configuration.* You change the emulator’s configuration with the Config Map Modify form and the Config General form.



```
                    Memory Map Configuration

          Unmapped memory:  Type tram  Attribute

Term                  Address Range                    Type  Attribute
  1 0c000..0c1ff                                       trom
  2 0d000..0d1ff                                       erom
  3 Empty                                              grd
  4 Empty                                              grd
  5 Empty                                              grd
  6 Empty                                              grd
  7 Empty                                              grd

 ←↑↓→ :Interfield movement    Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: M68040--Running user program         Emulation trace complete

                Address range to be mapped.  (ex. 1000..1fff)
```

The Config Map Modify Form



```
                    General Emulation Configuration

 Monitor type                 foreground  Monitor address? 100000

 Terminate monitor bus cycles?      [y]   Monitor interrupt priority level?  7

 Disable instruction/data caches?   [n]   Disable memory management unit?  [n]

 Is clock rate greater than 25MHz?  [n]   Enable interrupts from target?   [y]

 Enable breaks on writes to ROM?    [n]   Enable software breakpoints?     [y]

 Restrict to real-time runs?        [n]   Enable CMB interaction?          [n]

 Memory data access width?   don't care   Initial stack pointer?   07f00

                                          Initial program counter? 0

 ←↑↓→:Interfield movement   Ctrl←→:Field editing   TAB:Scroll choices   [y/n]

STATUS: M68040--Emulation re                 Emulation trace halted
Use the tab key to select the type of monitor to be used. The foreground monitor
should be selected when using the MMU/caches or when interrupts must be serviced
while the monitor is running. For initial plug-in "none" may be useful.
```

The Config General Form

Configuring and Mapping Memory

Each target system allocates memory and I/O as needed by the application. As the system design matures, memory locations and requirements may change. For example, the initial target system design may not have much memory. But a change in application definition might need more program code, requiring more memory. While the design is being changed, you can develop the program using emulation memory to simulate target system memory. Later on, you can use target system memory instead. You can even request that certain regions of emulator memory be treated as read-only (even though emulator memory can, in fact, be written to), causing a break to monitor on attempts to write to these memory regions. This is useful if you know that the specified memory range will eventually be implemented in target system ROM.

Since memory can be used in such a flexible way, you must specify to the emulator which addresses correspond to target system memory, and which correspond to emulator memory. That way, when the emulation processor attempts to read or write memory, the emulator knows whether it should respond to the request, or pass it on to the target system.

To ensure that timing of accesses to emulation memory are the same as accesses to target system memory, you can lock termination of emulation memory cycles and monitor bus cycles to the target system $\overline{\text{TA}}$ and $\overline{\text{TEA}}$ signals. If your target system termination signals are not available, you can allow termination of the emulation-access cycles to be done by signals within the emulator.

You specify how memory is going to be used by means of a *memory map.* A memory map consists of one or more *map terms.* Each map term indicates, for a given range of addresses, whether it is target memory or emulator memory.

To assign memory map terms

1 Select **C**onfig **M**ap **M**odify.

2 Use the arrow keys to move to one of the “Address Range” fields next to a term.

3 Type in an address range that has the lower and upper boundaries of the term you want (<lower>..<upper>). The addresses must be aligned on 256-byte boundaries (<lower> ends in 00 hex, and <upper> ends in ff hex).

4 Press **<Enter>** to move to the memory "Type” field for that term.

5 **Tab** to select a memory type as follows:

Type value	**Memory Assigned**
eram	Emulation RAM
erom	Emulation ROM
tram	Target System RAM
trom	Target System ROM
grd	Guarded memory

6 Press **<Enter>** to move to the “Attribute” field for that term.

7 **Tab** to select an attribute as follows:

dp to indicate that this block is to reside in the special 4-Kbyte block of dual-ported emulation memory on the probe.

lock indicates that accesses to emulation memory are terminated by target system $\overline{\text{TA}}$ and $\overline{\text{TEA}}$ signals instead of the internal termination signals of the emulator.

tci asserts the $\overline{\text{TCI}}$ line to the MC68040 for all addresses in this memory block, indicating that accesses to this block should not be cached.

(Combine multiple attributes by separating them with commas, for example: **dp,lock**.)

8 Repeat steps 2 through 7 for each address range you want to map (but no more than 7 address ranges).

9 Press **<End>**, then **<Enter>** to exit the map and save your changes. Press **<Esc>** to exit the map and discard your changes.

You need to specify the location and type of various memory regions used by your programs and your target system. The emulator needs this information to:

- Orient buffers for data transactions with emulation memory and the target system.
- Reserve emulation memory blocks.
- Set the type of memory blocks so that configuration items such as break on write to ROM will operate correctly.

The MC68040 emulator has seven map terms. Your address specifications must begin and end on 256-byte boundaries (256-byte resolution). To specify an address beginning on a 256-byte boundary, enter an address ending in 00. To specify an address ending on a 256-byte boundary, enter an address ending in ff. Because of the way the emulation memory system is designed, the amount of memory used by each map term corresponds to the nearest block size available, not the amount specified by the address range. For example, if a block size is 128 Kbytes, and you allocate 256 bytes of that block, all 128 Kbytes will be allocated, leaving 127.75 Kbytes as unusable memory.

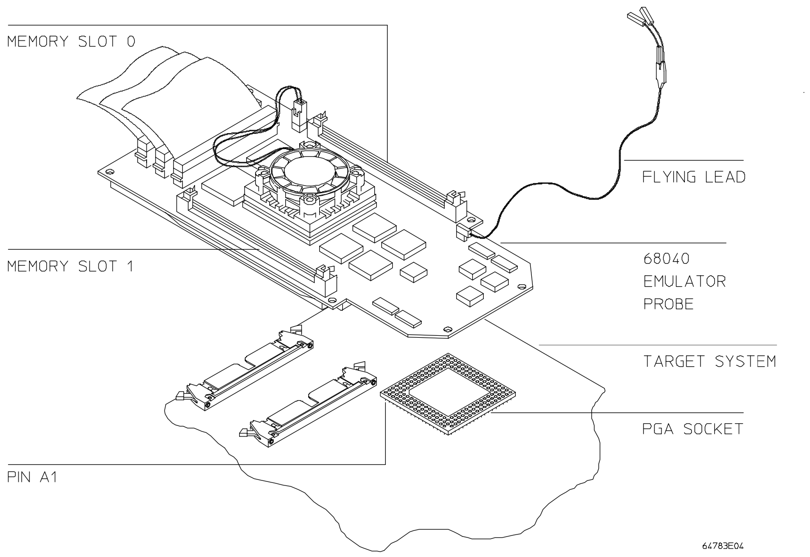
There is one 4-Kbyte block of dual-ported emulation memory on the emulator probe. (Dual-ported means that the emulation controller can access memory locations without interfering with program execution). This block can be mapped by specifying the **dp** attribute after the map address and memory type specification. If you use the foreground monitor, the emulator creates a map term reserving this block for the monitor code.

If you specify an address range less than 4 Kbytes with the **dp** attribute, all 4 Kbytes are 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 uses that memory to more closely match the requested address range to the block size.

There are also two memory sockets on the probe. This memory is not dual-ported; the monitor is used to read and write these locations when you display or modify memory. You can install 256-Kbyte, 1-Mbyte, or 4-Mbyte SRAMs in various configurations. The valid configurations are shown below.

Installation	**Memory slot 0****	**Memory slot 1****	**Blocks Available**
1	256K	256K	4-64K, 2-128K
2*	256K	1M	4-64K, 2-512K
3	1M	256K	4-256K, 2-128K
4	1M	1M	4-256K, 2-512K
5	256K	Empty	4-64K
6	1M	Empty	4-256K
7	4M	4M	4-1M, 2-2M
8	4M	1M	4-1M, 2-512K
9	4M	256K	4-1M, 2-128K
10	4M	Empty	4-1M

* Installation 2 is not recommended because it does not allocate blocks as well as Installation 3.
** Memory Slot 0 and Memory Slot 1 are marked on the probe board as BANK 0 and BANK 1. Their locations are also shown in illustrations in the Installation and Service Chapter of this manual.
Your selection of wait states may be affected if you install 4M memory modules. See "To enable one wait state" later in this chapter.



Emulator Probe Memory Slots

For each configuration, the “Blocks Available” indicate the minimum amount of memory that will be allocated if you specify a map term with that block size or less. If you need to use emulation memory, you should examine your target system design and install memory in the way that will maximize block usage. (See the examples at the end of this section.)

If you specify the **lock** attribute, the emulator waits for the target system $\overline{\text{TA}}$ or $\overline{\text{TEA}}$ signals to terminate an emulation memory cycle. This makes the bus cycle length identical to that of your target system, so that timing will be the same. If your target system does not return $\overline{\text{TA}}$ or $\overline{\text{TEA}}$ in the address range mapped to emulation memory, don’t use the **lock** attribute, because the system will hang while waiting for the target $\overline{\text{TA}}$ or $\overline{\text{TEA}}$ signals. (See “To interlock emulator and target termination signals for monitor cycles” for more information.)

If you don’t specify the **lock** attribute when you map a memory block, the target $\overline{\text{TA}}$ and $\overline{\text{TEA}}$ signals are ignored on accesses to that block.

If you specify the **tci** attribute, the $\overline{\text{TCI}}$ (transfer cache inhibit) line is asserted for accesses to that memory block. This prevents instructions or data from that memory block from being loaded into the processor cache memory. If you need to disable caching for all memory accesses, use the “Disable instruction/data caches?” configuration item. See “To disable the processor cache memory” in this chapter.

If you want to add a term that overlaps address ranges with an existing term, you must either redefine or delete the existing term.

If you change the monitor type or monitor base address, the memory map is reset. So, you should configure the emulator before you map memory. Otherwise, you may need to reenter the map definition.

Example

Suppose that you're using the emulator in-circuit, and that 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:

```
1c000..1c0ff tram
0..0ffff     trom
20000..20fff eram dp
```

Remember that you *must* use the background monitor if you want to use the dual-port emulation memory.

The relationship between memory ranges and block sizes of memory is easier to understand by looking at an example. Suppose you have Installation 1 from the table on the previous page. Then you enter the following map commands:

```
0..7fff eram
20000..3f000 eram
40000..4ffff eram
50000..500ff eram
```

If you haven't used the dual-port emulation RAM, the first map term that will fit is assigned to that memory. In this example, that is the last term that 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 other emulation memory, 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.

The mapper's resolution is independent of the block allocation. In the above example, if you set the "Unmapped memory: Type" to **grd** and your program accessed 8000h, the emulator would do a guarded memory break.

To assign the memory map default

1 Select **C**onfig **M**ap **M**odify to access the memory configuration.

2 Use the arrow keys (if necessary) to move to the field next to "Unmapped memory: Type."

3 **Tab** to select a memory type (and attribute, if desired) as follows:

Type, Attribute	**Memory assigned**
tram	Target system RAM
tram, tci	Target system RAM with tci enabled
trom	Target system ROM
trom, tci	Target system ROM with tci enabled
grd	Guarded memory space

4 Press **<End>**, then **<Enter>** to save your changes and exit the memory map. Press **<Esc>** to discard your changes and exit the memory map.

The unmapped memory type/attribute term specifies all address ranges not otherwise covered by existing memory map terms. This can save you time in memory mapping. If you selected **tram** or **trom**, you must also choose to include or not include the **tci** attribute.

- If you choose to include the **tci** attribute, no data that is sent to unmapped memory will be written into the caches.
- If you choose not to include the **tci** attribute, transactions that are sent to unmapped memory may also be loaded into the instruction and/or data caches.

Often you will want to be notified when the processor accesses a nonexistent memory location during a program run. Use the **grd** (guarded) memory type to do this. The emulator will break to monitor and display a message when a guarded memory access occurs.

Example

Suppose you're working in-circuit with a target system that has some peripherals and ROM, but no RAM. Eventually, this system will have RAM in all other address ranges. You might assign map terms for the peripherals and target ROM. Then set the unmapped memory type to **tram**.

To check the memory map

1 Select **C**onfig **M**ap **M**odify to access the memory map.

2 When you're finished viewing the map, you can press either **<End><Enter>** to exit (if you made changes and want to save them), or **<Esc>** (if you don't want to change the map).

To delete memory map terms

1 Select **C**onfig **M**ap **M**odify to access the memory map.

2 Use the arrow keys to move to the “Address Range” field of the term you want to delete.

3 Use the spacebar to wipe out the address range definition for that term. Then press **<Enter>**.

4 Repeat steps 2 and 3 for all terms you want to delete.

5 Press **<End>**, then **<Enter>** to exit the map and save your changes. Press **<Esc>** to exit the map and discard your changes.

If you delete map terms that are between other terms, the other terms are renumbered when you save the map. This eliminates gaps in the term numbering sequence.

To reset the memory map

- Select **C**onfig **M**ap **R**eset.

This removes all memory map terms and resets the map. Make sure that you want to reset the map beforehand. If you want to remove only one or two terms without resetting the map, see the previous section.

To enable one wait state

1 Select **C**onfig **G**eneral to access the emulator configuration.

2 Use the arrow keys to move to the field next to “Is clock rate greater than 25 MHz?”

3 Type **y** if your target system clock frequency is greater than 25 MHz. Type **n** if your target system clock frequency is at or below 25 MHz.

4 Regardless of your target system clock frequency, type **y** if you have installed any 4-Mbyte, 25-ns memory modules in BANK0 or BANK1 on the emulation probe.

5 Press **<End>**, then **<Enter>** to save your changes and exit the configuration. Press **<Esc>** to discard your changes and exit the configuration.

When the clock speed of BCLK is above 25 MHz, the emulator requires one wait state for all accesses to memory, including burst mode accesses. You add this wait state by selecting **y**. Without this wait state, emulator operation will be erratic. No wait states are required for memory accesses, including burst mode accesses, when the clock rate of BCLK is at or below 25 MHz, unless you have installed 4-Mbyte, 25-ns memory modules in BANK0 or BANK1 (or both) on the emulation probe. The 25-ns memory modules need the additional wait state during accesses by the emulation system.

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 accesses, 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.

To enable the memory management unit

- To turn on the MMU in the MC68040 emulation processor, gain access to the general configuration form, change the monitor type to foreground (if not foreground already), and change the Disable memory management unit? field to **n**.

Once enabled, the MMU of the MC68040 can be set up by the operating system to manage logical (virtual) memory in physical address space. The selection of a root pointer and the value in the translation control register determine how the MMU will manage memory. The MMU must be enabled by this configuration field before the operating system can establish those control values.

Your target system will enable the MMU during program execution by using the MDIS signal. The target system can disable the MMU even if it is enabled via the configuration question.

A foreground monitor must be used when the MMU of the MC68040 is enabled. If the background monitor is selected when you attempt to enable the MMU, a message will advise you to select the foreground monitor first.

Note Make sure the foreground monitor is mapped to memory space that has a 1:1 translation and is not write protected. Refer to the end of this chapter for instructions on how to map the foreground monitor to 1:1 address space when using the MMU.

Examples

To enable the MC68040 MMU so that the operating system can set it up to manage memory, perform the following:

Access the general configuration screen:

```
Config General
```

Use the arrow keys to move to the Monitor type: field and enter:

Tab (to select **foreground**)

Use the arrow keys to move to the Disable memory management unit? field and enter:

Tab (to select **n**)

To disable the MC68040 MMU, do the following:

Access the general configuration screen:

```
Config General
```

Use the arrow keys to move to the Disable memory management unit? field and enter:

Tab (to select **y**)

You can use the arrow keys to move to the Monitor type: field and select foreground, background, or none, as desired.

Using the Emulation Monitor

The emulation monitor is used to perform emulation functions such as display and modification of target system memory, emulation memory that is not dual-ported, and processor registers. You can choose either a foreground or background monitor, and the base address at which the monitor resides. (See the book *Concepts of Emulation and Analysis* supplied with your product manuals for more information on foreground and background monitors.)

If you initialize the emulator, then break to monitor, and then try to run the processor, the run will fail because the processor's stack pointer and program counter aren't initialized. A configuration item allows you to set these values for convenience so that the above sequence will work correctly.

The following table summarizes the implementation of the monitor configuration.

Background monitor	Foreground monitor
Monitor address, Terminate monitor bus cycles?, Monitor interrupt priority level—not available **Target system keep alive**—set address from which to periodically read a byte during background monitor operation	**Monitor address**—sets address block that will contain the monitor in dual-port memory **Terminate monitor bus cycles?**—interlocks target system cycle termination signals to terminate emulation accesses instead of using internal emulation cycle termination signals. **Monitor interrupt priority level**—allows lowering of interrupt mask to this level during foreground monitor execution **Target system keep alive**—not available

What is the background monitor?

When you select the background monitor, the monitor program overlays processor address space and doesn't use any processor memory resources. The MMU and processor caches are disabled, and dma cannot be performed. Also, when the emulator is executing a routine in the background monitor, target system interrupts are disabled (including level 7 interrupts). These conditions may not be tolerable to some target system designs.

If you're using the background monitor, you can set a "keep-alive address" from which the background monitor will periodically read a byte during monitor operation. See "To set the background monitor keep-alive address," later in this chapter.

What is the foreground monitor?

If you select the foreground monitor, you can choose a default monitor that is resident in the emulator, or you can design a custom foreground monitor that supports special target system needs.

If you use a foreground monitor, you must specify the base address where the monitor is loaded. This is done through the "Monitor address" configuration item, described later in this chapter. Since a foreground monitor must run in dual-ported memory, changing the monitor address automatically resets the memory map so that it has exactly one map term. This single map term maps the 4-Kbyte range of addresses, beginning with the monitor base address you specified, to the 4-Kbyte block of dual-ported memory in the emulator. You must then reenter any other map terms you need.

Interrupts can be enabled and configured during foreground execution, which may make the emulator more transparent in some applications. See "To set the foreground monitor interrupt priority."

A foreground monitor must be used when the MMU of the MC68040 is enabled, when the caches of the MC68040 are enabled, when your target system does dma activity, or when the target system performs bus arbitration.

To select the emulation monitor

1 Select **C**onfig **G**eneral to access the emulator configuration.

2 Use the arrow keys to move to the field to the right of “Monitor type.”

3 **Tab** to select **foreground** if you want to use a foreground monitor (either the default foreground monitor or a custom foreground monitor which you have built). Select **background** if you want to use the built-in background monitor. Select **none** if you want to use no monitor, see below.

4 Press **<End>**, then **<Enter>** to exit the configuration and save your changes. Press **<Esc>** to discard your changes and exit the configuration.

When you select the foreground monitor, the emulator maps the 4-Kbyte block of dual-port memory for exclusive use by the monitor. You can’t use any portion this 4-Kbyte block for any other purpose because doing so will destroy the currently loaded copy of the monitor.

When you select the foreground monitor, the memory map is reset because the map is affected. Always select the monitor type before entering your memory map. The processor is always reset when you change monitor types. The foreground monitor can only reside in emulation memory.

Both monitors use the trace exception vector (located at offset 24 in the vector table) to implement the **P**rocessor **S**tep command. Therefore, you may need to initialize this vector properly before using the step command. See the section “To step the processor” in the chapter titled "Using the Emulator" for more information.

There are two types of foreground monitors. One is resident in the emulator, and is automatically loaded whenever the processor exits the emulation reset state. If this monitor doesn’t meet your needs, you can modify the monitor source code (supplied with the emulator) and load it using the **M**emory **L**oad command. See “To load a program” in the chapter titled "Using the Emulator" for more information.

Selecting **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 the chapter on plugging the emulator into a target system). Sometimes the task of connecting an emulator to a target system is 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.

If you have trouble with emulation monitor functions, you can reload the monitor. Whenever the processor transitions out of the emulator reset state, the current monitor (background, default foreground, or custom foreground) is (re)loaded. When you load a configuration file, you will have to reload your custom foreground monitor.

More information on emulation monitors is given in the book *Concepts of Emulation and Analysis* that was supplied with your product manuals.

To set the monitor base address

1 Select **C**onfig **G**eneral to access the emulator configuration.

2 Use the arrow keys to move the cursor to the field next to “Monitor address.”

3 Type in a hexadecimal address on a 4-Kbyte boundary (XXXXX000h).

4 Press **<End>**, then **<Enter>** to exit the configuration and save your changes. Press **<Esc>** to discard your changes and exit the configuration.

As indicated earlier in this chapter (see “Emulation Monitor”), when you use a background monitor, this configuration item is not available. If you use a foreground monitor, changing this configuration item automatically resets the memory map, and creates a single map term which maps the monitor’s address range to the 4-Kbyte block of dual-ported memory. You cannot delete or alter this map term by using the **C**onfig **M**ap **M**odify command. Instead, you must change the monitor configuration by using the “Monitor type,” “Monitor address,” and “Terminate monitor bus cycles?” configuration items.

If the memory management feature of the MC68040 emulator is enabled, be sure the foreground monitor is mapped in a range 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 interlock emulator and target cycle termination signals for monitor cycles

1 Select **C**onfig **G**eneral to access the emulator configuration.

2 Use the arrow keys to move the cursor to the field to the right of the question “Terminate monitor bus cycles?”

3 Type **n** if you want to interlock emulator and target system cycle termination signals for monitor accesses. Type **y** if you want to terminate monitor accesses with only the emulator-generated signals.

4 Press **<End>**, then **<Enter>** to exit the configuration and save your changes. Press **<Esc>** to exit the configuration and discard your changes.

When you enable interlocking, emulation monitor cycles are terminated by the target system TA or TEA signals. Otherwise, the emulator-generated signals terminate the cycles.

This configuration item is available to the foreground monitor but not the background monitor. This configuration item affects the map term defined for the foreground monitor.

If you enable the interlock, and the foreground monitor is in an address range where the target system does not return TA or TEA, the emulator will stop functioning. Use the **P**rocessor **R**eset **H**old command to reset the processor. Then disable the interlock.

If you disable the interlock, the target $\overline{\text{TA}}$ and $\overline{\text{TEA}}$ signals will be ignored during monitor accesses. This is useful in ranges where the TA and/or TEA 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 you do not enable the interlock in ranges where the target system supplies TA and/or TEA.

Bus cycles will be visible to the target system during foreground monitor operation. If interlocking is disabled, these cycles may cause erratic system operation if the target system is not expecting them.

To set foreground monitor interrupt priority

1 Select **C**onfig **G**eneral to access the emulator configuration.

2 Make sure that the field next to “Monitor type?” says **foreground**. If not, choose the monitor type first. (You cannot set monitor interrupt priority with the background monitor.)

3 Use the arrow keys to move to the field next to the text “Monitor interrupt priority level.”

4 Type in a priority level in the range 0..7.

5 Press **<End>**, then **<Enter>** to exit the configuration and save your changes. Press **<Esc>** to exit the configuration and discard your changes.

During background monitor execution, interrupts are always disabled. This may cause problems for some target systems, especially those for real-time control where interrupt servicing must be done immediately.

To solve this problem, you can select the foreground monitor. Then set the interrupt priority level to one that allows your target system to function correctly, yet avoids excessive interrupt processing.

At monitor entry, if the processor’s interrupt priority level was greater than the value set by this configuration item, the monitor uses the previous priority level. Otherwise, the priority is lowered to the level you specify here.

The foreground monitor only lowers the interrupt priority to the level you specify when it is not executing critical code. When it is executing critical code, such as monitor entry, all interrupts are disabled.

Target system interrupts are blocked if you set disable target system interrupts. See “To enable target system interrupts.”

The emulator is reset when you change the setting of this configuration item.

Example

Suppose your target system has a disk device driver that uses interrupt level 5, and the service routine must be run to prevent disk drive damage. To allow interrupts of higher priority than level 4 to be serviced during foreground monitor execution, set this configuration item to **4**.

To set the background monitor keep-alive address

1 Select **C**onfig **G**eneral to access the emulator configuration.

2 Make sure that the field next to "Monitor type?" says **background**. If not, choose the monitor type first. (You cannot set the monitor keep-alive address with the foreground monitor.)

3 Use the arrow keys to move to the field next to the text "Target system keep alive."

4 To enable the target system keep-alive function, type in a hex address with optional function code. To disable the function, type **disabled**.

5 Press **<End>**, then **<Enter>** to exit the configuration and save your changes. Press **<Esc>** to exit the configuration and discard your changes.

Some target systems need to have memory locations read periodically in order to work properly. For example, your target system may have a watchdog timer that will time out if a specific address isn't read periodically.

In this situation, you set the "Target system keep alive" configuration item to the address that must be accessed. Then, when the emulator is in the background monitor, it will periodically read a byte from the specified address location.

Example

To select the background monitor and have it periodically read a byte from address ffff hex in user space, select **C**onfig **G**eneral and enter the address 0000ffff@u in the "Target system keep alive" field.

To preset the interrupt stack pointer and program counter

1 Select **C**onfig **G**eneral to enter the configuration screen.

2 Use the arrow keys to move to the field next to “Initial stack pointer.”

3 Type in an even hexadecimal address value to be used for the initial value of the ISP (Interrupt stack pointer). This value should correspond to the value loaded at memory address 0 of your vector table. Press **<Enter>** to accept the value.

4 Use the arrow keys to move to the field next to “Initial program counter.”

5 Type in an even hexadecimal address value to be used for the initial value of the PC (Program Counter). This value should correspond to the value loaded at memory address 4 of your vector table. Press **<Enter>** to accept the value.

6 Press **<End>**, then **<Enter>** to save your changes and exit the configuration. Press **<Esc>** to discard your changes and exit the configuration.

Normally, if you run the emulator from reset, the processor fetches the values at offsets 0 and 4 from the vector table, and loads these values into the interrupt stack pointer and program counter registers. It then begins running from the program counter address value. (You run from reset by entering the command **P**rocessor **G**o **R**eset.)

However, if you reset the emulator (**P**rocessor **R**eset **H**old), break to the monitor, and then run the emulator, the stack pointer and program counter values are not read from these locations. This configuration item allows you to specify their initial values.

These configuration items are provided as a convenience feature to initialize the stack pointer and program counter to predefined values when the emulator enters the monitor after a reset. This allows you to reset, break, and then run as if you had done a run from reset. (You can accomplish the same thing by using the **R**egisters **M**odify command to set the PC and ISP values while in the monitor.)

Normally you will set the interrupt stack pointer to the value contained at offset 0 of your vector table, and the program counter to the value contained at offset 4 in your vector table.

Example

Assume that the memory range 7000..7fff is mapped as **eram** and reserved as stack space in your design. To set the interrupt stack pointer to 7ff0 and the initial program counter to 400h, select:

```
Config General 7ff0 <Enter> 400 <Enter> <End> <Enter>
```

If you now use the **P**rocessor **R**eset **H**old command to reset the processor, followed by a **P**rocessor **B**reak command to break to the monitor, the isp is set to 7ff0 and the pc is set to 400.

To enable break on write to ROM

1 Select **C**onfig **G**eneral to access the emulator configuration.

2 Use the arrow keys to move to the field next to the question “Enable breaks on writes to ROM?”

3 Type **y** if you want the emulator to break to the monitor when a write to ROM is detected. Type **n** if you want to ignore these events.

4 Press **<End>**, then **<Enter>** to exit the configuration and save your changes. Press **<Esc>** to exit the configuration and discard your changes.

Typically, attempts to write to ROM (mapped as **erom** or **trom**) will cause a hardware break. But you can choose to disable hardware breakpoints. For example, you might want to disable the break that would occur on a write to ROM because you need to see the next few bus cycles after the write in the trace list.

The memory in the emulation or target system will be changed by processor writes, even if that memory has been mapped as ROM.

Setting Other Configuration Items

The emulator allows you to restrict commands to those that won’t temporarily interrupt target execution to perform monitor functions. This is important for some systems that require non-stop real-time code execution.

Also, you can disable the processor cache memories. The emulation-bus analyzer can’t see instructions (or data) that are fetched from cache. This can make trace displays difficult to interpret. When you disable the caches, all instructions and data are fetched from memory, and therefore will appear on the bus where the analyzer can see them.

You can block target system interrupts from the emulation processor. This can help you troubleshoot problems with spurious interrupts or allow you to delay testing of interrupt service routines.

You can enable CMB interaction. This allows you to run multiple emulators in a synchronized environment.

To restrict to real-time runs

1. Select **C**onfig **G**eneral to access the emulator configuration.

2. Use the arrow keys to move the cursor to the field next to the question “Restrict to real-time runs?”

3. Type **y** if you want to restrict the emulator command set to those that won’t interrupt real-time runs. Type **n** if you want to use the full emulator command set.

4. Press **<End>**, then **<Enter>** to exit the configuration and save your changes. Press **<Esc>** to exit the configuration and discard your changes.

The emulator uses the emulation monitor program to implement some features, such as register displays. When the processor executes the monitor, it is not executing your target system program. This may cause problems in target systems that need real-time program execution.

If you enable this configuration item, the emulator will stop running user code only with the **P**rocessor **R**eset, **P**rocessor **B**reak, **P**rocessor **G**o and **P**rocessor **S**tep commands. Commands such as **R**egister that require a break to monitor are rejected. Also, the **M**emory **D**isplay and **M**emory **M**odify commands will be rejected if the address argument specifies standard emulation memory (not dual-ported) or target system memory.

While this configuration item affects which commands will be accepted, it does not affect hardware breakpoints such as write to ROM, break on analyzer trigger or guarded memory access breaks. It also doesn't affect the emulator's response to software breakpoints.

When you disable this configuration item, all commands are accepted.

To enable the processor cache memories

1 Select **C**onfig **G**eneral to access the emulator configuration.

2 Use the arrow keys to move to the field next to the question "Disable instruction/data caches?"

3 Type **n** if you don't want the emulator to disable the instruction and data caches. Type **y** if you want the emulator to disable the instruction and data caches.

4 Press **<End>**, then **<Enter>** to exit the configuration and save your changes. Press **<Esc>** to exit the configuration and discard your changes.

The MC68040 processor has a cache that stores the most recently used instructions and another cache for recently used data.

The cache memories increase processor performance, but the emulation-bus analyzer can't see processor accesses to the internal cache because they do not appear on the bus. This may cause confusing trace displays or failure to trigger the analyzer, especially if the code is a small loop where all the instructions and operands fit into cache and registers.

When you disable the caches, the emulation processor will always access external memory. Then the analyzer will see all bus cycles, which will improve the trace list. Processor performance will be reduced.

When you are concerned about measuring processor performance, enable the caches. Note that setting this configuration item to **y** does not enable the caches. The target system must deassert the $\overline{\text{CDIS}}$ line, and the target program must set the cache-control register enable bits of the CACR to enable the caches. If you are making analyzer measurements with the caches enabled, you may need to experiment to find suitable trigger combinations.

When you set this configuration item to **y**, the emulator asserts the $\overline{\text{CDIS}}$ line to disable the caches.

If you need to disable caching only for accesses to a specific memory block, use the **tci** memory map attribute. This allows you to capture analysis information for specific memory ranges without dramatically affecting overall system performance. See “To assign memory map terms” in this chapter.

To enable target system interrupts

1 Select **C**onfig **G**eneral to access the emulator configuration.

2 Use the arrow keys to move to the field next to the question “Enable interrupts from target?”

3 Type **y** if you want the target system interrupt signals to reach the emulation processor. Type **n** if you want to block the target system interrupt signals.

4 Press **<End>**, then **<Enter>** to exit the form and save your changes. Press **<Esc>** to discard your changes and exit the configuration.

You may want to disable target system interrupts if your target system interrupt logic doesn’t work correctly or isn’t finished, or you may want to disable these interrupts if the service routines and vectors aren’t assigned. You can enable the interrupts when you’re ready to test the interrupt handling.

Target system interrupts are always disabled during background monitor execution. The foreground monitor also disables interrupts during certain critical routines, such as monitor exit and entry.

You can enable interrupts during the remainder of foreground execution. See “To set the foreground monitor interrupt priority.”

To enable CMB interaction

1 Select **C**onfig **G**eneral to access the emulator configuration.

2 Use the arrow keys to move to the field next to the question “Enable CMB interaction?”

3 Type **y** if you want this emulator to respond to synchronized runs and breaks via the CMB. Type **n** if you don’t want this emulator to participate in coordinated measurements.

4 Press **<End>**, then **<Enter>** to save your changes and exit the configuration. Press **<Esc>** to discard your changes and exit the configuration.

The Coordinated Measurement Bus (CMB) allows you to use multiple emulators in a synchronized measurement. See Chapter 6 for more information.

To set the memory access size

1 Select **C**onfig **G**eneral to access the emulator configuration.

2 Use the arrow keys to move to the field labeled “Memory data access width.”

3 **Tab** to select the instruction type used by the monitor to access memory as follows:

- To have the emulator use the optimum data type for memory accesses, select **don’t care**.
- To have the monitor use byte data type for memory accesses, select **bytes**.
- To have the monitor use word data type for memory accesses, select **words**.
- To have the monitor use long word data type for memory accesses, select **longs**.

4 Press **<End>**, then **<Enter>** to save your changes and exit the configuration. Press **<Esc>** to discard your changes and exit the configuration.

The access size you select affects your interaction with memory displays, modifications, and searches. It determines whether the emulator interprets data values as bytes, words, or long words. Use this to set the size you need initially. If you use the size parameters with individual commands, the global display mode will be changed.

The access size is different. When you display or modify target memory or emulation memory that is not dual-port, the emulator uses the monitor to read or write target memory locations. The access size determines whether the emulator uses byte, word, or longword sizing for the memory accesses.

If set to **don’t care**, the size you include in your command is used for the access. It will temporarily override the **don’t care** for the access. If set to **byte**, **word**, or **long**, the size selected in your command will have no effect on the actual memory access; it will be what you specified for the memory access size.

If you choose **byte**, **word**, or **long** for your access size, only that size will be used. In cases where access is made to misaligned addresses or memory content having an insufficient number of bytes, the emulator will perform read-modify-write transactions for the access, using the size you specify.

Configuring the Triggers

The HP 64700 contains two internal lines, TRIG1 and TRIG2, over which trigger signals can pass from the emulator or analyzer to other HP 64700s on the Coordinated Measurement Bus (CMB) or other instruments connected to the BNC connector.

You can configure the internal lines to make connections between the emulator, analyzer, any external state or timing analyzer, CMB TRIGGER line, or BNC connector. Measurements that depend on these connections are called *interactive measurements* or *coordinated measurements.* See Chapter 6, “Making Coordinated Measurements” for more coordinated measurement information.

This section shows you how to:

- Drive analyzer trigger signals to the CMB or BNC.
- Break emulator execution on trigger signals.
- Arm analyzers on trigger signals.
- Drive and receive CMB or BNC trigger signals at the same time.

To drive analyzer trigger signals to the CMB or BNC

1 Select the **C**onfig **T**rigger command.

2 Move the cursor to the TRIG1 BNC field and use the **<Tab>** key to select the arrow pointing towards BNC.

3 Move the cursor to the TRIG1 CMB field and use the **<Tab>** key to select the arrow pointing towards CMB.

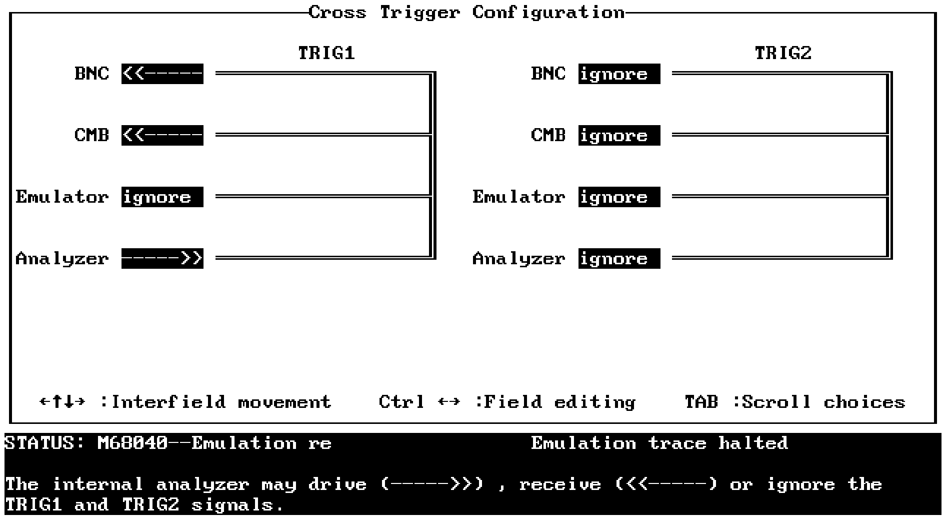
4 Move the cursor to the TRIG1 Analyzer field and use the **<Tab>** key to select the arrow pointing towards the internal line (away from “Analyzer”).

5 Press the **<End>** key and then press the **<Enter>** key to save the trigger configuration.

The preceding steps assume you are using TRIG1. Substitute TRIG2 in each step to use TRIG2.

Example

The trigger configuration screen below shows the emulation-bus analyzer driving both the CMB and BNC trigger signals over the TRIG1 line.



```
Cross Trigger Configuration

                 TRIG1                                  TRIG2
      BNC <<-----                            BNC ignore

      CMB <<-----                            CMB ignore

 Emulator ignore                        Emulator ignore

 Analyzer ----->>                       Analyzer ignore

  ←↑↓→ :Interfield movement    Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: M68040--Emulation re                Emulation trace halted

The internal analyzer may drive (----->>) , receive (<<-----) or ignore the
TRIG1 and TRIG2 signals.
```

To break emulator execution on trigger signals

1 Select the **C**onfig **T**rigger command.

2 Move the cursor to the Emulator field and use the **<Tab>** key to select the arrow pointing towards Emulator.

3 Move the cursor to the field associated with the source of the trigger signal (on the same internal line) and use the **<Tab>** key to select the arrow pointing towards the internal line.

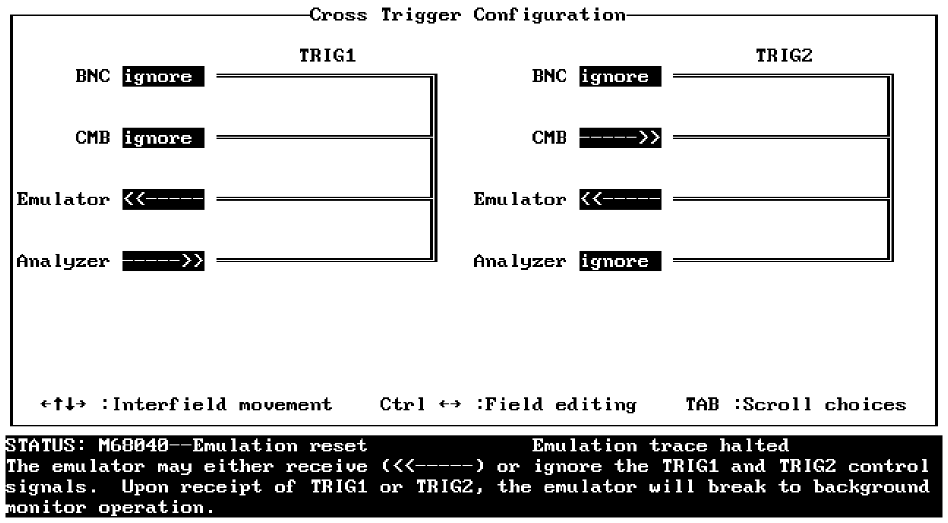
4 Press the **<End>** key and then press the **<Enter>** key to save the trigger configuration.

You can break user program execution into the monitor when a trigger signal is received from the emulation-bus analyzer, the CMB trigger line, or the BNC connector.

After the break occurs, the analyzer will stop driving the internal signal line that caused the break. Therefore, if TRIG1 is used both to break and to drive the CMB TRIGGER (for example), CMB TRIGGER will go true when the trigger is found and then will go false after the emulator breaks. However, if TRIG1 is used to cause the break and TRIG2 is used to drive the CMB TRIGGER, CMB TRIGGER will stay true after the trigger until the trace is halted or until the next trace starts.

Example

The trigger configuration screen below shows the emulation-bus analyzer trigger signal being used to break emulator execution over the TRIG1 internal line and the CMB TRIGGER signal being used to break emulator execution over the TRIG2 internal line.



```
                         Cross Trigger Configuration

                         TRIG1                                      TRIG2
      BNC ignore                          BNC ignore
      CMB ignore                          CMB ----->>
 Emulator <<-----                    Emulator <<-----
 Analyzer ----->>                    Analyzer ignore

  ←↑↓→ :Interfield movement    Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: M68040--Emulation reset              Emulation trace halted
The emulator may either receive (<<-----) or ignore the TRIG1 and TRIG2 control
signals.  Upon receipt of TRIG1 or TRIG2, the emulator will break to background
monitor operation.
```

To arm analyzers on trigger signals

1 Select the **C**onfig **T**rigger command.

2 Move the cursor to the Analyzer field and use the **<Tab>** key to select the arrow pointing towards the analyzer.

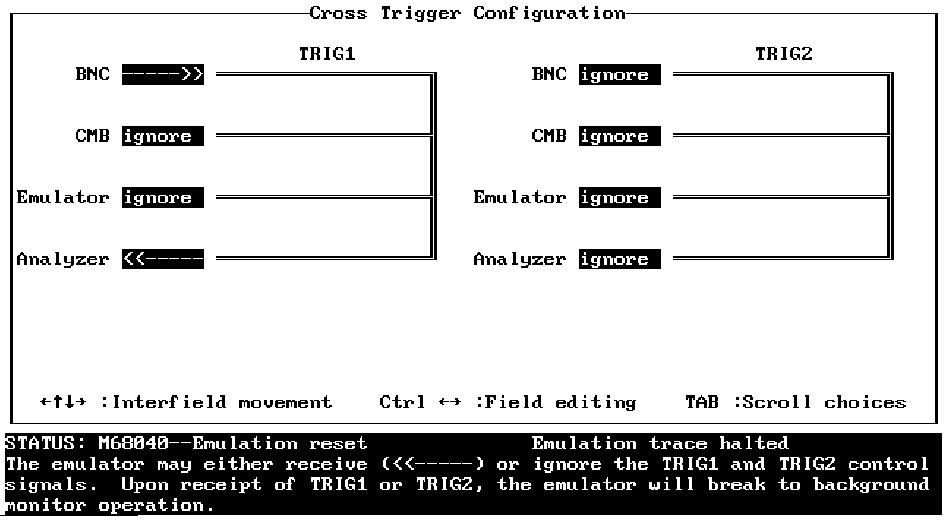
3 Move the cursor to the field associated with the source of the trigger signal (on the same internal line) and use the **<Tab>** key to select the arrow pointing towards the internal line.

4 Press <End> and then <Enter> to save the trigger configuration.

You can arm (that is, enable) the emulation-bus analyzer when a trigger signal is received from the CMB trigger line, the BNC connector, or the other analyzer's trigger output signal.

Example

The trigger configuration screen below shows the emulation-bus analyzer being armed by a trigger signal from the BNC connector over the TRIG1 internal line.



```
                         Cross Trigger Configuration

                     TRIG1                                   TRIG2
      BNC  ----->>                                BNC  ignore
      CMB  ignore                                 CMB  ignore
 Emulator  ignore                            Emulator  ignore
 Analyzer  <<-----                           Analyzer  ignore

 ←↑↓→ :Interfield movement     Ctrl ←→ :Field editing     TAB :Scroll choices
STATUS: M68040--Emulation reset              Emulation trace halted
The emulator may either receive (<<-----) or ignore the TRIG1 and TRIG2 control
signals.  Upon receipt of TRIG1 or TRIG2, the emulator will break to background
monitor operation.
```

To drive and receive CMB or BNC trigger signals at the same time

1. Select the **C**onfig **T**rigger command.

2. Move the cursor to the CMB or BNC field and use the **<Tab>** key to select the double-headed arrow pointing both towards CMB or BNC and the internal line.

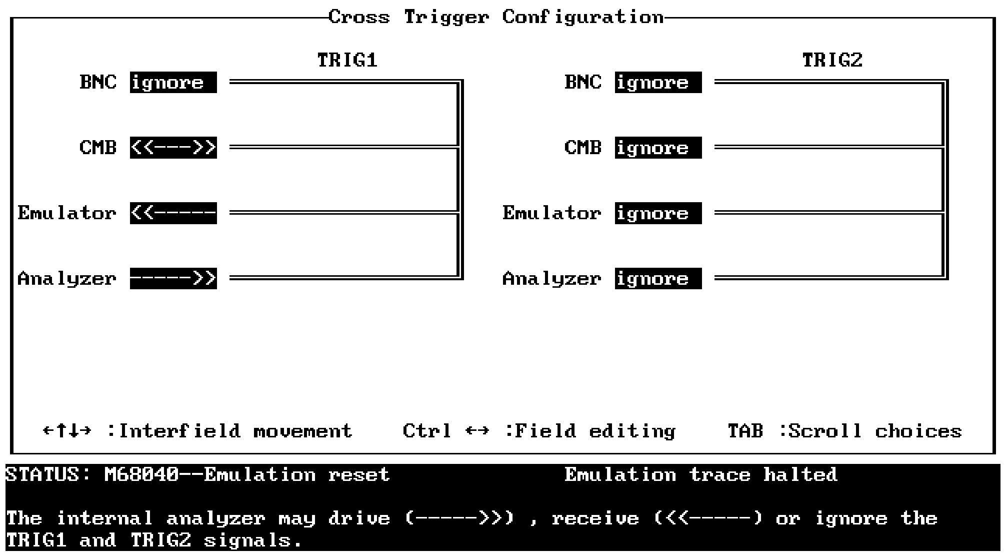
3. Move the cursor to the other fields (on the same internal line) and use the **<Tab>** key to make the desired selections.

4. Press **<End>** and then **<Enter>** to save the trigger configuration.

   The CMB or BNC TRIGGER signals may be driven by the analyzer trigger outputs, received to break the emulator or arm the analyzer, or both.

Example

The trigger configuration screen on the next page shows the emulation-bus analyzer trigger output and the CMB trigger line being used to break emulator execution over the TRIG1 internal line; it also shows the emulation-bus analyzer trigger driving the CMB TRIGGER line over the TRIG1 internal line.



```
                    Cross Trigger Configuration
                         TRIG1                               TRIG2
     BNC ignore                            BNC ignore

     CMB <<--->>                           CMB ignore

Emulator <<-----                      Emulator ignore

Analyzer ----->>                      Analyzer ignore


  ←↑↓→ :Interfield movement    Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: M68040--Emulation reset              Emulation trace halted

The internal analyzer may drive (----->>) , receive (<<-----) or ignore the
TRIG1 and TRIG2 signals.
```

Providing MMU Address Translations for the Foreground Monitor

When using the memory management unit (MMU) of the MC68040, the target system must provide correct address translations for the foreground monitor. To ensure correct address translations, you will need to understand your target system's memory map and MMU address translation structure. You may need to modify your mapping scheme or some of its mapping protections. If you do not obtain correct address translation for the foreground monitor, any attempt to break into the monitor after the MMU has been enabled may result in a target system bus error or undefined execution.

The foreground monitor will reside in the 4-Kbyte block of dual-port emulation memory. The dual-port 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.

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. For example, if the monitor code begins 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 up one pair of 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 a pair of 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.

- 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.

8

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.

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 the Installation and Service chapter in this manual for details.

☐ Run the performance verification procedure as described in the Installation and Service Chapter 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 the Installation and Service Chapter 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 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 following the trigger address. Then you can trigger on a write to the address of the 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. Refer to the paragraph titled "To display the trace list" in the "Using the Analyzer" chapter.

☐ 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. Refer to the paragraph titled "To display the trace list" in the "Using the Analyzer" chapter.

If there are unexplained states in the trace list

☐ Are you sure that the sequence, storage and trigger specifications are set up to exclude the states that you don't need?

☐ Try using the "Start" and "Operand" fields in the **A**nalysis **D**isplay form to inform the dequeuer which operand state belongs with the first instruction state.

☐ Try using the **low word** option to the **A**nalysis **D**isplay 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 the chapter titled, "Configuring the Emulator.")

If the analyzer won't trigger

☐ Instruction fetches from cache memory are not visible to the analyzer. You can disable the cache while using the analyzer using the **C**onfig **G**eneral command. Reenable the cache to improve performance when you're finished using the analyzer. See the chapter titled "Configuring the Emulator" for more information on disabling the cache.

If the analyzer fails to trigger on a program label address

☐ In your trace pattern, instead of specifying the address alone, precede it by **~3&** (for example, **~3&handler:<address symbol>**) to mask off the lower two bits of the address. This will cause the analyzer to trigger on any reference to the longword containing the desired symbol.

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. 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 and then the emulator.

2 Disconnect the emulator from the 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 the chapter titled "Installation and Service.")

4 Apply power to the emulator.

5 Start the PC Interface software. (See chapter 1.)

6 Select **S**ystem **T**erminal.

7 Type **tcf -e** to set the analyzer to easy configuration (required for performance verification).

8 Type **pv 1** to run performance verification.

If either the emulator or analyzer fail the performance verification, check the installation of those modules. (See the chapter titled "Configuring the Emulator.") If the installation is correct, contact your local HP Sales and Service office for assistance.

If you have trouble mapping memory

☐ The emulator uses a best fit algorithm to assign memory blocks to map requests. Since the memory block sizes available depend on the SRAM installations and the use of the dual-port memory, it is 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 SRAM installations will aggravate the problem.

☐ 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 that purpose. 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.)

See the section “Memory Mapping” in the chapter titled "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 and/or HP 64172B 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 (**eram**) or emulation ROM (**erom**). DMA to emulation memory resources 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, and 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 **P**rocessor **G**o **P**c or **A**ddress command instead of a **P**rocessor **G**o **R**eset command. For further information, refer to the chapter on plugging the emulator into 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 **P**rocessor **D**eMMU **V**erbose_Load 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.

- ☐ 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. Only leave address ranges that need to be reverse translated in the file. Then 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 the chapter titled "Using the Emulation-Bus Analyzer" for how to store and load a deMMUer file.

- ☐ Read "Using the deMMUer" in the chapter titled "Using Memory Management" for ways to make more efficient use of deMMUer resources, and to understand how deMMUer resources are allocated when using different root pointers or when using function-code mappings.

If you only see physical memory addresses in the analyzer measurement results

☐ Check to see if you enabled the deMMUer with the command: **P**rocessor **D**eMMU **E**nable.

☐ Check to see if you loaded the deMMUer with the information needed to reverse translations made by the MMU with the command: **P**rocessor **D**eMMU **V**erbose_Load.

☐ Read "Using the deMMUer" in the chapter titled "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 **P**rocessor **D**eMMU **V**erbose_Load command to see a list of the physical address spaces that will be deMMUed.

☐ Check the setup of the MMU mapping tables. Make sure that 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 **P**rocessor **D**eMMU **L**oad 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 **P**rocessor **D**eMMU **L**oad 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 **P**rocessor **D**eMMU **L**oad command. Issue the **P**rocessor **D**eMMU **L**oad command again.

- ☐ Read "Using the deMMUer" in the chapter titled "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 command: **P**rocessor **R**eset **M**onitor. 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 the chapter titled, "Using Memory Management" for a detailed example that discusses how to solve a "can’t break into monitor" problem.

Part 3

Reference

Emulator Features

This part of the manual gives a comprehensive description of all emulator features, and all PC Interface commands. It does not attempt to explain *how* to do things with the emulator. Instead, it tells what commands are available, and what they do.

In This Part

Chapter 9, "Using Memory Management," gives you an understanding of emulation and analysis of the MC68040 Memory Management Unit.

Chapter 10, “Emulator Commands,” gives a breakdown of the menu hierarchies of the PC Interface. Each group of related commands is explained, to give you a complete picture of what each command actually does.

Chapter 11, “Expressions,” shows the syntax of expressions (such as addresses, registers names) that are unique to the MC68040.

Chapter 12, “Error Messages,” lists the PC Interface error and status messages unique to the MC68040 emulator, and gives their meanings.

Chapter 13, “Data Formats,” describes the content and format of the various files used by the PC Interface, such as symbol database files and configuration files.

Chapter 14. “Specifications and Characteristics,” contains the electrical specifications and characteristics for the MC68040 emulator.

If you want to know how to accomplish commonly performed tasks, see Part 2 of this manual. If you are looking for a general introduction to using the emulator, see Part 1.

9

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. Select **A**nalysis **D**isplay and refer to "To display the trace list" in chapter 5 (Using the Analyzer) of this manual for details.

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 (ex: **<address>@s**).

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/disable MMU?" general configuration parameter.

If you enter **disable**, the $\overline{\text{MDIS}}$ line is asserted. If you enter **enable**, 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 an 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 the emulation configuration chapter (Chapter 7).

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 enter a **P**rocessor **G**o **A**ddress **<address>** command, the address you enter must be the 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 (**M**emory **D**isplay **W**ord **100**) 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 **M**emory **D**isplay **W**ord **100@a** (where "a" = "absolute" = "physical").

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 the access.

Seeing Details of 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, enter the command **P**rocessor **M**MU **M**appings **<End><Enter>**. 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
  Logical Address         Physical Address         Attributes
!STATUS  160! MMU is not enabled via TC register
```

The above listing shows privilege modes were included in the mapping scheme. The logical and physical addresses are 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 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 **P**rocessor **M**MU **M**appings 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 **M**appings display for paths designated invalid, or for paths containing illegal entries.

To avoid a list of mappings that scrolls for a long time, specify a limited address range in your command. The command **P**rocessor **M**MU **M**appings, followed by Address(range): **0..0ffff** instructs the emulator to show the valid mappings for only the logical addresses in the range of 0 through 0ffff, instead of all possible mappings.

Another way to limit the number of address ranges shown in an MMU mappings display is to limit the listing to only user or supervisor address space. The command **P**rocessor **M**MU **M**appings, followed by Address(range): **0..0ffff@u** will show all of the mappings for addresses from 0 through 0ffff in user address space.

Note: For convenience, the **P**rocessor **M**MU **M**appings command will use the logical address range from the most recent entry in the **P**rocessor **M**MU **M**appings form. To change the default logical address range back to the full address space, enter 0..0ffffffffh beside Address(range): in the form.

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, use the command: **P**rocessor **M**MU **M**appings, followed by Address(range): **<address> [..[<address>]]@s**. This tells the emulator to show the supervisor mapping for the associated logical address or address range.
- To see only the mappings in user address space, use the command: **P**rocessor **M**MU **M**appings, followed by Address(range): **<address> [..[<address>]]@u**.
- If you specify no privilege mode, then 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, enter a command such as: **P**rocessor **M**MU **T**ables, and in the form that appears, enter the desired address. The **T**ables 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

Status can be assigned to an address at any point in its mapping. To interpret status, you must OR the status information at each level of the mapping. For example: the "M" bit shows that the content of the page indicated by Table C has been modified (by a write or read-modify-write). This applies only to addresses in this page. A "y" might have been shown under the "S" status bit in the A line. It would indicate that only supervisor accesses are allowed for pages under the A table. This restriction would apply to all addresses of this table, even though S=1 only appeared at the upper level of the table.

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 would use a command like **P**rocessor **M**MU **T**ables, and then enter the Address: **40f8@u**. 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. **u** or **s**).

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 enter a command like: **P**rocessor **M**MU **T**ables, and then enter the Address: **40f8**, and then use **Tab** to select Table 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

- **P**rocessor **D**eMMU **D**isable. 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.
- **P**rocessor **D**eMMU **E**nable. Your analyzer receives logical addresses translated by the deMMUer according to the tables in place when you last loaded the deMMUer.
- **P**rocessor **D**eMMU **L**oad. The emulator reads the MMU registers and interprets the translation tables to load the deMMUer. You will see a form that allows you to use the present values of the MMU registers during the loading process, or to specify values to override the present MMU registers during the loading process. The deMMUer is also enabled after it is loaded.
- **P**rocessor **D**eMMU **V**erbose_Load. This sets verbose mode for the deMMUer load function. It offers the same register-override selections as Processor DeMMU Load, and additionally displays a list 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 priviledge mode.

- Processor DeMMU File Load. The emulator reads a file containing an MMU setup, and loads the deMMUer to reverse translate addresses in the file. This file must have been created using the Processor DeMMU File Store command. It will have a ".ED" filename extension.
- Processor DeMMU File Verbose_Load. The emulator reads a file containing an MMU setup, and loads the deMMUer to reverse translate addresses in the file. A list is displayed of the physical address ranges that will be reverse translated by the deMMUer.
- Processor DeMMU File Store. The emulator stores the present setup of the MMU in a file. You can edit this file to remove address ranges that do not need to be reverse translated. Then you can load this file into the deMMUer using the Processor DeMU File Load or Verbose_Load command.

What the emulator does when it loads the deMMUer

When the emulator loads the deMMUer, it does the following:

- Temporarily breaks into the monitor.
- Reads the MMU tables and provides spaces in the deMMUer table to reverse translate all of the valid addresses in the MMU tables.
- If there are more valid physical memory addresses than can be accommodated in the deMMUer table, the emulator reads the emulation memory map for help in selecting appropriate address ranges to reverse translate.
- Provides deMMUer table entries to reverse translate small address spaces defined in the emulation memory map before providing table entries to reverse translate larger address spaces.
- Allocates remaining resources to reverse translate addresses beginning with the lowest remaining address.
- If there are valid physical memory addresses remaining after all available spaces have been used in the deMMUer table, the emulator displays the "out of resources" message.

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 enter the **P**rocessor **D**eMMU **L**oad command and your emulator performs its task and returns the main menu set of commands to the screen, you won’t need to know about the deMMUer resource limitations. When the deMMUer is loaded without any problems, the main menu set of commands 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 "deMMUer out of resources".

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.

4M RAM	**Unused**	**2M Peripherals**	**Unused**	**4M ROM**	**Unused**

```
0            4M                256M       258M                512M      514M
```

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.

00000000..003fffff eram
10000000..101fffff tram
20000000..203fffff trom
other tram

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 **V**erbose_Load 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 the chapter titled "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
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 "Processor MMU Mappings" 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 **P**rocessor **M**MU **M**appings 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, enter an enable value in the Processor MMU Mappings form that appears when you enter the Processor MMU Mappings command. Do this by selecting "no" beside "Use TC reg:", and entering 8000h in the field beside "Value:".

You can also use the **P**rocessor **M**MU **M**appings command to test your mappings before you enable the MMU. This command, by itself, reads the 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 **P**rocessor **M**MU **M**appings command, the emulator reads the MMU registers (TC, URP, SRP, ITT0, ITT1, DTT0, and DTT1) and the MMU tables. If you do not have correct values in the registers, the emulator will let you specify correct values to be used when composing the display of translations. For example, by entering "no" beside Use TC reg:, a "Value" field appears where you can specify a new value to override the TC register.

If the TC register has a disable value, you must use this form to override it with an enable value before the emulator can read the content of the MMU tables.

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 use the **P**rocessor **M**MU **T**ables command and enter the suspect addresses in the Address: field to look at the mapping tables used to translate those 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 (**R**egister **D**isplay) to see the value of the logical address in the program counter. Then use the **P**rocessor **M**MU **T**ables command and specify the suspect <address> 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 (**A**nalysis **T**race **M**odify and trigger on the **<pc address>**). Select a **center** trigger so you can see activity preceding and following the trigger point. In order to capture every transaction on the emulation bus, qualify all states for capture.

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 (**P**rocessor **M**MU **T**ables for each **<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 using a command, such as (**P**rocessor **M**MU **T**ables, Address: **<suspect_address>**, Table level: **b**).

Failure due to system halt

If the emulator and/or target system simply stops operating, set up the analyzer to trace with a trigger-never specification (Trigger on **no state**) 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. Use the addresses in **P**rocessor **M**MU **T**ables, Address: **<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 with the command **P**rocessor **R**eset **M**onitor.

The **R**eset command 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. Enter the command: **P**rocessor **M**MU **M**appings and use the Processor MMU Mappings form to override the TC register with 8000h (an enable value that allows the emulator to read the MMU tables).

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 with your **P**rocessor **M**MU **M**appings command (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 enter the command: **P**rocessor **M**MU **T**ables (plus the TC register override), and enter the address **4000** on the form. 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 4000h was placed in physical address C000h. Table C points to the page containing the foreground monitor. The base address of Table C is 00000600h, and the content used by logical address 4000h is at index 016 whose physical address is 00000640h. 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: **M**emory **M**odify **L**ong, and beside Address, enter: **00000640@a=0000401f**.

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. Enter the command: **P**rocessor **M**MU **T**ables, (include the value to override the TC register) and enter the address **4000** on the form.

```
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.

10

Emulator Commands

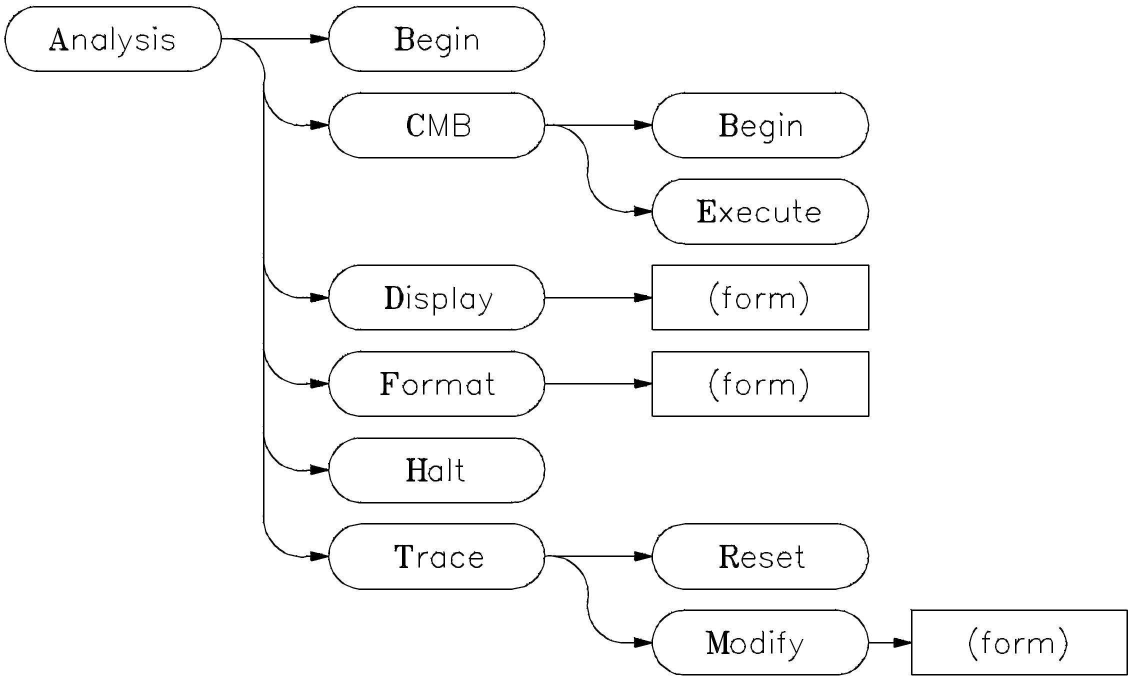
A complete description of all emulator commands, and what they do.

This chapter shows all commands that you can select from PC Interface menus. Each top-level command on the main menu (for example, **P**rocessor) is shown as a horizontal tree, with the main menu command at the left, and all of its subcommands branching to the right. The tree also shows which subcommands require typed-in information (such as an address), and which commands bring up a form to be filled in. The top level commands are arranged in alphabetical order to help you find them quickly.

In addition to showing command hierarchies, this chapter explains the purpose of each command. However, no attempt is made to explain things like the syntax of symbolic addresses, or how to fill in a particular form. Instead, you are told *what* each command does.

Analysis Commands

The **A**nalysis command hierarchy is shown below. These commands are all concerned with operation of the emulation bus analyzer. There are several configuration items that affect the analyzer. These are covered in the **C**onfig command hierarchy, later in this chapter.



Analysis Begin

Analysis **B**egin — Starts an analyzer trace. This causes the analyzer to begin monitoring bus activity and storing in the trace buffer any states that match your trigger and storage qualifiers. The **A**nalysis **T**race command, described later in this chapter, lets you specify which states you wish to capture in the trace buffer. However, until you actually *start* a trace, the analyzer isn't even "looking" at the bus, so no states will be captured.

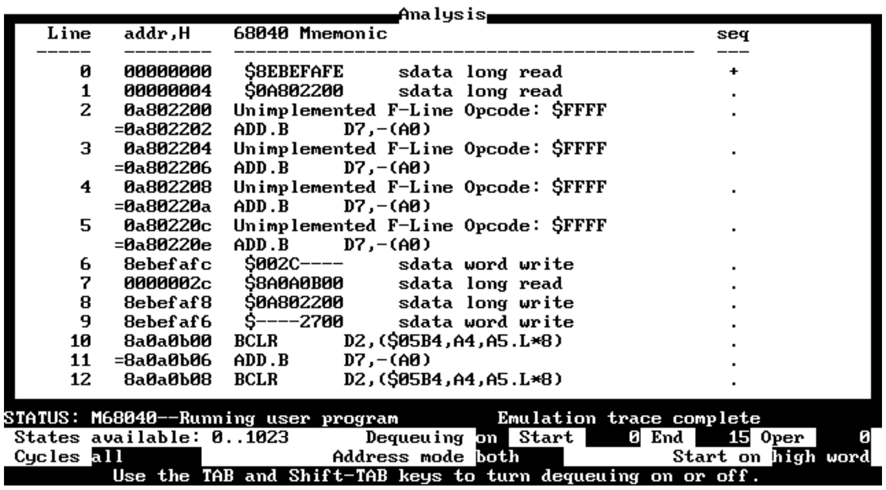
Analysis CMB

Analysis **C**MB **B**egin — Informs all emulators connected to the CMB, that upon receipt of the next EXECUTE signal, each emulator should begin an analyzer trace.

Analysis **C**MB **E**xecute — Sends an EXECUTE signal to all analyzers. This is exactly the same command as **P**rocessor **C**MB **E**xecute.

Analysis Display

Analysis **D**isplay — Opens the form shown in the following figure. This command allows you to view the contents of the trace buffer. It gives you a formatted representation of the buffer's contents in the "Analysis" window. Because there is usually far more information in the trace buffer than can be displayed on your screen, the form shown below allows you to specify exactly what you wish to see. The meanings of the fields are as shown.



```
                                    Analysis
 Line    addr,H     68040 Mnemonic                                      seq
 -----   --------   -------------------------------------------        ---
     0   00000000    $8EBEFAFE      sdata long read                      +
     1   00000004    $0A802200      sdata long read                      .
     2   0a802200   Unimplemented F-Line Opcode: $FFFF                   .
        =0a802202   ADD.B      D7,-(A0)
     3   0a802204   Unimplemented F-Line Opcode: $FFFF                   .
        =0a802206   ADD.B      D7,-(A0)
     4   0a802208   Unimplemented F-Line Opcode: $FFFF                   .
        =0a80220a   ADD.B      D7,-(A0)
     5   0a80220c   Unimplemented F-Line Opcode: $FFFF                   .
        =0a80220e   ADD.B      D7,-(A0)
     6   8ebefafc    $002C----      sdata word write                     .
     7   0000002c    $8A0A0B00      sdata long read                      .
     8   8ebefaf8    $0A802200      sdata long write                     .
     9   8ebefaf6    $----2700      sdata word write                     .
    10   8a0a0b00   BCLR       D2,($05B4,A4,A5.L*8)                      .
    11  =8a0a0b06   ADD.B      D7,-(A0)                                  .
    12   8a0a0b08   BCLR       D2,($05B4,A4,A5.L*8)                      .

STATUS: M68040--Running user program         Emulation trace complete
 States available: 0..1023        Dequeuing on  Start        0 End       15 Oper        0
 Cycles all                   Address mode both                   Start on high word
          Use the TAB and Shift-TAB keys to turn dequeuing on or off.
```

Analysis Display Form

"States Available:" Displays the range of states available for display. Values are dependent on the trigger position and whether counting is **on** or **off**. If counting is **on**, the number of states available is reduced from 1024 to 512, since counting consumes states and memory. Negative states occur before the trigger and positive states occur after the trigger, with the trigger being at zero.

“Dequeuing:” Turns software instruction dequeuing **on** or **off** using the **Tab** or **Shift-Tab** key. If dequeuing is **on**, a value must be entered in the “Oper” field.

“Start”/“End:” These hold the range of states to be displayed. “Start” holds the first state and “End” holds the last state to be displayed. These values must be contained in the “States Available” range. Each time you display the trace buffer, these two fields are automatically modified to reflect the next of group of undisplayed lines.

“Oper:” Used when software instruction dequeuing has been enabled, this field will only appear during the first display of the most recent trace, or if the “Start” state has been modified. It will hold a number that represents the operand cycle associated with the first disassembled instruction.

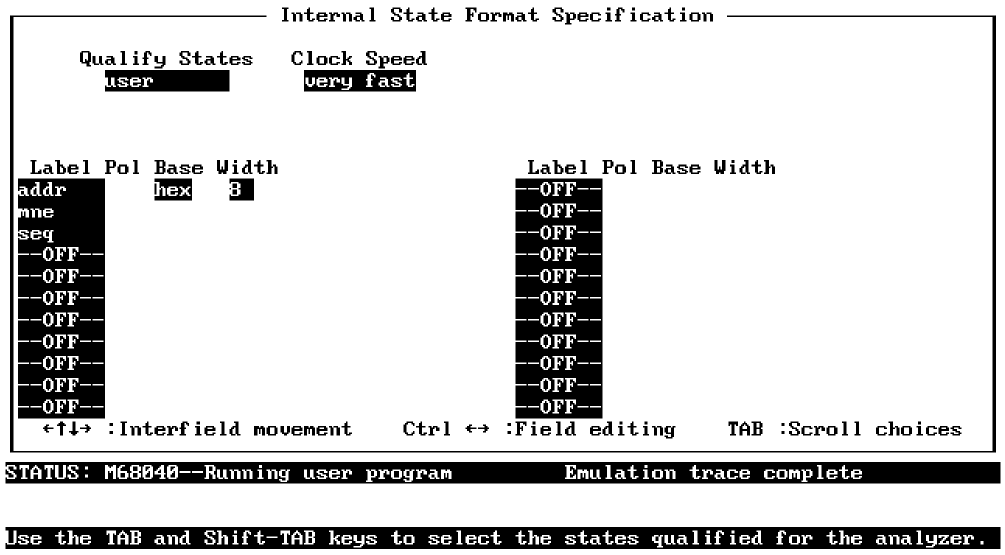
“Cycles:” The **Tab** or **Shift-Tab** key is used to select **all**—to display both operand and instruction cycles, or **instr only**—to display only instruction cycles.

“Address Mode:” The **Tab** or **Shift-Tab** key is used to select **absolute**—to display numeric addresses, **symbol**—to display only symbols (when they are available), or **both**—to display a mix.

“Start on:” Appears only during the first display or the entry of a “Start” state other than the default. The **Tab** or **Shift-Tab** key is used to select **High Word** or **Low Word** to instruct the analyzer to disassemble instructions beginning with the upper or lower 16 bits of captive data.

Analysis Format

Analysis **F**ormat — Affects what you see when you display the trace buffer. However, selecting this command does not actually display the trace buffer. Instead, it lets you fill in the form and fields shown below, specifying the output format for all subsequent **A**nalysis **D**isplay commands.



```
Internal State Format Specification

   Qualify States   Clock Speed
     user             very fast


 Label Pol Base Width                        Label Pol Base Width
addr       hex   8                          --OFF--
mne                                         --OFF--
seq                                         --OFF--
--OFF--                                     --OFF--
--OFF--                                     --OFF--
--OFF--                                     --OFF--
--OFF--                                     --OFF--
--OFF--                                     --OFF--
--OFF--                                     --OFF--
--OFF--                                     --OFF--
--OFF--                                     --OFF--
  ←↑↓→ :Interfield movement    Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: M68040--Running user program         Emulation trace complete

Use the TAB and Shift-TAB keys to select the states qualified for the analyzer.
```

Analysis Format Form

“Qualify States:” The **Tab** or **Shift-Tab** key is used to select **user**—to display user program states, **background**—to display monitor program states, or **all**—to display both user and monitor program states.

“Clock Speed:” The **Tab** or **Shift-Tab** key is used to select **slow, fast,** or **very fast**. The trace state and time count qualifiers are limited by the analyzer clock rate settings as in the following table.

Analyzer clock rate	Clock Speed setting	Valid Count Qualifier options
clock ≤ 16 MHz	slow	Count <state> Count time
clock ≤ 20 MHz	fast	Count <state>
clock ≥ 20 MHz	very fast	Count none

To determine the correct setting for this field use the following equation.

$$\text{Analyzer Clock Rate} = \frac{\text{Processor Clock Rate}}{(1 + \text{numberof wait states})}$$

Remember that the emulator requires one wait state for all accesses when the external clock is greater than or equal to 25 MHz. Choose the data rate option according to the data rate you calculate with the equation above. If no burst cycles are performed, the analyzer clock speed can be set "slow".

“Label:” Use the **Tab** or **Shift-Tab** key to select one of the following:

addr (for address lines)
data (for data lines)
mne (for instruction mnemonics)
stat (for processor status information)
count (state and time counts)
seq (sequencer state change indicator)
OFF (no label in this position)

(Base): If a base specification is available for the label, use **Tab** or **Shift-Tab** to select one of the following base options:

hex (hexidecimal)
bin (binary)
oct (octal)
dec (decimal)

(Width): This field appears only if **addr** was chosen in the Label field. It holds a width in the range of 4..50.

(Relative): This field appears only if **count** was chosen in the Label field. Use the **Tab** or **Shift-Tab** key to select **rel** to show the count (or time) relative to the previous state, or **abs** to show the count relative to the trigger state.

A good way to understand the relationship between the **A**nalysis **F**ormat and **D**isplay commands is to think of the display command as producing a report. The format command lets you customize the columns of that report, indicating what goes in each column (address, time stamp, etc.), the order of those columns, and the width of each column. The display command then uses that format for all reports it produces, until you change the format again.

Analysis Halt

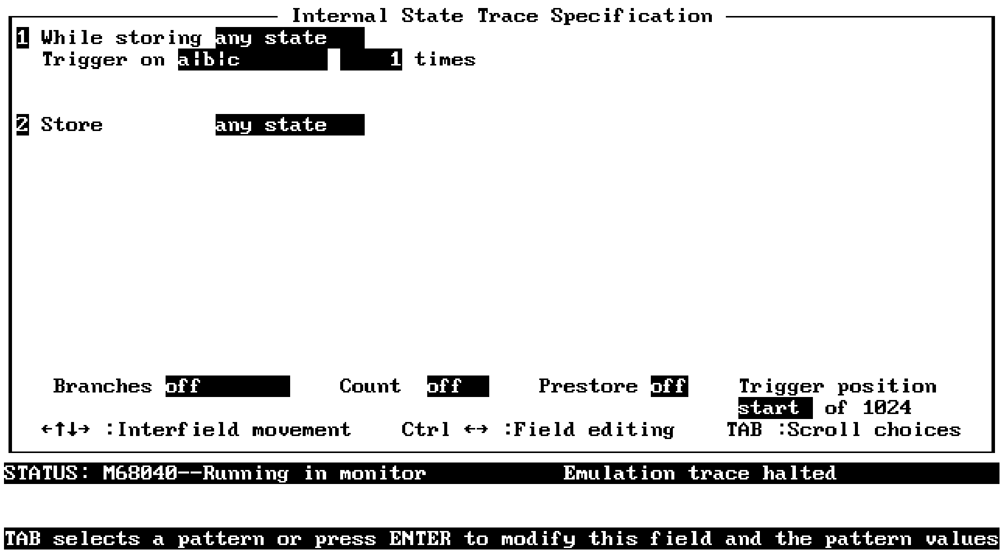
Analysis **H**alt — Forces termination of the current trace, even though the trace buffer may not be full yet. Most of the time, when you start a trace, the trace buffer fills up quite rapidly. However, if you specify a trigger that never gets matched, or you have a complex series of sequencer terms, the trace buffer may never fill. To remedy the situation, you need to halt the incomplete trace, respecify your trigger or sequence terms, and then restart the trace.

Analysis Trace

Analysis **T**race — Lets you specify the trigger, storage qualifier, and any sequence terms you need. It does not start the trace. To do that, you need to use the **A**nalysis **B**egin command.

Analysis **T**race Reset — This resets the trace specification back to the default: trigger on any state, and store any states after the trigger, with the trigger state appearing as the first state in the trace buffer.

Analysis **T**race **M**odify — This allows you to describe exactly what the trigger pattern is, plus any storage qualifier patterns, and branch term qualifiers. The two forms and the fields are as follows:


```
                    Internal State Trace Specification
1 While storing any state
  Trigger on a|b|c            1 times


2 Store         any state









   Branches off          Count off      Prestore off     Trigger position
                                                         start  of 1024
  ←↑↓→ :Interfield movement    Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: M68040--Running in monitor          Emulation trace halted

TAB selects a pattern or press ENTER to modify this field and the pattern values
```

Analysis Trace Modify Form: Level 1

“Count:” The **Tab** or **Shift-Tab** key is used to select **time**—to count time intervals, **state**—to count bus states, or **off**—to disable counting. If the count is set to **state**, then a qualifier field appears below the “Count” field.

“State Qualifier (unlabeled):” This field appears only if **state** was chosen in the “Count” field. It holds a qualifier used when counting bus states. See chapter 5, “To define a qualifier” for more information.

“Term Number (unlabeled):” This field displays the term number. The **Tab** or **Shift-Tab** key is used to select **I**—to insert a term, **D**—to delete a term, or **T**—to assign a trigger level. If **T** is chosen, then a value must be entered in the “Trigger on” field.

“Trigger on:” This field appears only if **T** was entered in the “Term Number” field. It holds a state qualifier name. **<Enter>** leads to the second form shown below.

“Times:” This field holds a number that specifies how many times the qualifier must be found before the sequencer advances to the next state. It must be a number between 1 and 65535.

“Branches:” The **Tab** or **Shift-Tab** key is used to select **restart on**—to enable a global restart, **off**—to disable the global restart, or **per level**—to specify individual secondary branch qualifiers. If **restart on** is chosen, then a state qualifier must be entered in the field below the “Branches” field. If **per level** is chosen, then a state qualifier must be entered in the field next to the “Else on” label.

“Branch Qualifier (unlabeled):” This field appears below the “Branches” field only if **restart on** was chosen in the “Branches” field. It holds a qualifier used during global restarts. See chapter 5, “To define a qualifier” for more information.

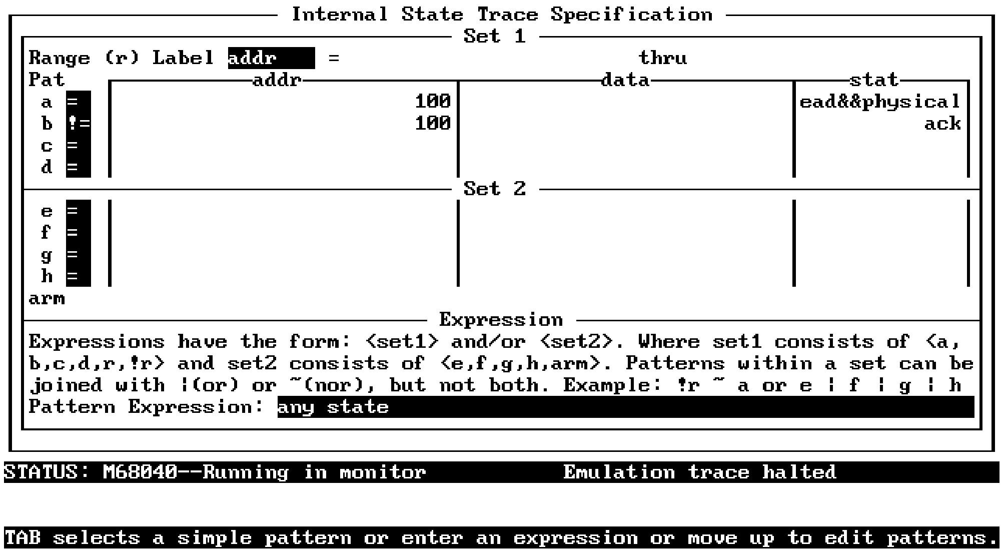
“Level Qualifier (unlabeled):” This field appears next to the “Else on” label only if **per level** was chosen in the “Branches” field. It holds the sequence level used for secondary branches. See chapter 5, “To define a qualifier” for more information.

“Prestore:” The **Tab** or **Shift-Tab** key is used to select **on**—to enable a prestore specification, or **off**—to disable a previous prestore specification. If **on** is chosen, then a state qualifier must be entered in the field below the “Prestore” label.

"Prestore Qualifier (unlabeled):" This field appears under the "Prestore" label only if **on** was chosen in the "Prestore" field. It holds the prestore qualification. See chapter 5, "To define a qualifier" for more information.

"Trigger Position:" The **Tab** or **Shift-Tab** key can be used to select **start**, **center**, or **end**. The trigger may be positioned at a specific state by typing in a value between 1 and 512 (or 1 and 1024 if counting is off).

"Equals:" Every pattern on the screen has an "Equals" field next to it. The **Tab** or **Shift-Tab** key is used to select **=**—to set the pattern equal to an expression, or **!=**—to set the pattern not equal to the expression.



```
                      Internal State Trace Specification
                                  Set 1
Range (r) Label addr     =                        thru
Pat              addr                 data                 stat
 a  =                      100                        ead&&physical
 b  !=                     100                                  ack
 c  =
 d  =
                                  Set 2
 e  =
 f  =
 g  =
 h  =
arm
                                Expression
Expressions have the form: <set1> and/or <set2>. Where set1 consists of <a,
b,c,d,r,!r> and set2 consists of <e,f,g,h,arm>. Patterns within a set can be
joined with |(or) or ~(nor), but not both. Example: !r ~ a or e | f | g | h
Pattern Expression: any state

STATUS: M68040--Running in monitor          Emulation trace halted


TAB selects a simple pattern or enter an expression or move up to edit patterns.
```

Analysis Trace Modify Form: Level 2

"Addr:" Each address field in the column holds an address expression, or can be left blank. The expressions can be made of numeric expressions, symbols, or a combination of both.

"Data:" Each data field in the column holds a data expression made of numeric expressions, or can be left blank.

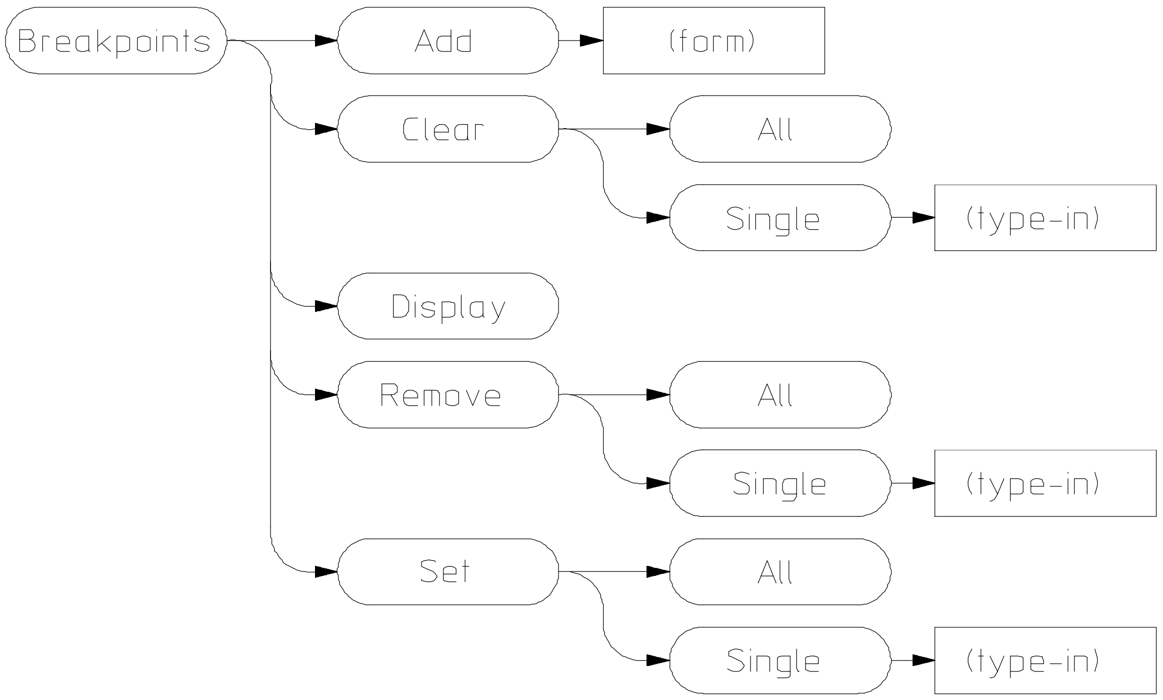
“Stat:” The **Tab** or **Shift-Tab** key can be used to select from predefined status qualifiers. See STATUS in the chapter titled "Expression Syntax" for a listing. You can also enter a status expression using numeric values and combine status qualifiers using logical operators.

“Range (r) Label:” This allows you to define a range of address or data values. The **Tab** or **Shift-Tab** key allows you to select **addr** or **data** as the group of processor lines for which you want to define a range. If you chose **addr**, then upper and lower bounds can be entered using address expressions. If you chose **data**, then use data expressions.

Breakpoint Commands

Shown below is the menu hierarchy for the **B**reakpoints command. These are debugging commands which allow you to stop your program at critical points, examine registers and/or memory locations, then resume execution, if you wish.

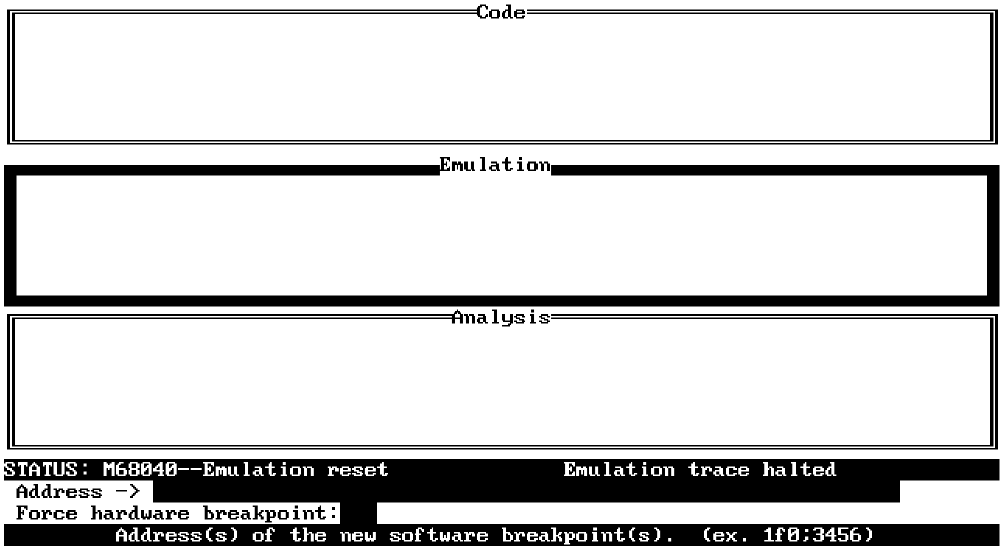
Breakpoints are implemented with the BKPT instruction. When you issue **P**rocessor **G**o command, the emulator replaces each instruction which has an enabled breakpoint with a BKPT instruction. If your program reaches any of these BKPT instructions, a break to monitor occurs. The emulator then replaces the BKPT with the instruction that was originally there, and disables the breakpoint.



64783S21

Breakpoints Add

Breakpoints **A**dd — Adds a breakpoint to the list of breakpoints in your program. It also *sets* the breakpoint, meaning the breakpoint is enabled. When a breakpoint is executed in your program, it is automatically disabled, or “unset”. However, the breakpoint address is retained in the list of breakpoints. To set the breakpoint again, you use the **B**reakpoints **S**et command.



```
STATUS: M68040--Emulation reset            Emulation trace halted
 Address ->
 Force hardware breakpoint:
         Address(s) of the new software breakpoint(s).  (ex. 1f0;3456)
```

Breakpoints Add Form

"Address:" specifies the address where the breakpoint is to be added.

"Force hardware breakpoint:" Select "yes" to force the emulator to use one of its eight hardware resources to store the breakpoint address instead of storing the breakpoint instruction in software. Use this option when you know that the software might be overwritten before the breakpoint instruction is fetched. Select "no" to have the breakpoint instruction placed at the address in memory. An unlimited number of breakpoints can be placed within the software.

Breakpoints Clear

Clearing a breakpoint “unsets” it, without removing it from the list of breakpoints. That way, you can temporarily disable one or all breakpoints, then later enable them quickly, without reentering the addresses from scratch.

Breakpoints **C**lear **A**ll — This command permits you to clear all the breakpoints in the breakpoint list.

Breakpoints **C**lear **S**ingle — This command permits you to clear one of the breakpoints in the breakpoint list.

Breakpoints Display

Breakpoints **D**isplay — Lists all the breakpoints in your program, and tells whether each is enabled or not.

Breakpoints Remove

Removing a breakpoint from the list also disables it, as you might expect. If you want to enable a breakpoint after it has been removed from the list, you must add it to the list again.

Breakpoints **R**emove **A**ll — This permits you to remove all the breakpoints from the breakpoint list.

Breakpoints **R**emove **S**ingle — This permits you to remove one of the breakpoints from the breakpoint list.

Breakpoints Set

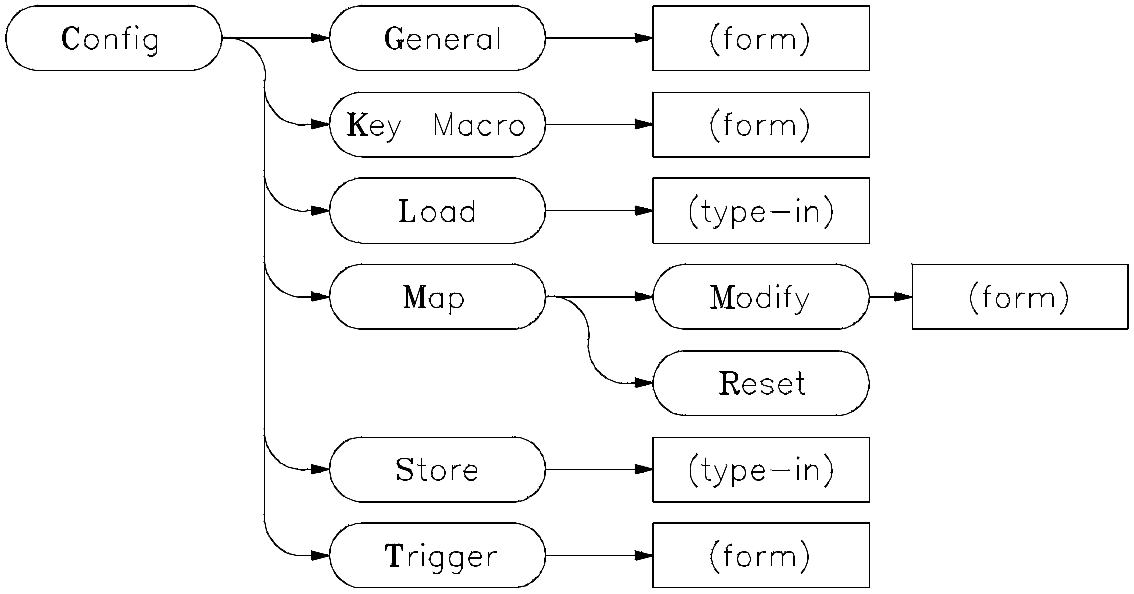
Breakpoints **S**et **A**ll — This permits you to set all the breakpoints in the breakpoint list.

Breakpoints **S**et **S**ingle — This permits you to set one of the breakpoints in the breakpoint list.

Setting a breakpoint enables it. A breakpoint must be added to the list before you can set it. When a breakpoint is encountered in your program, it is automatically cleared (disabled). You can also explicitly clear breakpoints with the **B**reakpoints **C**lear command.

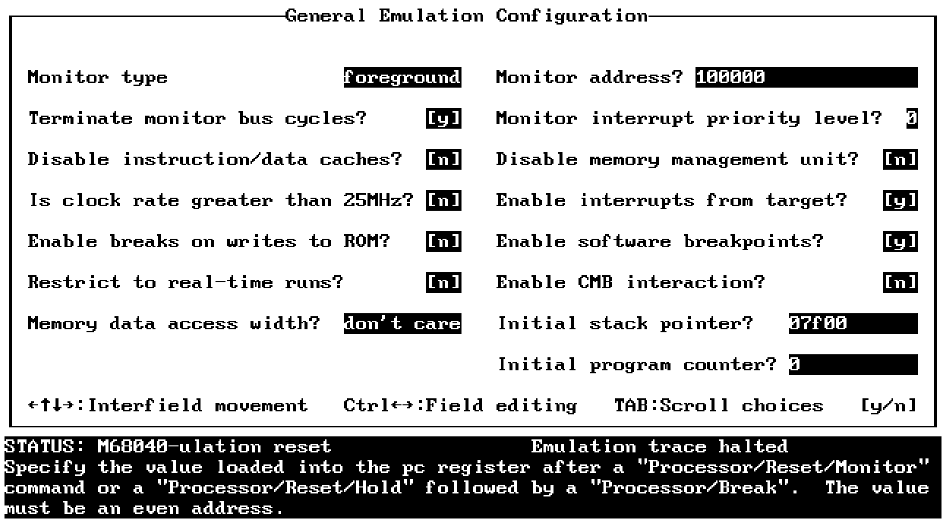
Configuration Commands

Configuration commands allow you to customize emulator configuration for the specific needs of your target system and programs. Configuration is divided into two main areas: memory map configuration and general emulator configuration. These are explained in the first few pages of “Configuring the Emulator” (chapter 7). The **C**onfig command hierarchy is shown below.



Config General

Config **G**eneral — Allows you to modify the general emulator configuration options. It brings up one of the forms shown below. The set of fields within the form change, depending on which monitor is selected. The meaning of each field is as described below.



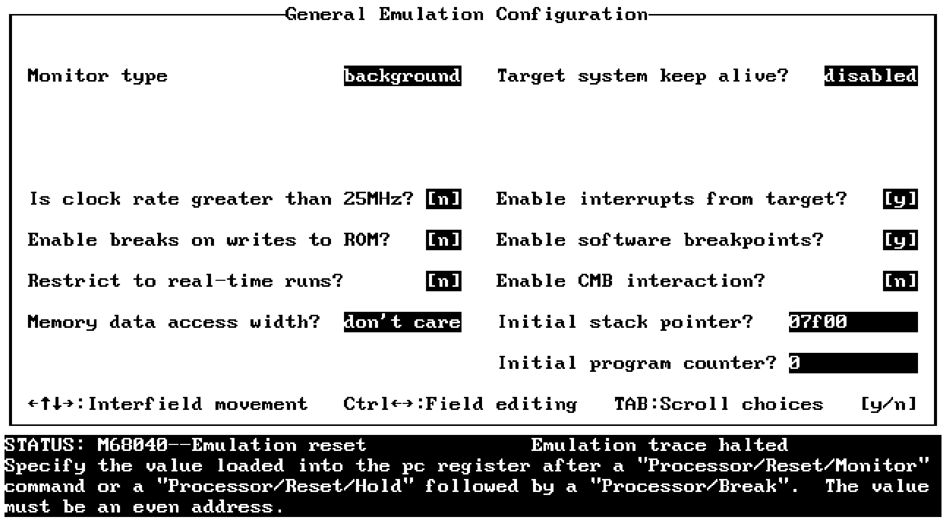
```
─────────────────────General Emulation Configuration─────────────────────

Monitor type                  foreground  Monitor address? 100000

Terminate monitor bus cycles?        [y]  Monitor interrupt priority level?  3

Disable instruction/data caches?     [n]  Disable memory management unit?  [n]

Is clock rate greater than 25MHz?    [n]  Enable interrupts from target?   [y]

Enable breaks on writes to ROM?      [n]  Enable software breakpoints?     [y]

Restrict to real-time runs?          [n]  Enable CMB interaction?          [n]

Memory data access width?     don't care  Initial stack pointer?   37f00

                                          Initial program counter? 0

←↑↓→:Interfield movement   Ctrl←→:Field editing   TAB:Scroll choices   [y/n]
STATUS: M68040-ulation reset                    Emulation trace halted
Specify the value loaded into the pc register after a "Processor/Reset/Monitor"
command or a "Processor/Reset/Hold" followed by a "Processor/Break".  The value
must be an even address.
```

Config General Form, Monitor Foreground Selected

"Monitor type:" Can either be foreground, background, or none.

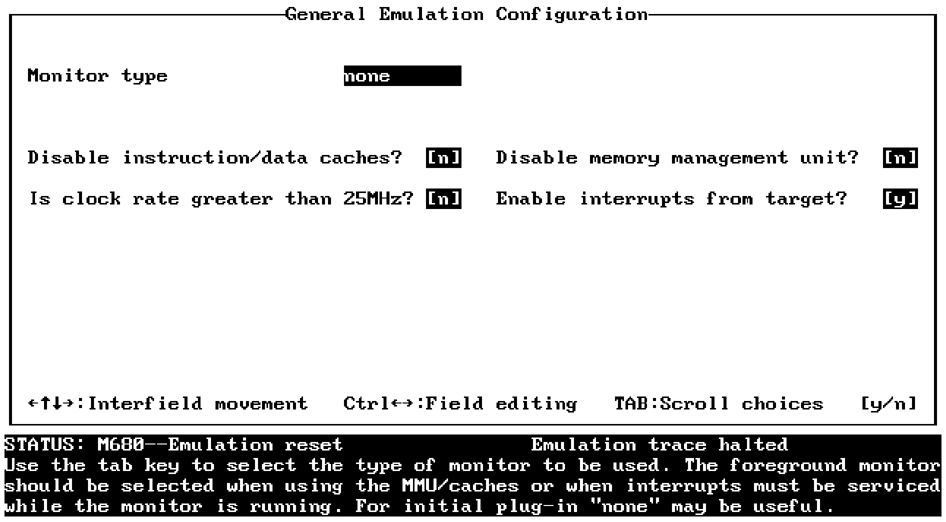
"Terminate monitor bus cycles?:" Allows you to specify whether the emulator or the target system will terminate foreground monitor bus cycles.

- Type **y** if the monitor is in an address range where the target system does not return $\overline{\text{TA}}$ or $\overline{\text{TEA}}$. Monitor cycles will be terminated by emulator-generated cycle-termination signals. Cycle termination signals generated by the target system during access to the foreground monitor, including $\overline{\text{TEA}}$, will be ignored.
- Type **n** if monitor cycles will be terminated when the target system $\overline{\text{TA}}$ and/or $\overline{\text{TEA}}$ signals are asserted. If you select **n** and the monitor is in a range where the target system does not return $\overline{\text{TA}}$ and/or $\overline{\text{TEA}}$, the emulator will stop.



```
                     General Emulation Configuration

Monitor type                 background   Target system keep alive?    disabled



Is clock rate greater than 25MHz? [n]    Enable interrupts from target?    [y]

Enable breaks on writes to ROM?   [n]    Enable software breakpoints?      [y]

Restrict to real-time runs?       [n]    Enable CMB interaction?           [n]

Memory data access width?  don't care    Initial stack pointer?   37f00

                                         Initial program counter? 0

←↑↓→:Interfield movement   Ctrl←→:Field editing   TAB:Scroll choices   [y/n]
STATUS: M68040--Emulation reset              Emulation trace halted
Specify the value loaded into the pc register after a "Processor/Reset/Monitor"
command or a "Processor/Reset/Hold" followed by a "Processor/Break".  The value
must be an even address.
```

Config General Form, Monitor Background Selected



```
                     General Emulation Configuration

Monitor type                 none


Disable instruction/data caches? [n]    Disable memory management unit? [n]

Is clock rate greater than 25MHz? [n]   Enable interrupts from target?  [y]







←↑↓→:Interfield movement   Ctrl←→:Field editing   TAB:Scroll choices   [y/n]
STATUS: M680--Emulation reset                Emulation trace halted
Use the tab key to select the type of monitor to be used. The foreground monitor
should be selected when using the MMU/caches or when interrupts must be serviced
while the monitor is running. For initial plug-in "none" may be useful.
```

Config General Form, Monitor None Selected

"Disable instruction/data caches?:" Allows you to select whether the processor caches will be active. If the caches are disabled, all instruction and data transactions will appear on the processor buses where they can be captured by the emulation-bus analyzer. If the caches are enabled, transactions will be completed in the fastest and most efficient manner, but trace analysis and trace triggering may be difficult.

“Is clock rate greater than 25MHZ?:” Enables one wait state if clock is greater than 25MHZ.

“Enable breaks on writes to ROM?:” Type y if you want the emulator to break to the monitor when a write to ROM is detected.

“Restrict to real-time runs?:” Restricts the emulation to real-time runs. If you enable this configuration item, the emulator will break to the monitor only with the **P**rocessor **R**eset, **P**rocessor **B**reak, **P**rocessor **G**o, and **P**rocessor **S**tep commands.

"Memory data access width?:" Specifies the type of microprocessor cycles that are used to read or write values to memory. Select **don’t care** if you want the emulator to select the optimum access size. Select **bytes** if the emulator should make only 8-bit accesses, **words** if only 16-bit accesses, and **longs** if the emulator should make only 32-bit accesses.

“Monitor address?:" Specify the base address for the foreground monitor.

"Monitor interrupt priority level?:" Enter a number from 0 to 7. Interrupts having values higher than the number you enter here will be recognized by the emulation processor during foreground monitor execution.

Disable memory management unit?:" Select **n** to let the MMU of the emulation processor control placement of the target program in physical memory. The target system will be able to enable and disable the MMU with the MDIS signal. Select **y** to have the emulator disable the MMU with the MDIS signal.

"Target system keep alive?:” Sets the background monitor keep alive address. To enable the target system keep-alive function, type in a hex address with optional function code. To disable the keep-alive function, select disabled.

“Enable interrupts from target?:” To enable target system interrupt signals to reach the emulation processor, type **y**.

“Enable software breakpoints?:” Allows you to enable or disable software breakpoints.

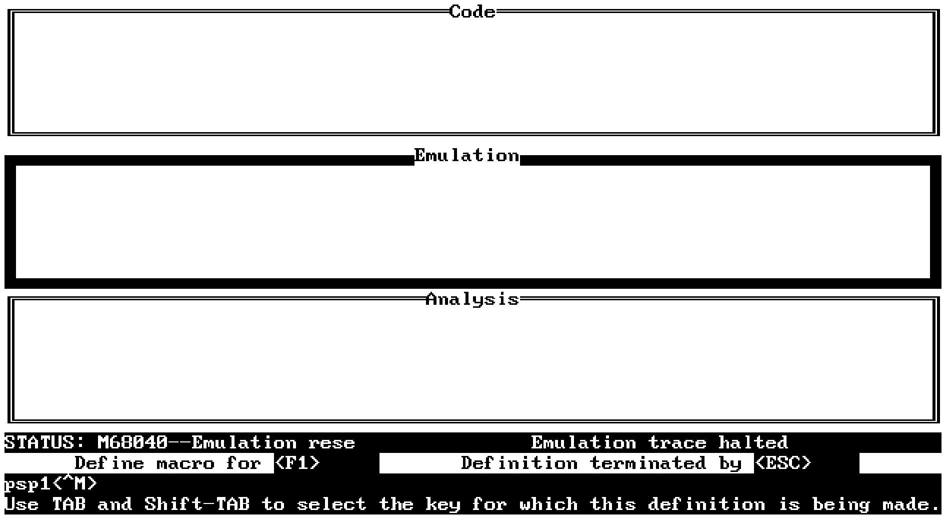
“Enable CMB interaction?:” Allows you to enable or disable multiple emulator start/stop synchronization. Type **y** if you want this emulator to respond to synchronized runs and breaks via the CMB.

“Initial stack pointer?:” Presets the interrupt stack pointer. This field holds the address of the pointer.

“Initial program counter?:” Presets the program counter.

Config Key_Macro

Config **K**ey_Macro — Allows you to define key macros. You can define function keys F1-F10. After selecting the key that will be used to define the macro definition, you can enter the macro definition in the last field.



```
STATUS: M68040--Emulation rese              Emulation trace halted
      Define macro for <F1>            Definition terminated by <ESC>
psp1<^M>
Use TAB and Shift-TAB to select the key for which this definition is being made.
```

Config Key_Macro Form

“Define macro for:” Specifies the function key to which the macro is assigned.

“Definition terminated by:” Specifies the key that allows you to exit the macro definition. You can select <ESC> or one of the function keys.

“(macro definition field):” Contains the keystrokes that execute PC Interface commands. The key you defined in the “Definition terminated by” field is used to exit the macro definition.

Config Load

The **C**onfig **L**oad command lets you load configuration information from a file that was stored previously with the **C**onfig **S**tore command. This loads all information accessed via the **C**onfig **G**eneral command, the **M**emory **M**ap **M**odify command, and the **A**nalysis **F**ormat form.

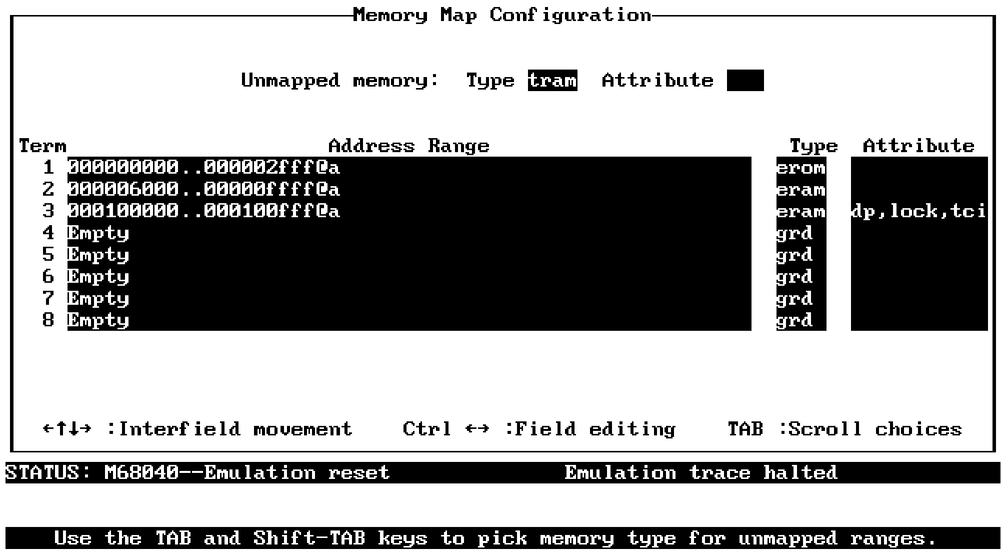
Config Map

These commands are used to describe the memory map.

Config **M**ap **R**eset — Resets the memory map configuration to the default values.

Config **M**ap **M**odify — Opens the following form and allows you to modify the memory map as follows:

"Unmapped memory type:" Specifies the characterization of memory ranges



```
Memory Map Configuration

          Unmapped memory:  Type tram  Attribute

Term                      Address Range                    Type  Attribute
  1 000000000..000002fff@a                                 erom
  2 000006000..00000ffff@a                                 eram
  3 000100000..000100fff@a                                 eram  dp,lock,tci
  4 Empty                                                  grd
  5 Empty                                                  grd
  6 Empty                                                  grd
  7 Empty                                                  grd
  8 Empty                                                  grd

  ←↑↓→ :Interfield movement    Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: M68040--Emulation reset             Emulation trace halted

   Use the TAB and Shift-TAB keys to pick memory type for unmapped ranges.
```

Config Map Modify Form

not mapped. Use the **Tab** or **Shift-Tab** key to select **tram**, **trom**, or **grd**.

“Unmapped memory attribute:” Specifies whether the transfer cache is inhibited or not in unmapped memory ranges. Use the **Tab** key to select **tci** if you want to prevent data that is sent to unmapped ranges from being written into the caches. Leave the field blank if you are willing to let data sent to unmapped ranges also be written into the caches.

“Term (1-8):” You can map up to 8 memory ranges. The resolution of mapped ranges is 256 bytes.

“Address Range:” Specifies the address range to be characterized by the mapper.

"Type:" Characterizes the type of memory to be used for the address range. You can use the **Tab** or **Shift-Tab** key to select **eram**—emulation RAM, **erom**—emulation ROM, **tram**—target RAM, **trom**—target ROM, **grd**—guarded, or **dp**—dual-port memory.

"Attribute:" Specifies **dp**, **lock**, **tci**, or none as the memory attribute. Select **dp** to indicate that this block is to reside in the special 4-Kbyte block of dual-port emulation memory on the probe. Select **lock** to indicate that the target system (TA and TEA) and emulation cycle termination signals should be interlocked for this memory range only. Select **tci** to indicate that read data will not be written into the instruction and/or data caches for transactions within this memory range.

Config Store

Config **S**tore — Saves the PC Interface configuration into a named file.

Config Trigger

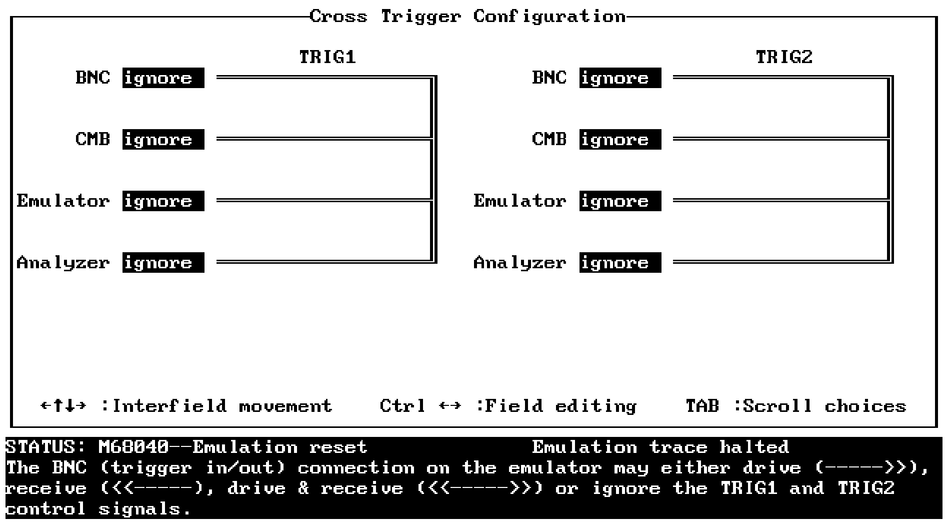
Config **T**rigger — Defines the devices that drive and receive the internal TRIG1 and TRIG2 signals. The analyzers can drive these internal signals, which can then drive signals on the CMB and BNC ports. The analyzers can be armed (activated) on the reception of these signals. Emulator execution can break into the monitor on reception of these signals. The form and fields are as follows.

"BNC:" The **Tab** or **Shift-Tab** key is used to select **drive**, **receive**, **both**, or **ignore**.

"CMB:" The **Tab** or **Shift-Tab** key is used to select **drive**, **receive**, **both**, or **ignore**.

"Emulator:" The **Tab** or **Shift-Tab** key is used to select **receive** or **ignore**. Upon receipt, the emulator will break back to monitor operation.

"Analyzer:" The **Tab** or **Shift-Tab** key is used to select **drive**, **receive**, or **ignore**.

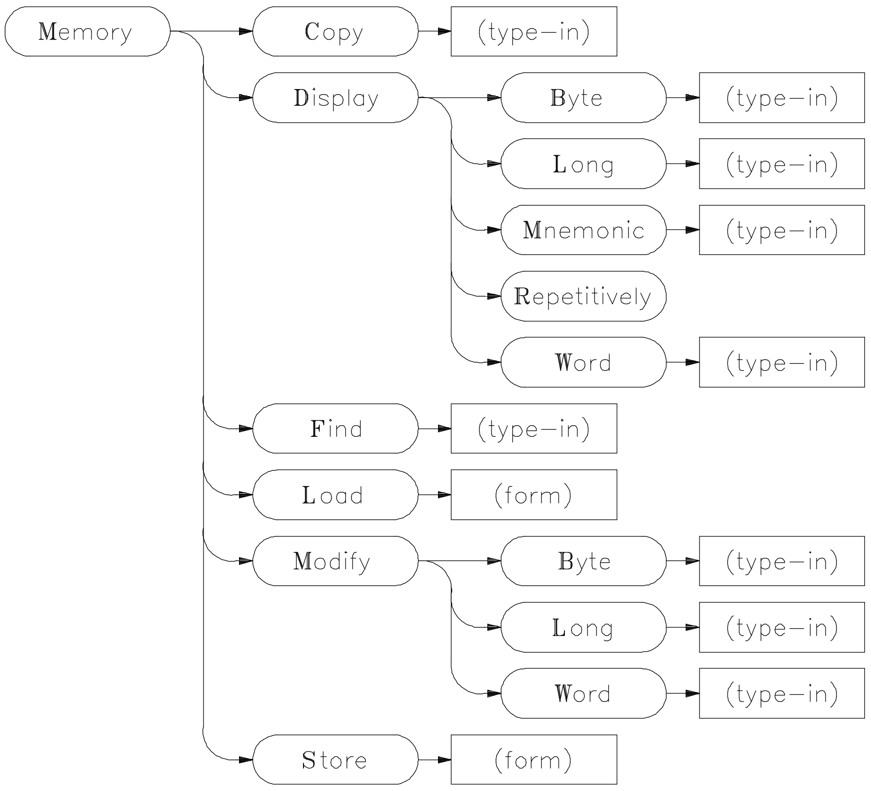


```
Cross Trigger Configuration

                    TRIG1                                 TRIG2
     BNC ignore                             BNC ignore
     CMB ignore                             CMB ignore
Emulator ignore                        Emulator ignore
Analyzer ignore                        Analyzer ignore

 ←↑↓→ :Interfield movement    Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: M68040--Emulation reset               Emulation trace halted
The BNC (trigger in/out) connection on the emulator may either drive (----->>),
receive (<<-----), drive & receive (<<----->>) or ignore the TRIG1 and TRIG2
control signals.
```

Config Trigger Form

Memory Commands

The **M**emory command hierarchy is shown below. These commands are all concerned with memory operation.



Memory Copy

The **M**emory **C**opy command copies the contents of one range of memory to another.

Memory Display

Memory **D**isplay **B**yte — Displays, in byte format (8-bit), the contents of the memory locations specified.

Memory **D**isplay **M**nemonic — Displays, in disassembled mnemonic format, the contents of the memory locations specified.

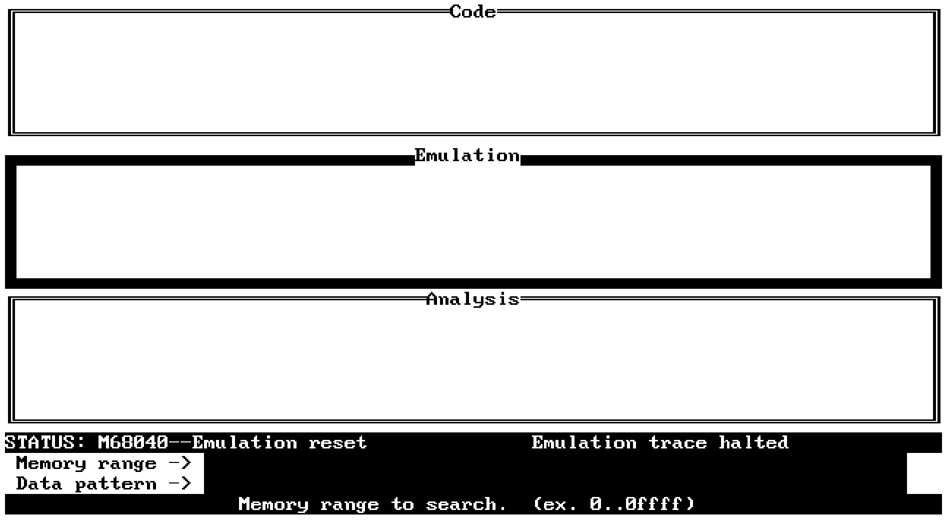
Memory **D**isplay **R**epetitively — Repetitively displays the contents of the memory locations specified by the last memory display command.

Memory **D**isplay **W**ord — Displays, in word format (16-bit), the contents of the memory locations specified.

Memory **D**isplay **L**ong — Displays, in word format (32-bit), the contents of the memory locations specified.

Memory Find

Memory **F**ind — Searches a range of memory for a data pattern. The fields are:



```
STATUS: M68040--Emulation reset            Emulation trace halted
 Memory range ->
 Data pattern ->
                      Memory range to search.  (ex. 0..0ffff)
```

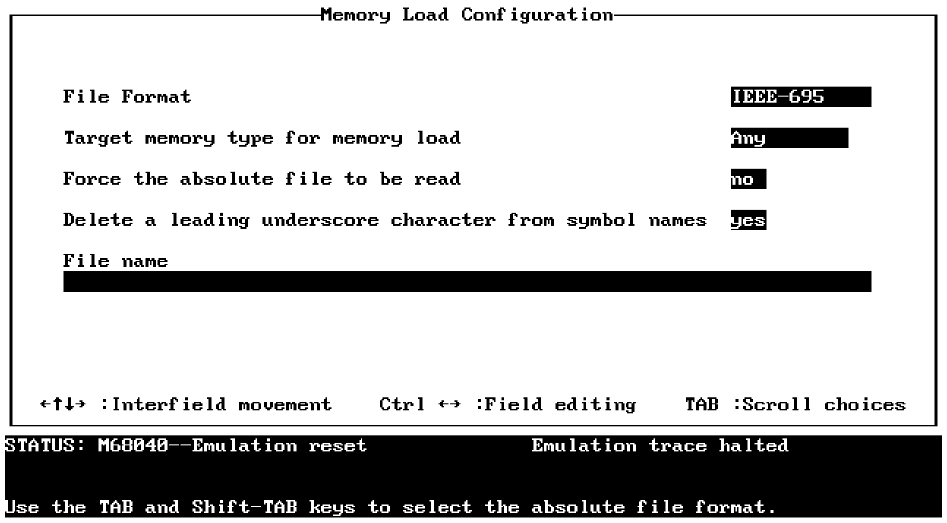
Memory Find Form

"Memory range:" Specifies the range of memory in which to search for the data pattern

"Data pattern:" Specifies the data pattern to be searched for. Up to eight byte of data are expected. You can search for ASCII strings enclosed by string delimiters.

Memory Load

Memory **L**oad — Accesses the following form which in turn allows you to load an absolute file into memory.



```
Memory Load Configuration

File Format                                              IEEE-695

Target memory type for memory load                       Any

Force the absolute file to be read                       no

Delete a leading underscore character from symbol names  yes

File name


←↑↓→ :Interfield movement    Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: M68040--Emulation reset            Emulation trace halted

Use the TAB and Shift-TAB keys to select the absolute file format.
```

Memory Load Form

“File Format:” Specifies the format of the absolute file to be loaded. You can use the **Tab** or **Shift-Tab** key to select **IEEE-695**, **HP64000**, **Raw HP64000**, **Intel_Hex**, **Motorola_Hex**, or **Ext_Tek_Hex**.

When you select the **IEEE-695** option, an absolute file name is expected. The extension of an IEEE-695 absolute file cannot be “.HPA” or “.HPS”.

When you select the **HP64000** option, a linker symbol file name is expected. The extension of a HP 64000 linker symbol file is “.L”.

“Target memory type for memory load:” Specifies the type of memory into which the absolute file should be loaded. You can use the **Tab** or **Shift-Tab** key to select emulation memory, target system memory, both, or a custom foreground monitor.

“Force the absolute file to be read:” When loading IEEE-695 or HP64000 format absolute files, this option can force the absolute file reader to be run. If you don’t force the absolute file reader to be run, it may or may not be run depending on the creation date/time of the memory image (.HPA) and symbol (.HPS) files. You can select: no, yes.

“Delete a leading underscore character from symbol names:” When loading IEEE-695 files, you can cause any leading underscores to be deleted from symbol names. You can use the **Tab** or **Shift-Tab** key to select: **yes** or **no**.

“File name:” Specifies the name of the absolute file.

Memory Modify

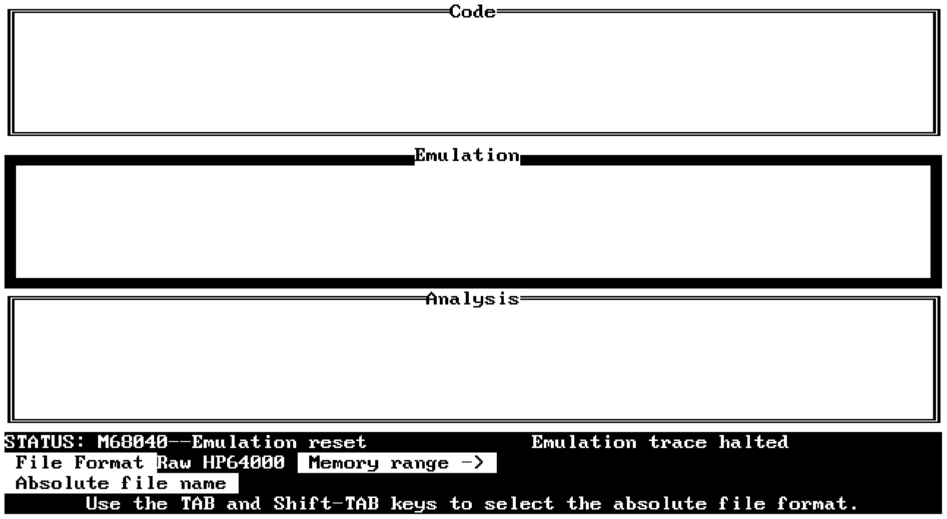
Memory **M**odify **B**yte — Modifies, with byte (8-bit) values, the contents of the memory locations specified to the values specified.

Memory **M**odify **W**ord — Modifies, with short (16-bit) values, the contents of the memory locations specified to the values specified.

Memory **M**odify **L**ong — Modifies, with word (32-bit) values, the contents of the memory locations specified to the values specified.

Memory Store

Memory **S**tore — Saves the contents of memory locations into an absolute file. The fields are as shown:



```
STATUS: M68040--Emulation reset            Emulation trace halted
 File Format Raw HP64000  Memory range ->
 Absolute file name
       Use the TAB and Shift-TAB keys to select the absolute file format.
```

Memory Store Form

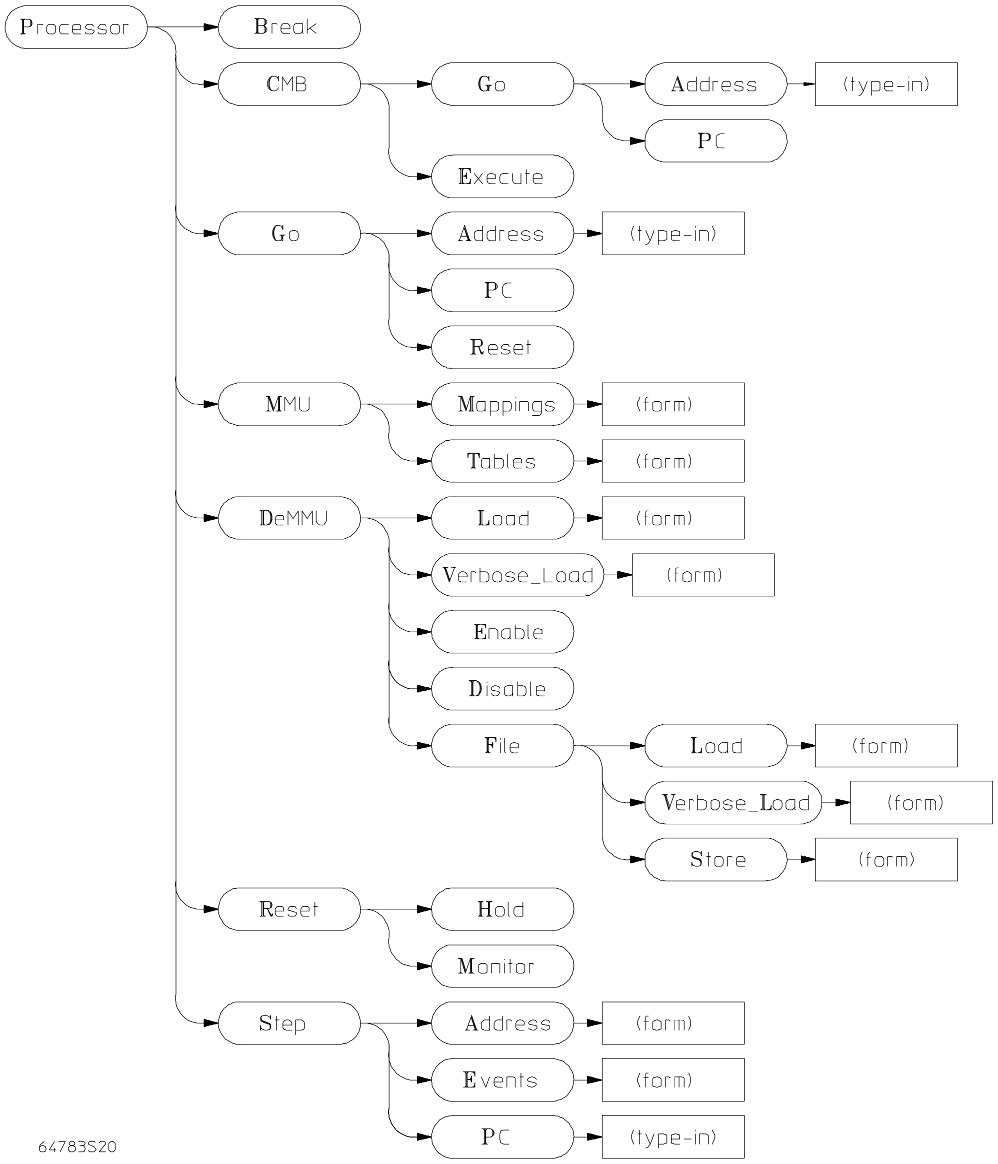
“file format:” Sets the format of the absolute file. You can use the **Tab** or **Shift-Tab** key to select **Raw HP64000**, **Intel_Hex**, **Motorola_Hex**, or **Ext_Tek_Hex**.

“memory range:” Specifies the range of memory locations to be saved to the absolute file.

“absolute file name:” Specifies the name of the absolute file.

Processor Commands

The **P**rocessor command hierarchy is shown below. These commands are all concerned with the operation of the processor.



Processor Break

Processor **B**reak — Breaks emulator execution into the monitor.

Processor CMB

Processor **C**MB **E**xecute — Issues a CMB EXECUTE signal.

Processor **C**MB **G**o **A**ddress — Specifies the address for the emulator to execute from on the reception of the CMB EXECUTE signal.

Processor **C**MB **G**o **P**c — Specifies that the emulator execute from the current program counter address upon reception of the CMB EXECUTE signal.

Processor Go

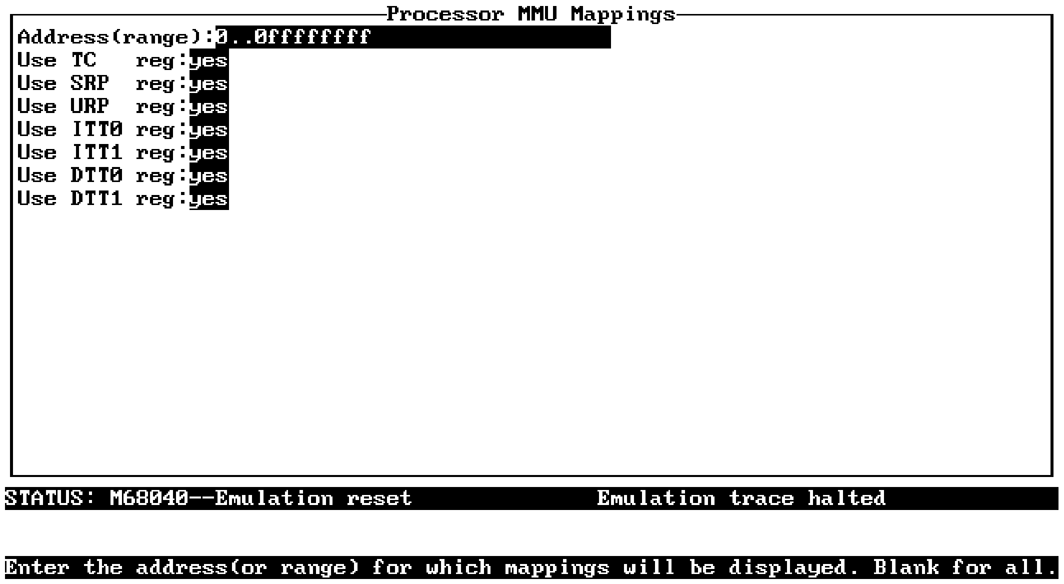
Processor **G**o **A**ddress — Starts emulator execution of the user program at the specified address.

Processor **G**o **P**c — Starts emulator execution of the user program at the current program counter address.

Processor **G**o **R**eset — Runs the emulator from the reset state.

Processor MMU

Processor **M**MU **M**appings — Instructs the emulator to list the present logical-to-physical mappings controlled by the TC, SRP, URP, ITT0, ITT1, DTT0, and DTT1 registers, and the present MMU tables. Opens the following form:



```
Processor MMU Mappings
Address(range):0..0ffffffff
Use TC   reg:yes
Use SRP  reg:yes
Use URP  reg:yes
Use ITT0 reg:yes
Use ITT1 reg:yes
Use DTT0 reg:yes
Use DTT1 reg:yes

STATUS: M68040--Emulation reset             Emulation trace halted

Enter the address(or range) for which mappings will be displayed. Blank for all.
```

Processor MMU Mappings Form

Address(range):" Specifies a logical address reference for the MMU information to be displayed.

"Use TC reg:" Specifies a value to be used as the TC (translation control) register when reading the tables and showing the address mappings. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

"Use SRP reg:" Specifies a value to be used as the SRP (supervisor root pointer) when reading the tables and showing the address mappings. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

"Use URP reg:" Specifies a value to be used as the URP (user root pointer) when reading the tables and showing the address mappings. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

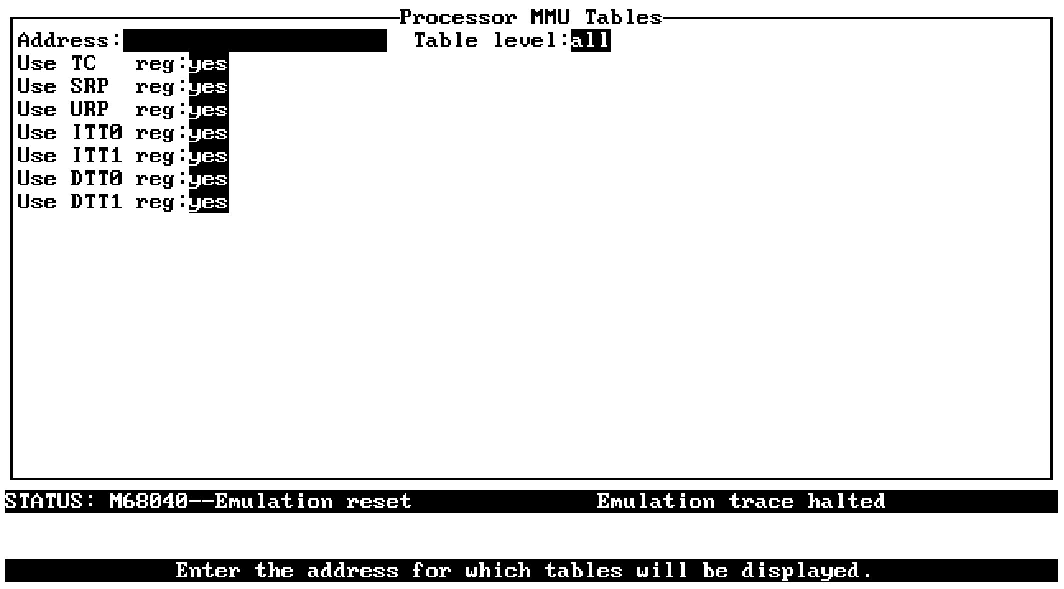
"Use ITT0 reg:" Specifies a value to be used as the ITT0 (instruction transparent translation register 0) when reading the tables and showing the address mappings. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

"Use ITT1 reg:" Specifies a value to be used as the ITT1 (instruction transparent translation register 1) when reading the tables and showing the address mappings. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

"Use DTT0 reg:" Specifies a value to be used as the DTT0 (data transparent translation register 0) when reading the tables and showing the address mappings. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

"Use DTT1 reg:" Specifies a value to be used as the DTT1 (data transparent translation register 1) when reading the tables and showing the address mappings. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

Processor **M**MU **T**ables — Instructs the emulator to show the translation path through the MMU tables for a single logical address, or show details of one table used in the translation of a single logical address. Opens the following form:



```
─────────────────────────────Processor MMU Tables─────────────────────────────
Address:                      Table level:all
Use TC   reg:yes
Use SRP  reg:yes
Use URP  reg:yes
Use ITT0 reg:yes
Use ITT1 reg:yes
Use DTT0 reg:yes
Use DTT1 reg:yes

STATUS: M68040--Emulation reset              Emulation trace halted

             Enter the address for which tables will be displayed.
```

Processor MMU Tables Form

Address(range):" Specifies a logical address reference for the MMU information to be displayed.

"Table level:" Specifies display of the content of the selected translation table at the point referenced by the logical address specified in the form.

"Use TC reg:" Specifies a value to be used as the TC (translation control) register when reading the tables and showing the details of the selected table. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

"Use SRP reg:" Specifies a value to be used in place of the present content of the SRP (supervisor root pointer) when reading the tables and showing the details of the selected table. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

"Use URP reg:" Specifies a value to be used as the URP (user root pointer) when reading the tables and showing the details of the selected table. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

"Use ITT0 reg:" Specifies a value to be used as the ITT0 (instruction transparent translation register 0) when reading the tables and showing the details of the selected table. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

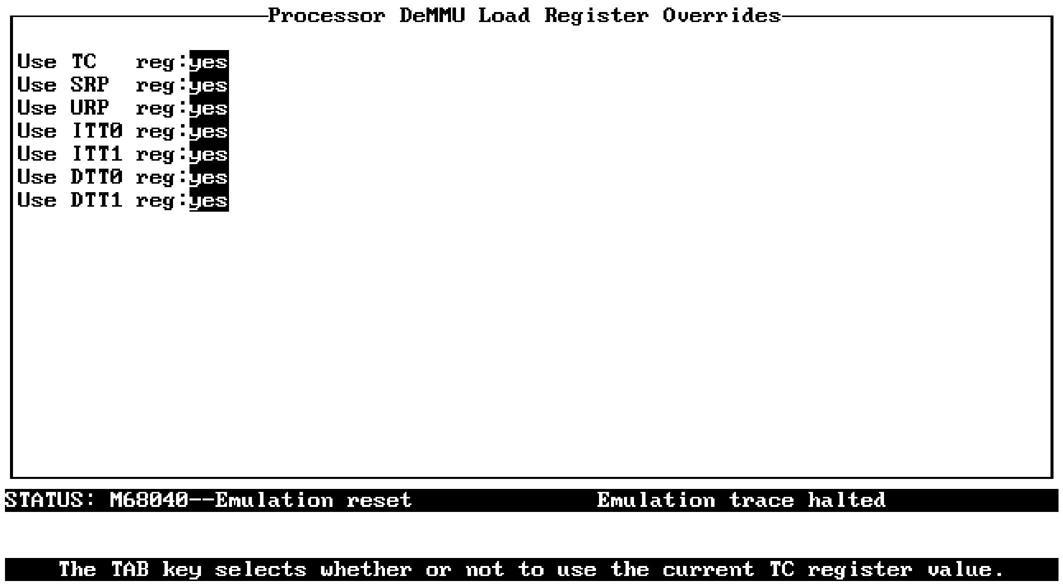
"Use ITT1 reg:" Specifies a value to be used as the ITT1 (instruction transparent translation register 1) when reading the tables and showing the details of the selected table. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

"Use DTT0 reg:" Specifies a value to be used as the DTT0 (data transparent translation register 0) when reading the tables and showing the details of the selected table. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

"Use DTT1 reg:" Specifies a value to be used as the DTT1 (data transparent translation register 1) when reading the tables and showing the details of the selected table. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

Processor **D**eMMU **L**oad and Verbose_Load — Causes the emulator to read the MMU registers and interpret the translation tables to load the deMMUer. Opens the following form:

Processor **D**eMMU **V**erbose_Load — This sets the verbose mode for the deMMUer load function. A list is displayed of the physical address ranges that will be reverse-translated by the deMMUer. Opens the following form:



```
Processor DeMMU Load Register Overrides

Use TC   reg:yes
Use SRP  reg:yes
Use URP  reg:yes
Use ITT0 reg:yes
Use ITT1 reg:yes
Use DTT0 reg:yes
Use DTT1 reg:yes

STATUS: M68040--Emulation reset            Emulation trace halted

  The TAB key selects whether or not to use the current TC register value.
```

Processor DeMMU Load and Verbose_Load Form

"Use TC reg:" Specifies a value to be used as the TC (translation control) register when reading the MMU registers and interpreting the translation tables to load the deMMUer. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

"Use SRP reg:" Specifies a value to be used in place of the present content of the SRP (supervisor root pointer) when reading the MMU registers and interpreting the translation tables to load the deMMUer. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

"Use URP reg:" Specifies a value to be used as the URP (user root pointer) when reading the MMU registers and interpreting the translation tables to load the deMMUer. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

"Use ITT0 reg:" Specifies a value to be used as the ITT0 (instruction transparent translation register 0) when reading the MMU registers and interpreting the translation tables to load the deMMUer. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

"Use ITT1 reg:" Specifies a value to be used as the ITT1 (instruction transparent translation register 1) when reading the MMU registers and interpreting the translation tables to load the deMMUer. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

"Use DTT0 reg:" Specifies a value to be used as the DTT0 (data transparent translation register 0) when reading the MMU registers and interpreting the translation tables to load the deMMUer. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

"Use DTT1 reg:" Specifies a value to be used as the DTT1 (data transparent translation register 1) when reading the MMU registers and interpreting the translation tables to load the deMMUer. If you select **yes**, the current register value is used. If you select **no**, you can type in a value to be used.

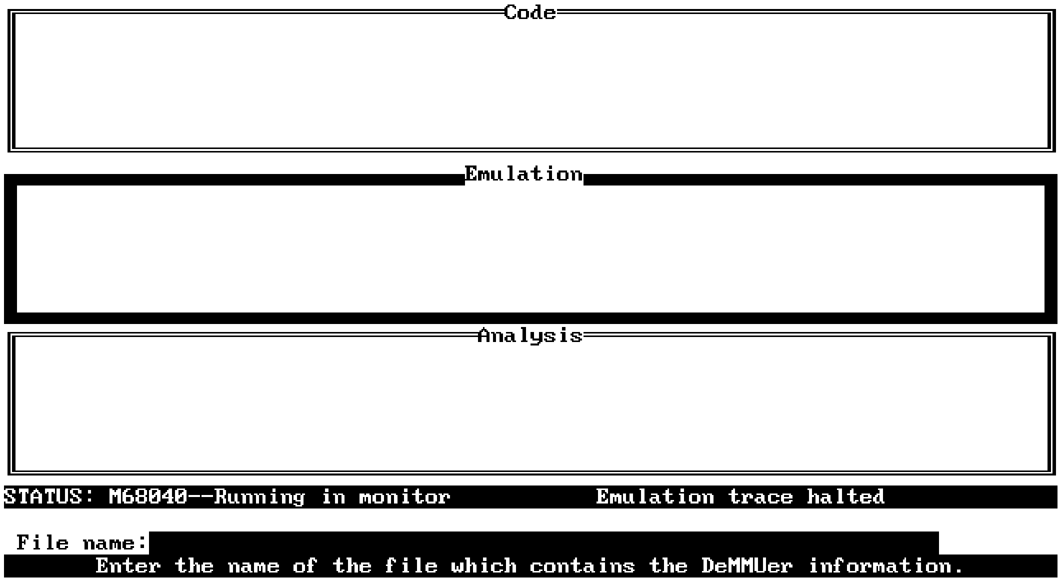
Processor **D**eMMU **E**nable — This causes your analyzer to receive logical addresses translated by the deMMUer according to the translation tables in place, or the demmuer file content, when you last loaded the deMMUer.

Processor **D**eMMU **D**isable — This causes your analyzer to receive 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.

Processor **D**eMMU **F**ile **L**oad — This lets you load a file that contains an MMU setup. The deMMUer will configure itself to reverse translate addresses for the MMU setup in this file instead of the MMU setup in the emulation processor.

Processor **D**eMMU **F**ile **V**erbose_Load — This lets you load a file that contains an MMU setup. The deMMUer will configure itself to reverse translate addresses for the MMU setup in this file instead of the MMU setup in the emulation processor. The display will show you the ranges of address that will be reverse translated by the deMMUer.

Processor **D**eMMU **F**ile **S**tore — Use this feature to overcome problems when the deMMUer is unable to reverse translate all of the address ranges contained in the present setup of the MMU. This command lets you store a file that contains the present setup of the MMU. You can use an editor of your choice to remove address ranges that you do not need to have reverse translated by the deMMUer. Then you can load this file (using one of the **P**rocessor **D**eMMU **F**ile ... commands above). The deMMUer will configure itself to reverse translate addresses for the MMU setup in this file instead of the MMU setup in the emulation processor.



```
STATUS: M68040--Running in monitor          Emulation trace halted

 File name:
       Enter the name of the file which contains the DeMMUer information.
```

Processor DeMMU File Load/Verbose_Load/Store Form

"File name:" Allows you to enter a DOS file name, along with its complete path name, to either store the present MMU setup or to be loaded into the deMMUer to configure it to provide reverse translations.

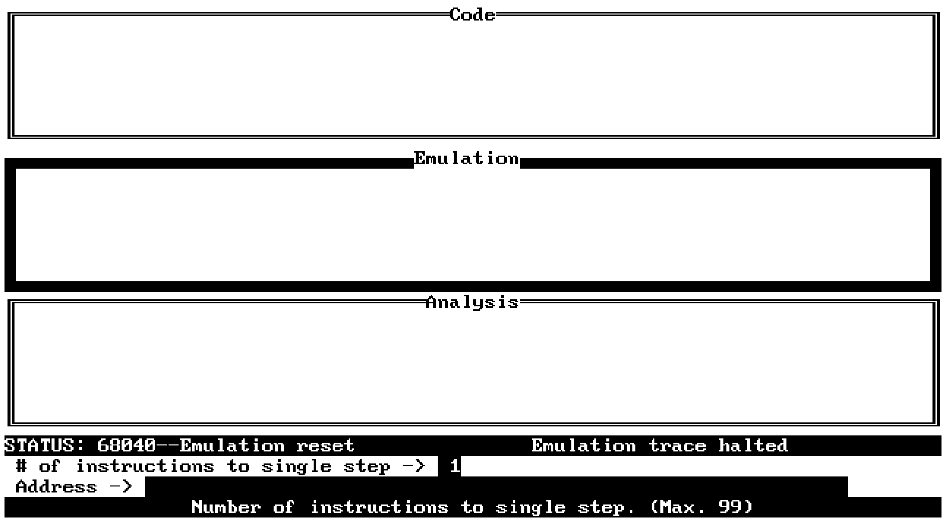
Processor Reset

Processor **R**eset **H**old — Resets the emulation processor, and holds the emulation processor in the reset state.

Processor **R**eset **M**onitor — Resets the emulation processor, and breaks emulator execution into the monitor.

Processor Step

Processor **S**tep **A**ddress — Instructs the emulator to step a number of instructions (1 to 99) from the address specified. Opens the following form:



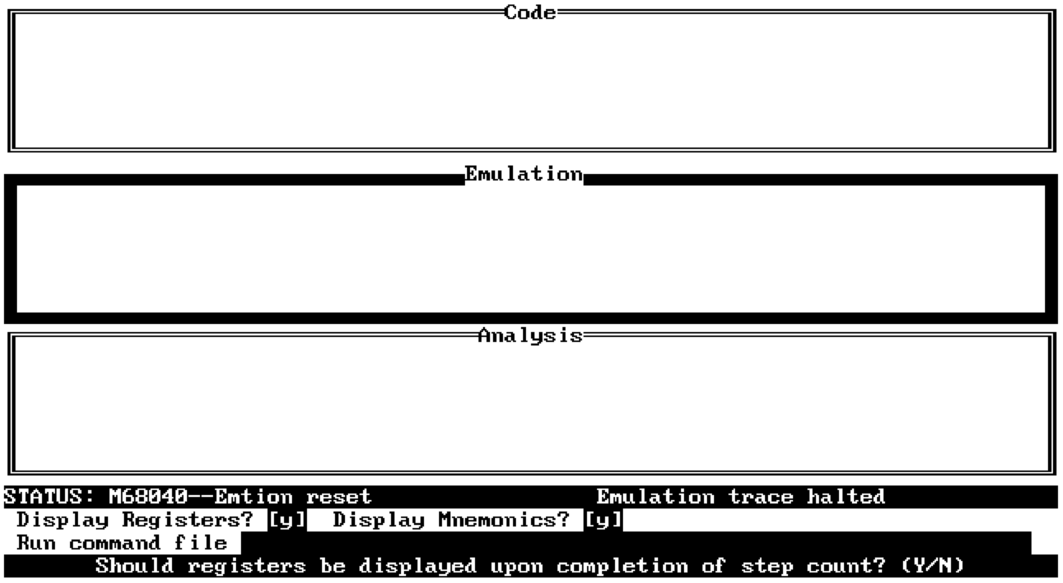
```
STATUS: 68040--Emulation reset                Emulation trace halted
 # of instructions to single step ->  1
 Address ->
                Number of instructions to single step. (Max. 99)
```

Processor Step Address Form

"# of instructions to single step:" Specify the number of instructions to step. The step default is 1.

"Address:" Starting address of instruction stepping.

Processor **S**tep **E**vents — Specifies whether registers are displayed, whether instruction mnemonics are displayed, and whether a command file should be executed on the completion of a step instruction.



```
STATUS: M68040--Emtion reset                 Emulation trace halted
 Display Registers? [y]  Display Mnemonics? [y]
 Run command file
       Should registers be displayed upon completion of step count? (Y/N)
```

Processor Step Events Form

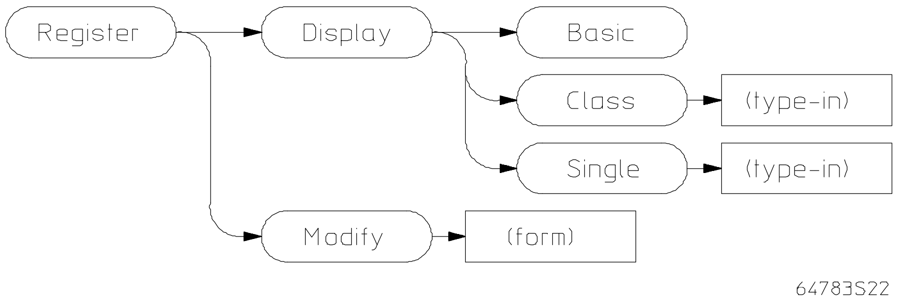
"Display Registers?:" Specifies whether registers should be displayed after a step command. You can select: y, n.

"Display Mnemonics?:" Specifies whether instruction mnemonics should be displayed after a step command. You can select: y, n.

"Run command file:" Specifies the name of a command file to be run on the completion of a step instruction.

Processor **S**tep **P**C — Instructs the emulator to step a number of instructions (1 to 99) from the current program counter address.

Register Commands



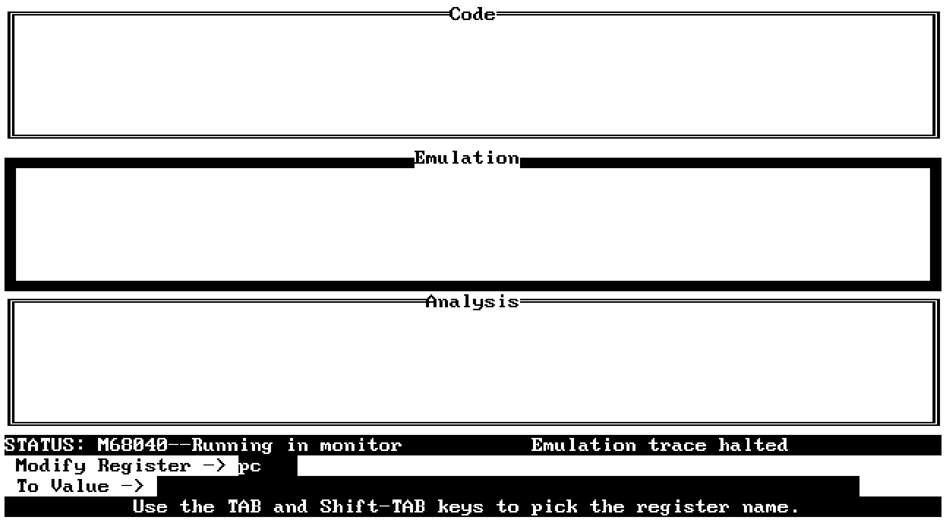
Register Display

Register **D**isplay **A**ll — Displays the contents of all the registers.

Register **D**isplay **S**ingle — Displays the contents of a specified registers.

Register Modify

Register **M**odify — Modifies the contents of the register specified to the value specified. You specify the register and its contents with the following form and fields .



```
STATUS: M68040--Running in monitor          Emulation trace halted
 Modify Register -> pc
 To Value ->
           Use the TAB and Shift-TAB keys to pick the register name.
```

Register Modify Form

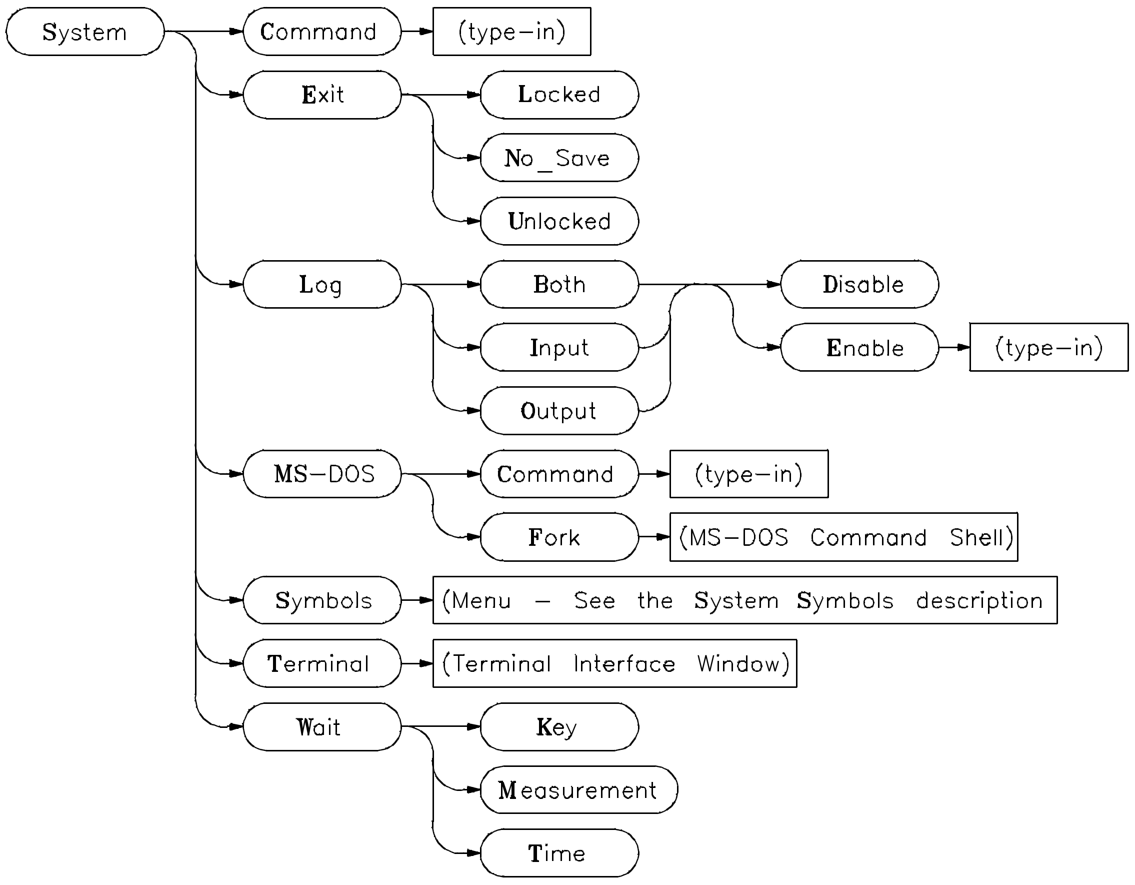
"Modify Register:" — You can use the **Tab** or **Shift-Tab** key to select registers as shown in the following table.

Register Class	**Register Names**
Basic	**pc, st, usp, isp, msp, cacr, d0..d7, a0..a7, vbr, dfc, sfc**
Fpu	**fpcr, fpsr, fpiar, fp0..fp7**
Mmu	**dtt0, dtt1, itt0, itt1, mmusr, tc, srp, urp**

"To Value:" — Specifies the value that the register chosen in the "Modify Register" field should be set to.

System Commands

The **S**ystem command hierarchy is shown below. The commands are all concerned with system operation.



System Command

System **C**ommand — Executes PC Interface commands from the named command file.

System Exit

System **E**xit **L**ocked — Exits the PC Interface, and specifies that the current configuration will be present the next time you start up the PC Interface.

System **E**xit **N**o_save — Exits the PC Interface and specifies that the configuration present when you entered the PC Interface will be present the next time you start up the PC Interface.

System **E**xit **U**nlocked — Exits the PC Interface without saving the current configuration.

System MS-DOS

System **M**S-DOS **C**ommand — Allows you to enter MS-DOS commands from the PC Interface.

System **M**S-DOS **F**ork — Forks an MS-DOS shell. You must enter the MS-DOS EXIT command to return to the PC Interface.

Note that when using the "**S**ystem **M**S-DOS ..." commands, do not run any programs that modify the RS-232 ports. Also, you should not run any programs that consume memory without returning it to the host computer. Some examples include: operating HP AdvanceLink, starting a LAN (Local Area Network) connection, and copying files from another computer system. Also, you will not be able to execute any commands that require large amounts of system memory, such as compilers, complex text editors, and so on. You can use the MS-DOS CHKDSK command to determine the amount of free memory available after you fork the system.

System Log

System **L**og **I**nput **D**isable — Turns off the logging of PC interface commands and command input.

System **L**og **I**nput **E**nable — Causes subsequent PC Interface commands and command input to be logged to the specified file. The saved commands may be used later as command files.

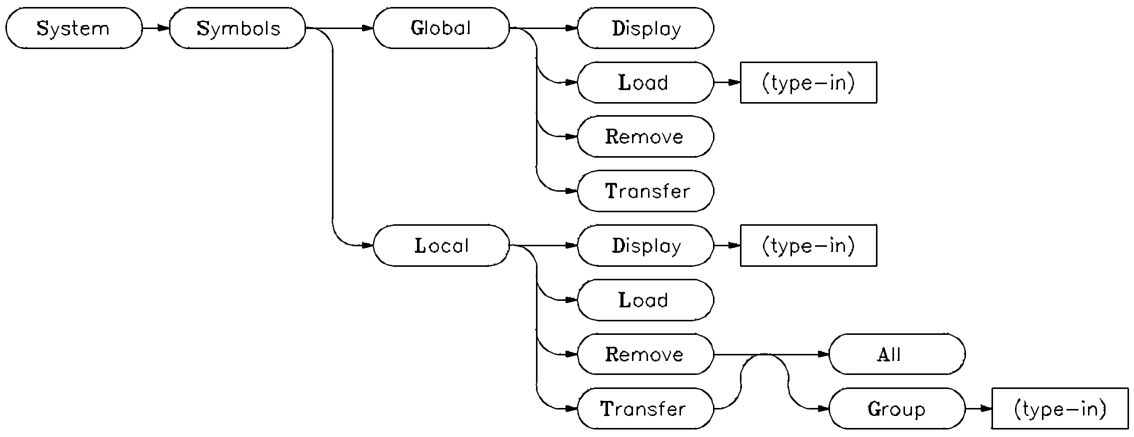
System **L**og **O**utput **D**isable — Turns off the logging of the output from PC Interface commands.

System **L**og **O**utput **E**nable — Causes the output of subsequent PC Interface to be logged to the specified file. This command is useful if you want to save the results of a series of PC Interface commands.

System **L**og **B**oth **D**isable — Turns off the logging of PC Interface commands, command input, and command output.

System **L**og **B**oth **E**nable — Causes subsequent PC Interface commands, command input, and the output from the commands to be logged to the specified file. This command is useful if you want to save the input and output of a series of PC Interface commands.

System Symbols



System **S**ymbols **G**lobal **D**isplay — Displays global symbols.

System **S**ymbols **G**lobal **L**oad — Associates the PC interface with a symbol database

System **S**ymbols **G**lobal **R**emove — Removes global symbols from the emulator.

System **S**ymbols **G**lobal **T**ransfer — Transfers global symbols into the emulator so that they can appear in mnemonic memory displays and in trace displays.

System **S**ymbols **L**ocal **D**isplay — Displays local symbols from a specified module name.

System **S**ymbols **L**ocal **L**oaded — Display the local symbols that were previously transferred to the emulator.

System **S**ymbols **L**ocal **R**emove **A**ll — Removes all local symbols previously transferred into the emulator.

System **S**ymbols **L**ocal **R**emove **G**roup — Removes the local symbols from the specified modules that were previously transferred into the emulator.

System **S**ymbols **L**ocal **T**ransfer **A**ll — Transfers all local symbols into the emulator so that they can appear in mnemonic memory displays and in trace displays.

System **S**ymbols **L**ocal **T**ransfer **G**roup — Transfers the local symbols from the specified modules into the emulator so that they can appear in mnemonic memory displays and in trace displays.

System Terminal

System **T**erminal — Provides a terminal emulation window that allows you to access the HP 64700's Terminal Interface. Use the Terminal Interface **help** or **?** command for information. Enter **<Ctrl>** \ to abort the Terminal Interface. Use **<Ctrl> A** to cycle through the active windows.

System Wait

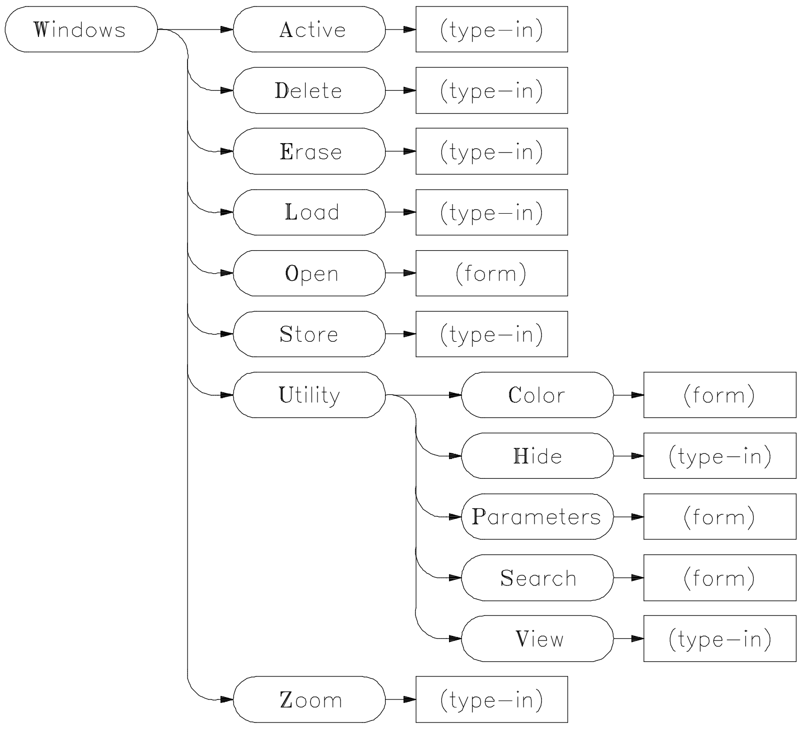
System **W**ait **K**ey — Causes the PC Interface to wait until any key is pressed before allowing further entry of commands. This command is typically used in command files to stop the execution of PC Interface commands at a certain point.

System **W**ait **M**easurement — Causes the PC Interface to wait until the emulation analyzer measurement is complete before allowing further entry of commands. This command is typically used in command files to delay the execution of PC Interface commands until after the trace measurement is complete.

System **W**ait **T**ime — Causes the PC Interface to wait a specified number of seconds before allowing further entry of commands. This command is typically used in command files to delay the execution of PC Interface commands at a certain point.

Window Commands

The **W**indow command hierarchy is shown below. The commands are all concerned with the operation of windows.



Window Active

Window **A**ctive — Selects the active window by entering the window name. You can also use **<Ctrl> A** to cycle through windows.

Window Delete

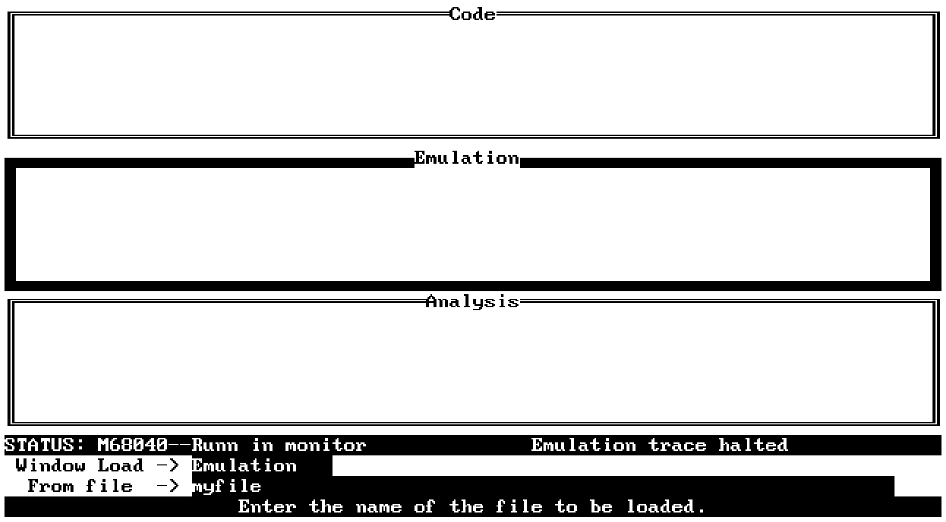
Window **D**elete — Deletes the user-defined window whose name is typed in.

Window Erase

Window **E**rase — Erases the contents of the window whose name is typed in. You can also use **<Ctrl> e** to erase the contents of the active window.

Window Load

Window **L**oad — Loads the window with the contents of the specified text file. This should not be confused with loading emulator memory. The fields are as follows:



```
STATUS: M68040--Runn in monitor              Emulation trace halted
 Window Load -> Emulation
  From file  -> myfile
                    Enter the name of the file to be loaded.
```

Window Load Form

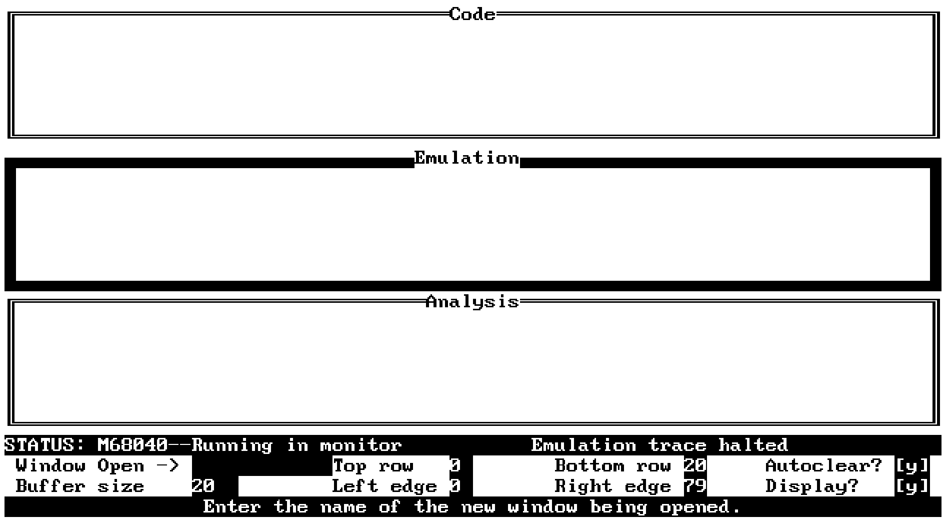
“Window Load:” The **Tab** or **Shift-Tab** key is used to select the window to load from the list of active windows.

“From File:” The **Tab** or **Shift-Tab** key can be used to select the name of the file to load into the specified window. Or, the name of the file can be typed in.

Window Open

Window **O**pen — Creates and defines parameters for a user-defined window. The fields are:

“Window Open:” Specifies the name of the user-defined window to open.



```
STATUS: M68040--Running in monitor          Emulation trace halted
 Window Open ->            Top row    0         Bottom row 20      Autoclear? [y]
 Buffer size     20        Left edge  0         Right edge 79      Display?   [y]
                 Enter the name of the new window being opened.
```

Window Open Form

"Top row:" Specifies the row number (1 to 20) of the window's top row.

"Bottom row:" Specifies the row number (1 to 20) of the window's bottom row.

"Autoclear?:" Specifies whether the contents of the window are cleared prior to each time it is written to. You can select: y, n.

"Buffer size:" Specifies the number of rows (20 to 1030) of buffer space that can be used by the window.

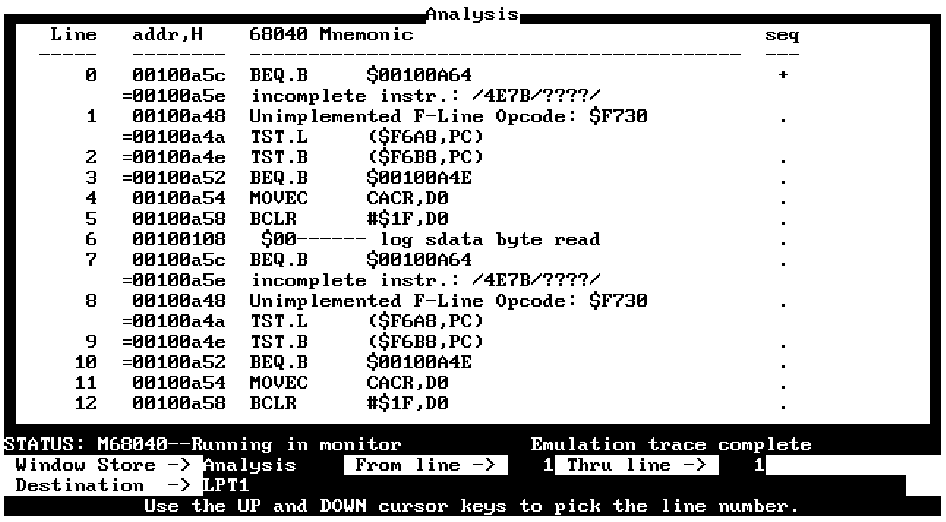
"Left edge:" Specifies the column number (0 to 79) of the window's left edge.

"Right edge:" Specifies the column number (0 to 79) of the window's right edge.

"Display?:" Specifies whether the window should be displayed after it is opened. You can select: **y** or **n**.

Window Store

Window **S**tore — Saves selected lines from the window to a specified file. This command can be used to print window contents by specifying as the destination the name of the printer device (LPT1 for example). The fields are:



```
                                    Analysis
  Line    addr,H    68040 Mnemonic                                  seq
  -----   --------  ----------------------------------------------  ---
      0   00100a5c  BEQ.B      $00100A64                             +
         =00100a5e  incomplete instr.: /4E7B/????/
      1   00100a48  Unimplemented F-Line Opcode: $F730               .
         =00100a4a  TST.L      ($F6A8,PC)
      2  =00100a4e  TST.B      ($F6B8,PC)                            .
      3  =00100a52  BEQ.B      $00100A4E                             .
      4   00100a54  MOVEC      CACR,D0                               .
      5   00100a58  BCLR       #$1F,D0                               .
      6   00100108   $00------ log sdata byte read                   .
      7   00100a5c  BEQ.B      $00100A64                             .
         =00100a5e  incomplete instr.: /4E7B/????/
      8   00100a48  Unimplemented F-Line Opcode: $F730               .
         =00100a4a  TST.L      ($F6A8,PC)
      9  =00100a4e  TST.B      ($F6B8,PC)                            .
     10  =00100a52  BEQ.B      $00100A4E                             .
     11   00100a54  MOVEC      CACR,D0                               .
     12   00100a58  BCLR       #$1F,D0                               .

STATUS: M68040--Running in monitor          Emulation trace complete
 Window Store -> Analysis     From line ->    1 Thru line ->    1
 Destination  -> LPT1
            Use the UP and DOWN cursor keys to pick the line number.
```

Window Store Form

“Window Store:” The **Tab** or **Shift-Tab** key can be used to select the name of the window.

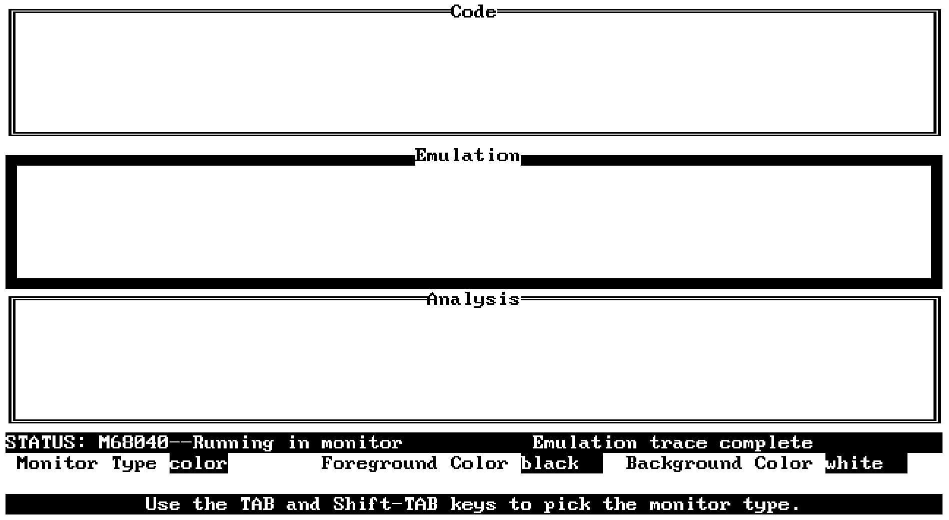
“From Line:” Specifies the first window line in the range of lines to be stored.

“Thru Line:” Specifies the last window line in the range of lines to be stored.

“Destination:” Specifies the destination file that the window lines are to be stored in.

Window Utility

Window **U**tility **C**olor — Specifies the monitor type and the foreground and background colors if the monitor type is color. The fields are as follows:



```
STATUS: M68040--Running in monitor          Emulation trace complete
 Monitor Type color        Foreground Color black     Background Color white

            Use the TAB and Shift-TAB keys to pick the monitor type.
```

Window Utility Color Form

“Monitor:” The **Tab** or **Shift-Tab** key can be used to select the monitor type, **color** or **monochrome**.

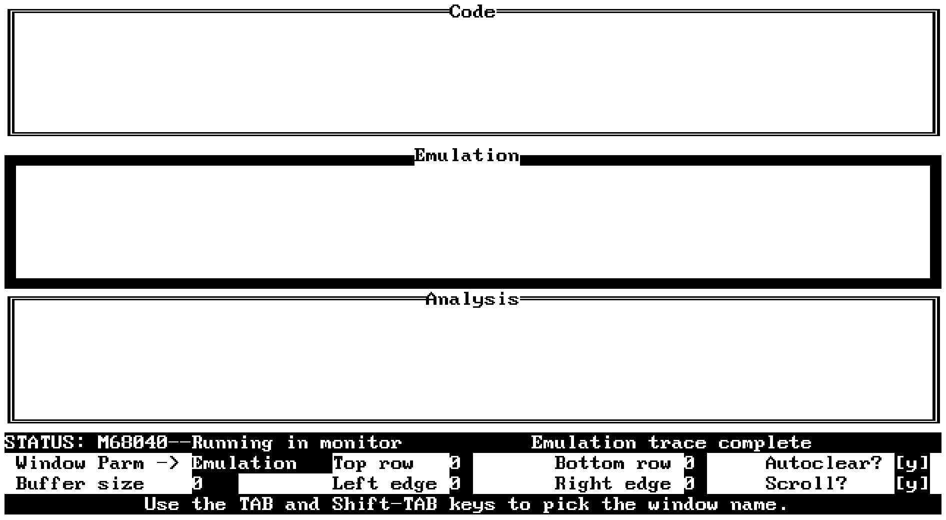
“Foreground Color:” The **Tab** or **Shift-Tab** key can be used to select the foreground color if **color** was chosen for the monitor type.

“Background Color:” The **Tab** or **Shift-Tab** key can be used to select the background color if **color** was chosen for the monitor type.

Window **U**tility **H**ide — Causes the specified window to be hidden from the PC Interface display. Enter the window name in the following field to hide it.

“Window Utility Hide:” Specify the name of the window to be hidden.

Window **U**tility **P**arameters — Allows you to customize the parameters of the specified window. You can change the top and bottom rows, the left and right edges, whether the contents of the window are cleared prior to each time it is written to, and whether information updated all at once. The form and fields are as follows:



```
                                     Code

                                   Emulation

                                   Analysis

STATUS: M68040--Running in monitor           Emulation trace complete
 Window Parm -> Emulation    Top row    0        Bottom row 0        Autoclear? [y]
 Buffer size    0            Left edge  0        Right edge 0        Scroll?    [y]
           Use the TAB and Shift-TAB keys to pick the window name.
```

Window Utility Parameters Form

"Window Parm: Specifies the name of the window whose parameters you wish to change. The **Tab** or **Shift-Tab** key can be used to select the window name from the list of active windows.

"Top row:" Specifies the row number (1 to 20) of the window's top row.

"Bottom row:" Specifies the row number (1 to 20) of the window's bottom row.

"Autoclear?:" Specifies whether the contents of the window are cleared prior to each time it is written. You can select **y** or **n**.

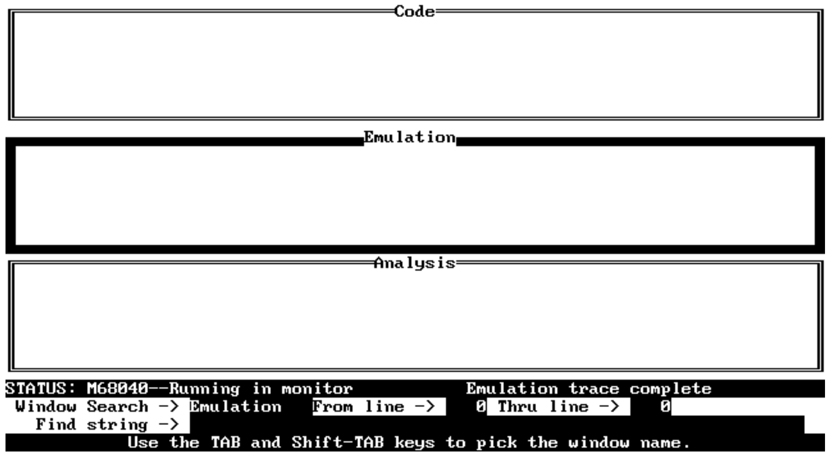
"Buffer size:" Specifies the number of rows (20 to 1030) of buffer space that can be used by the window.

"Left edge:" Specifies the column number (0 to 79) of the window's left edge.

"Right edge:" Specifies the column number (0 to 79) of the window's right edge.

"Scroll?:" Specifies whether the window should be displayed after it is modified. You can select **y** or **n**.

Window **U**tility **S**earch — Searches for a string between the specified line numbers in a window. The fields are:



```
STATUS: M68040--Running in monitor          Emulation trace complete
 Window Search -> Emulation    From line ->    0 Thru line ->    0
   Find string ->
            Use the TAB and Shift-TAB keys to pick the window name.
```

Window Utility Search Form

"Window Search:" The **Tab** or **Shift-Tab** key can be used to select the name of the window in which the string should be searched for.

"From Line:" This field holds the first line in the range of lines to be searched.

"Thru Line:" This field holds the last line in the range of lines to be searched.

"Find String:" This field holds the string to be searched for in the specified range of lines.

Window **U**tility **V**iew — Returns hidden windows to the PC Interface display.

Window Zoom

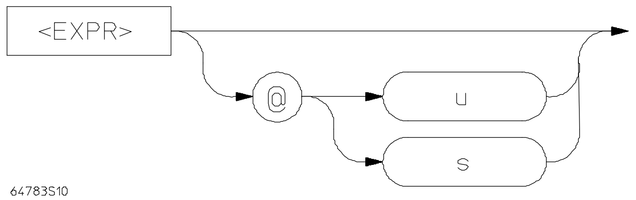
Window **Z**oom — Causes the specified window to occupy the full screen. **<Ctrl> z** can also be used to zoom the currently active window.

11

Expression Syntax

Describes the syntax of expressions, addresses and other typed-in items that are specific to the MC68040 emulator.

ADDRESS



<ADDR>

Address expressions allow you to enter an address in a form recognized by the MC68040 emulator. When you see the address variable <ADDR> in a command sequence, remember that it is unique to the MC68040 emulator.

The <EXPR> part here must be a 32-bit number, with an optional radix specifier. Numbers occupying less than 32 bits are left zero-filled to 32-bits. You can also specify either supervisor or user to qualify an address specification. The @ symbol is required only if you specify supervisor or user. Otherwise, @ must be omitted.

In some contexts you can specify a *range* of addresses, such as when specifying a memory map. An address range has the form

<EXPR>..<ADDRESS>

That is, the first address does not have a function code because it is assumed to be the same as the ending <ADDRESS>.

Examples

Suppose you create the following memory map:

```
0..0fff eram
```

Now, the following memory display commands are valid:

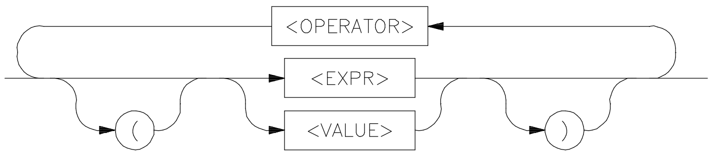
```
Memory Display Byte 0..0f <Enter>
Memory Display Byte 1000..100f@u <Enter>
Memory Display Byte 1000..100f@s <Enter>
```

You can specify the base with the address. For example:

```
100t    (100 base ten)
701o    (701 base eight)
234o@s (234 base eight in supervisor space)
```

Expressions

<EXPR>



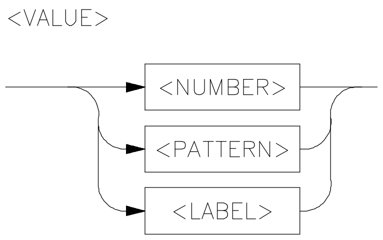
Numeric expressions are the root of all HP 64700 PC Interface expression types, including analyzer expressions and address specifications.

The expression capability in the PC Interface is very powerful. You may specify numbers in one of four different bases and use many different arithmetic and logical operators to form more complex expressions.

PC Interface expressions consist of other **expressions** and **values**, which may be modified by various **operators**. You may change the precedence of operators by enclosing expressions within parentheses.

Values

Values consist of **numbers** (in one of four bases), **patterns** (hexadecimal, octal, or binary numbers that also include don't care values), and **labels** (symbols which reference other numbers, from a symbol database).



Numbers are in hexadecimal, decimal, octal, or binary. You specify the base as follows:

Y y	Binary (example: 10010y)
Q q O o	Octal (example: 377o or 377q)
T t	Decimal (example: 197T)
H h	Hexadecimal (example: 0A7fH) (Note that hexadecimal numbers starting with any one of the letter digits A-F must be prefixed with a zero; otherwise the system will return an error message)

If you do not specify a base, numbers default to hexadecimal or decimal, depending on the context.

All numbers used in address specification, analyzer expressions, and any other specification relating to a microprocessor address, data or status value defaults to hexadecimal.

Numbers used to specify repeat count values, such as in the analyzer trace specification or step count, default to decimal.

Floating point numbers are supported by some emulators. The only legal operations on these values are addition, subtraction, multiplication, and division. You represent a floating point value as a decimal in the form:

```
[+|-]X.X[E[+|-]X]
```

Where X's represent decimal values and E indicates a following exponent.

Patterns are hexadecimal, octal, or binary numbers which include don't care digits, specified by the letters **X** or **x**. The character **?** represents a pattern of all don't care digits. For example:

1011xx11y

0A7Xh (equivalent to 000010100111xxxxy)

2x5Q (equivalent to 010xxx101y)

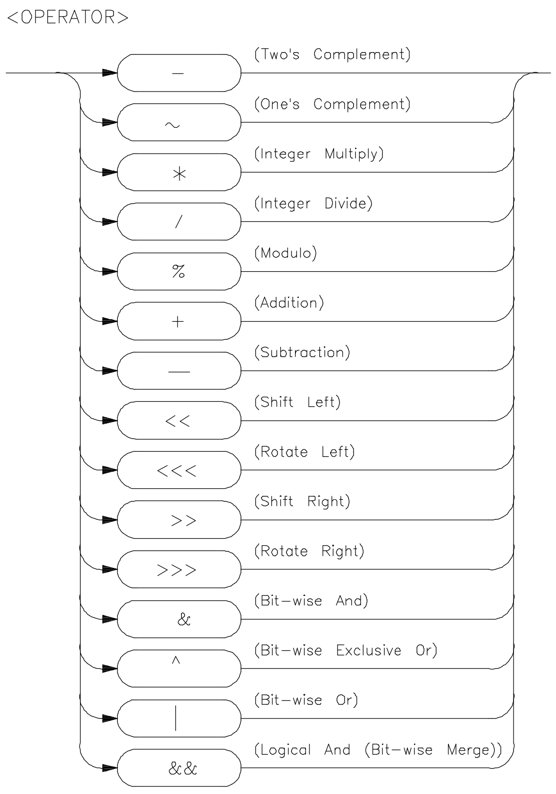
You will generally use patterns only in analyzer expressions.

Labels refer to names equated to numbers via the symbol database loaded for the current program or loaded using the **S**ystem **S**ymbols **G**lobal **L**oad command.

Operators

The expression capability includes a powerful set of operators, freeing you from the need to evaluate expressions before entering them into other expressions. All operations are carried out on 32-bit two's complement signed integers (values which are not 32-bit will be padded out with zeros when expression evaluation occurs). For emulators that directly support floating point data types, addition, subtraction, multiplication and division are done using 64-bit floating point format.

The operators are listed in the following diagram and described in order of evaluation precedence. As mentioned above, you may use parentheses in the expression to change the order of evaluation.



Note If your emulator supports symbols, and you are using a symbol in an expression, only the **+** and **-** operators are valid before and after the symbol. For example: 100h+main-5

- ~	Unary two's complement, unary one's complement. Two's complement is not allowed on patterns containing don't care bits. This is the truth table for one's complement: **0 => 1** **1 => 0** **X => X** Examples: **~1x0y = 0x1Y** **-1101Y = 0011Y**
* / %	Integer multiply, integer divide, integer modulo. These operations are not allowed on patterns containing don't care bits. Examples: **30afH*21 = 06468fH** **23T%4T=3** **0fa6/2 = 07d3h**
+ **-**	Addition, subtraction. Not allowed on patterns containing don't care bits. Examples: **03dh+03fh = 07ch** **1110Y-101Y = 1001Y**

<< <<< >> >>>	Shift left, rotate left, shift right, rotate right (you must specify the number of locations to shift or rotate after the operator).

Examples:

1x0Y<<1 = 1x00Y
1x0Y>>1 = 01xY
1x01Y>>>1 = 10000000000000000000000000001x0Y
0xxf0abcdH>>>4 = 0dxxf0abcH

& This symbol (&) represents a bit-wise AND operation. The truth table is as follows:

&	**0**	**1**	**X**
0	0	0	0
1	0	1	X
X	0	X	X

Example:

10xxy&11x1Y = 10xxY

^ This symbol (^) represents a bit-wise exclusive OR operation. The truth table is as follows:

^	**0**	**1**	**X**
0	0	1	0
1	1	0	X
X	0	X	X

Example:

10xxY^11x1Y = 01xxY

|

This symbol (|) represents a bit-wise inclusive OR operation. The truth table is as follows:

|	**0**	**1**	**X**
0	0	1	0
1	1	1	1
X	0	1	X

Example:

10xxY|11x1Y = 11x1Y

&&

This symbol (&&) represents a bit-wise merge operation. The truth table is as follows:

&&	**0**	**1**	**X**
0	0	*	0
1	*	1	1
X	0	1	X

An overlap, indicated by a ***** in the merge truth table, may occur if two patterns specify different values for a pattern bit. If an overlap occurs, the first pattern's value for that bit overrides the second pattern's value.

Example:

10xxY&&11x1Y = 10x1Y

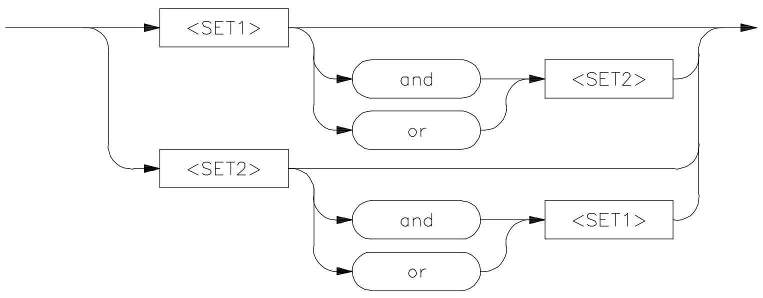
Using Expressions in Addressing and Analyzer Expressions

You can use the expression evaluation capability to form more powerful expressions for use in specifying addressing and analyzer expressions. For example, suppose you want to trigger the analyzer on the access to trap vector 13. Instead of calculating the address, since you know the base address is 080 hex and each vector is 4 address bytes, you can specify this as:

```
a=(080h+(13T*4))
```

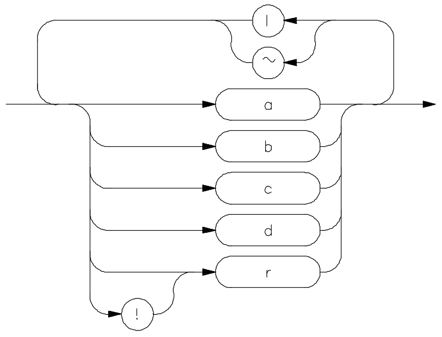
Analyzer Pattern Expressions

<COMPLEX_EXPR>



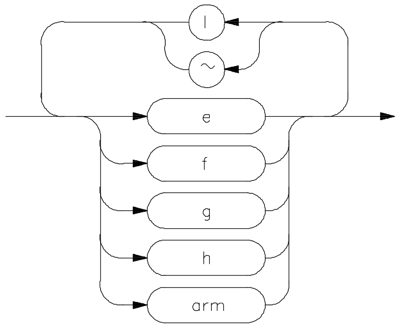
<SET1>

(restricted to one operator type in the set)



<SET2>

(restricted to one operator type in the set)



Complex Expressions Description

In the PC Interface analyzer trace specification, you use pattern labels, which have been assigned to various simple expressions, to form complex expressions.

Pattern Labels and Ranges

You assign pattern labels to simple expressions using the analyzer trace specification form. For example:

```
Pattern a: addr=2000
Pattern b: data!=00
Pattern c: stat=dma
Pattern d: addr=2000 and data=23
Pattern e: addr!=2105 and data!=0fc
```

You can also assign the range value:

```
Range: data=42..44
```

Sets

The pattern labels, along with the range and arm specifications, are divided into two sets.

Set 1:

a,b,c,d,r,!r

Set 2:

e,f,g,h,arm

Intraset Operations

You use intraset operators to form relational expressions between members of the same set. The operators are:

~ (intraset logical NOR)

| (intraset logical OR)

The operators must remain the same throughout a given intraset expression. So, you could form intraset expressions such as these:

```
a~b~r
```

(Pattern a NOR pattern b NOR range.)

```
b | !r
```

(Pattern b OR (NOT range).)

```
e | arm
```

(Pattern e OR arm.)

```
f ~ h
```

(Pattern f NOR pattern h.)

You **cannot** use the intraset operators to form expressions between set 1 and set 2. Also, remember that the intraset operator must remain the same throughout the set. Therefore, the following examples are **invalid**:

```
b~c|d
```

(This is incorrect because the operator must remain the same throughout the set.)

```
b~e
```

(You cannot use intraset operators for interset operations.)

Interset Operations

You use interset operators to form relational expressions between members of set 1 and set 2. The operators are:

and (interset logical AND)

or (interset logical OR)

You can then form the following types of expressions:

(set 1 expression) and (set 2 expression)

(set 1 expression) or (set 2 expression)

The order of sets does not matter:

(set 2 expression) and (set 1 expression)

Combination

You can use both the intraset and interset operators to form very powerful expressions.

```
a~b and e|arm
c or f~g~h
```

However, you cannot repeat different sets to extend the expression. The following is **invalid:**

```
a~b and e and c and g
```

DeMorgan’s Theorem and Complex Expressions

It seems that you only have a few operators to form logical expressions. However, using the combination of the simple and complex expression operators, along with a knowledge of DeMorgan’s Theorem, you can form virtually any expression you might need in setting up an analyzer specification.

DeMorgan’s theorem in brief says that

A NOR B = (NOT A) AND (NOT B)

and

A NAND B = (NOT A) OR (NOT B)

The NOR function is provided as an intraset operator. However, the NAND function is not provided directly. Suppose you wanted to set up an analyzer trace of the condition

(addr=2000) NAND (data=23)

This can be done easily using the simple and complex expression capabilities. First, you would define the simple expressions as the inverse of the values you wanted to NAND:

```
Pattern a: addr!=2000
Pattern b: data!=23
```

Then you would OR these together using the intraset operators:

```
a|b
```

This is effectively the same as:

(NOT addr=2000) OR (NOT data=23) =
(addr=2000) NAND (data=23)

If you need an intraset AND operator, you can use the same theory. Suppose you actually wanted:

(addr=2000) AND (data=23)

First, define the simple expressions as the inverse values:

```
Pattern a: addr!=2000
Pattern b: data!=23
```

Then you would NOR these together using the intraset operators:

```
a~b
```

This is effectively the same as:

(NOT addr=2000) NOR (NOT data=23) =
(addr=2000) AND (data=23)

STATUS

Some status equates are predefined by the emulator. These equates may be used in analysis expressions to qualify triggering or storage on specific bus cycle types.

68040 Equates

Name	**Value**	**Description**
ack	11xxxxxxxx1xxxxxy	Acknowledge access.
alt0	10xxxxxxxx1x001xy	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 table 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.

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.

Error Messages

This chapter contains descriptions of error messages that can occur while using the PC Interface of the MC68040 emulator. When errors occur, you can display the error messages and their associated numbers (if any) in the Error_Log window. Obtain that window with the command:

```
Window Utility View
```

In the form that appears, type in "Error_Log".

The error messages in this chapter are listed in alphabetical order, and each description includes the cause of the error and the action you should take to remedy the situation.

The emulator can return messages to the display only when it is prompted to do so. Situations may occur where an error is generated as the result of some command, but the error message is not displayed until the next command (or a carriage return) is entered.

Up to eight error messages can be displayed at one time. If more than eight error messages are generated, only the last eight are displayed.

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 $\overline{\text{TEA}}$ (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 $\overline{\text{TEA}}$.

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.

Another window exists by this name

Cause: You tried to open a new user window with a name that is already in use.

Action: Choose another name and use **W**indow **O**pen to open the window.

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 **P**rocessor **S**tep 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.

Cannot interpret emulator output

Cause: The PC Interface cannot interpret the emulator’s output. You may have started the PC Interface and then turned off the emulator.

Action: Restart the emulator and the PC Interface. A memory resident program (TSR) or device driver in your PC may be preventing reliable serial communication.

Cannot open command download file <filename>. Exit immediately

Cause: The command download file you specified cannot be opened.

Action: Exit and ensure that the HPBIN environment variable is set to HP64700\BIN or the BIN directory where the PC Interface was installed.

Cannot open command file <filename>

Cause: The command file you specified cannot be opened.

Action: Try again, making sure you enter the command file name correctly.

Cannot open configuration file <filename>

Cause: The configuration file you specified cannot be opened.

Action: Exit and try again, making sure you enter the configuration file name correctly.

Cannot set monitor to type specified

Cause: Your monitor does not support the specified attributes (monochrome or color).

Action: Select another type.

Clock speed not available with current count qualifier (Error 1239)

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

Action: Change the count qualifier; then reenter the command. See the chapter titled, "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. Refer to the chapter titled "Making Coordinated Measurements" for further information.

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. Refer to the chapter titled "Making Coordinated Measurements" for further information.

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 configuration 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 **M**emory **C**opy or **M**emory **M**odify 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 **M**emory **C**opy command. Then remap the target memory range to emulation memory, and load the absolute file into emulation memory using the **M**emory **L**oad command. Refer to the chapter titled, "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 Trigger on or Find specification in the Internal State Trace Specification form.

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** or **very fast**. This error also occurs when you attempt to specify a “state” count qualifier when the maximum qualified clock speed is **fast**.

Action: Change the clock speed; then change the count qualifier. See the chapter titled, "Using the Analyzer" for more information.

Corrupt symbol file found

Cause: An attempt was made to access a symbol database file which was not of the correct format.

Action: Try again making sure to use correct file names.

Coverage not supported (Error 175)

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

Critical error occurred while attempting to access the specified file

Cause: You may have left the disk out of the floppy drive or failed to close the drive door.

Action: Insert the disk and make sure the drive door is fully closed. Then try to access the specified file again.

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 mmu/cache while background monitor is selected (Status 157)

Cause: This status message indicates that the MMU and/or cache was 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.

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 **C**onfig **M**ap **M**odify command and use the **dp** attribute for only one address range, or select a background monitor and recreate your map.

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 and then you assigned the **dp** attribute.

Action: You can:

- Reenter the address specification in your map and specify the **dp** attribute. Be sure to restrict the address range to 4 Kbytes.
- Reenter your specification, and use regular emulation memory. That is, do not include the **dp** attribute.

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 "Terminate monitor 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: "Memory Display ..."). 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.

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 and Service office for assistance.

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.

Full disk

Cause: Your disk is full.

Action: Either make room on this disk by deleting unnecessary files or insert another disk and try again.

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 breakpoint:" 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: Answer "no" to the "Force hardware breakpoint:" option 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 the Installation and Service Chapter 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 specify that counts be displayed absolute, or you can 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 value specified

Cause: The value you entered may be outside the legal range for the item or the item may have a specific boundary requirement (for example, 4 Kbyte boundary).

Action: Check the reference part of this manual for more information and try again.

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.

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 **R**egister **D**isplay and **R**egister **M**odify 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 and/or the master stack.

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 **R**egister **D**isplay and **R**egister **M**odify 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 and/or the master stack.

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 **R**egister **D**isplay and **R**egister **M**odify 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 **R**egister **D**isplay and **R**egister **M**odify 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 and/or the master stack.

Invalid address: <address> (Error 310)

Cause: 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. 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. 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 attribute for memory type : <attribute> (Error 140)

Cause: The dual-port memory and "Terminate monitor 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 the chapter titled, "Configuring the Emulator" for information on the memory type attributes.

Invalid attribute. The dual ported (dp) attribute is only allowed on one term

Cause: You tried to set the dual ported (dp) attribute on at least two terms.

Action: Set the attribute to something other than dual ported on all but one term in the **C**onfig **M**ap **M**odify form.

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: Refer to the "Expressions Syntax" chapter for 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 configuration file: <filename>

Cause: You tried to load a file that was not a valid configuration file. Configuration files must have matching pairs of section headers and cannot contain blank lines.

Action: Try your command again and check the configuration file name.

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: Reenter the command, specifying only configuration items that are supported by your emulator. Refer to the "Configuring the Emulator" chapter 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 data in <section> section of configuration file <name>

Check the <section> part of the configuration file called <name> for errors. Compare your configuration file with one produced with the **C**onfig **S**tore command. Fix any errors and try again.

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 available in the Analysis Display form.

Invalid emulation configuration received from emulator

Cause: A memory resident program (TSR) or device driver in your PC may be preventing reliable serial communication.

Action: Remove it and try again.

Invalid emulation memory map configuration received from emulator

Cause: A memory resident program (TSR) or device driver in your PC may be preventing reliable serial communication.

Action: Remove it and try again.

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 Processor DeMMU File Store 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: Reenter the expression, following the syntax rules for that type of expression. Refer to the chapter titled, "Expression Syntax" 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 EXPRESSIONS 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 **tci** (Transfer Cache Inhibit), **lock** (Terminate monitor bus cycles), and **dp** (Dual Port Memory).

Action: Respecify your map attributes, 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: **eram, erom, tram, trom, grd.**

Action: Respecify your memory type using 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. Refer to the chapter titled, "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 your specification 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: Reenter the command with the correct syntax. Refer to the chapter titled, "Emulator Commands" for more information.

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 the chapter titled, "Using the Analyzer" for more information.

Invalid state number specified

Cause: You specified a state number that was not in the valid range of states.

Action: Enter a number between -1024 and 1023.

Invalid syntax: expected range

Cause: You entered something that was not a range of values.

Action: Try entering something in the form first..last.

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 trigger option specified

Cause: Either the value entered on the **C**onfig **T**rigger form was not valid, or an attempt was made to drive an analyzer with both trig1 and trig2.

Action: Use the <Tab> key to view the valid choices, or drive the analyzer with either trig1 or trig2

Invalid window buffer size specified

Cause: You tried to modify the window buffer size to an invalid value.

Action: Use **W**indow **U**tility **P**arameters to change the buffer size (the number of rows that can be used by this window) to a value in the range of 20 to 1030.

Invalid window color specified

Cause: You either tried to set the foreground or background color to an invalid color or you tried to set the color in a monochrome monitor.

Action: If your monitor is monochrome you cannot set any colors. Otherwise, use the **Tab** key to select the foreground and background colors in **W**indow **U**tility **C**olor.

Invalid window corners specified

Cause: You tried to set invalid window corners.

Action: Use **W**indow **U**tility **P**arameters to specify window rows and edges. Window rows must be values in the range of 1 to 20. Window edges must be values in the range of 0 to 79.

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 **addr**, **data**, and **stat**, 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 your label specification, 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, and then reenter the analyzer command using the newly defined label.

Logging already enabled to file:

Cause: You tried to use **S**ystem **L**og **I**nput **E**nable, **S**ystem **L**og **O**utput **E**nable, or **S**ystem **L**og **B**oth **E**nable to enable logging to a file that had already been enabled. No action is required.

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. See the ADDRESS syntax pages in the chapter titled, "Expression Syntax" in this manual.

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 **R**egister **D**isplay and **R**egister **M**odify 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 and/or the master stack.

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 **R**egister **D**isplay and **R**egister **M**odify 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 and/or the master stack.

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 **R**egister **D**isplay and **R**egister **M**odify 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 and/or the master stack.

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 **R**egister **D**isplay and **R**egister **M**odify 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 corrupted

Cause: System memory has been corrupted.

Action: Quit as soon as possible.

Memory modify aborted; next address: <address> (Error 754)

Cause: One of these messages is displayed if a break occurs during processing of the **M**emory **C**opy or **M**emory **M**odify 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 **M**emory **C**opy or **M**emory **M**odify 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. Refer to the chapter titled, "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 in the "Use TC reg:" field of the associated form when invoking the MMU or deMMUer commands.

Module name specified is not a valid local symbol module.

Cause: You may have entered the module name incorrectly.

Action: Try again, making sure to enter the correct module name. If you are not sure of the module name, you can display a listing of local symbol modules using the **S**ystem **S**ymbols **G**lobal **D**isplay command.

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 the chapter that discusses the emulation configuration options.

Nesting of command files exceeded 8 levels ... command file execution halted

Cause: You cannot nest more than eight levels in command files.

Action: Try condensing the command files into fewer levels.

No local symbol modules found

Cause: An attempt was made to display local symbols from a symbol database that contained none. No action is required.

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 modifications allowed while trace is in progress

Cause: You tried to modify the trace while it was in progress.

Action: You must stop the trace using **A**nalysis **H**alt. Make any necessary modifications and then restart the trace using **A**nalysis **B**egin.

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 software breakpoints are currently defined

Cause: The **B**reakpoints command you entered could not be executed because there are no software breakpoints currently defined in the breakpoint list.

Action: You must first add the breakpoint using **B**reakpoint **A**dd, and then repeat the command you entered.

No symbol database, cannot look up: <symbol>

Cause: This message occurs when there is no symbol database in memory, where <symbol> is the symbol you entered that cannot be found.

Action: You must load symbols before they can be referenced. You can use the **S**ystem **S**ymbols commands to load symbols, or the **M**emory **L**oad command to load a user program.

No symbol file is currently active

Cause: This message occurs when there is no symbol database in memory.

Action: You must load symbols before they can be referenced. You can use the **S**ystem **S**ymbols commands to load symbols, or the **M**emory **L**oad command to load a user program.

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 breakpoint:" 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.

PC Interface is out of memory

Cause: The PC Interface has tried to allocate memory on your PC and failed.

Action: Try erasing the contents of windows (<CTRL>e and <CTRL>a). It may be necessary to exit the PC Interface and re-enter to reinitialize the heap.

PC Interface is out of memory. Please exit immediately

Cause: The menu system of the PC Interface has tried to allocate memory on your PC and failed.

Action: Exit immediately. Otherwise, subsequent commands could cause your PC to hang, requiring a reboot.

Previous memory display command was too big

Cause: In the previous memory display command you specified a memory expression that was too complex.

Action: Try **M**emory **D**isplay again, specifying a simpler expression.

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. Refer to the chapter titled "Using the Analyzer" for details of how to use the sequence.

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 "Register Commands" pages in the chapter titled, "Emulator Commands" in this manual for a list of register names.

Repetitive memory display in progress. -- ESC to abort

Cause: This status message lets you know that a repetitive memory display is in progress.

Action: You can press the <ESC> key to stop the display.

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 "Unmapped memory: Type" term. For example, if "Unmapped memory: Type" is mapped to target RAM and the **tci** (transfer cache inhibit) attribute is OFF, one hardware resource is required to add a map term for target memory that requires **tci** 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 "Unmapped memory: Type" term to target memory both with the **tci** attribute on and off. Find out which specification for "Unmapped 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 **P**rocessor **B**reak 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)

Cause: You tried to set a software breakpoint that was not previously defined. <address> prints the address of the breakpoint you attempted to set.

Action: Add 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 **P**rocessor **S**tep 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 **P**rocessor **S**tep command with a stepcount greater than zero. The break could have been due to any of the break conditions or a <CTRL> c break. The number of steps completed is displayed.

Stepping failed (Error 680)

Cause: Stepping has failed for some reason. For example, this message will appear if the emulator can't modify the trace vector, which is used to implement the step function. Usually, this error message will occur with other error messages.

Action: Refer to the descriptions of the accompanying error messages to find out more about why stepping failed.

Symbol cannot contain text after the wildcard (Error 879)

Cause: You tried to include text after the wildcard specified in the symbol name (for example, **symbol*text**).

Action: Enter the symbol again, but do not include text after the wildcard (*).

Symbol cannot contain wildcard in this context (Error 878)

Cause: You tried to enter a global, local, or user symbol name using the wildcard (*) incorrectly.

Action: When you enter the symbol name again, include the wildcard (*) at the end of the symbol.

Symbol not found: <symbol> (Error 877)

Cause: This occurs when you try to enter a symbol name that doesn’t exist.

Action: Enter a valid symbol name.

Target 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 $\overline{\text{TEA}}$.

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 "Terminate monitor 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 $\overline{\text{TEA}}$ 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 $\overline{\text{TA}}$ or $\overline{\text{TEA}}$. 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 configuration question "Terminate monitor 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.

The maximum number of windows are already defined

Cause: You tried to open a new window when the maximum number of user windows has already been defined.

Action: Either use an existing window or delete any unnecessary windows using **W**indows **D**elete and then open the new window using **W**indow **O**pen.

The name of an empty window was specified

Cause: The action you specified could not be carried out because the window you specified was empty. For example, if you entered **W**indow **S**tore and the name of an empty window, you might get this error. No action is necessary.

There is currently no default local module

Cause: You referenced a line number symbol (for example, "line 23") without first specifying a local symbol module.

Action: Try again, making sure to specify a local symbol module first (for example, "main: line 23"). Once a local symbol module has been referenced, it becomes the default until another is specified.

Trace status equate not found: <status_equate>

Cause: The trace status equate you specified in the **stat** field of the pattern screen could not be found. For example, you may have entered "byte" when "siz_byte" was expected.

Action: Use the <Tab> key to view the valid choices.

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 **C**onfig **T**rigger command and form to have the analyzer send its trigger-recognition signal to the emulator to cause an emulation break. 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)

Cause: 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 the chapter titled "Using the Emulator" in this manual.

Unable to access the specified file

Cause: The file you specified could not be found.

Action: Try again, making sure that you enter the correct file name.

Unable to allocate sufficient memory

Cause: The PC Interface could not start due to insufficient memory.

Action: Remove memory resident programs (TSRs) or device drivers and try again. Your PC must have a minimum of 640K.

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 **P**rocessor **B**reak command to break to the monitor. If reset by the target system, release that reset. If halted, try **P**rocessor **R**eset and **P**rocessor **B**reak 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 and display the emulation trace patterns in the complex configuration, or display the trace format to see where the label is used. You must change the pattern or format specification to remove the label before you can delete it.

Unable to execute the MS-DOS command

Cause: You specified an MS-DOS command that cannot be executed because it requires too much memory.

Action: Use the MS-DOS CHKDSK command to see how much memory is available. (You may not be able to run MS-DOS CHKDSK if there is not enough memory available.)

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 load specified symbol file

Cause: The symbol file you specified could not be loaded.

Action: Try again, making sure to enter the correct file name.

Unable to load the specified configuration file

Cause: The configuration file you specified could not be loaded.

Action: Try again, making sure to enter the correct file name.

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 open the requested file for logging purposes

Cause: Either the file you requested cannot be created or it cannot be written to.

Action: Check to make sure that you entered the correct file name.

Unable to open the specified window ... ESC to abort

Cause: There is insufficient memory left on your PC to open the specified window.

Action: Try a smaller buffer size or remove memory resident programs (TSRs) or device drivers from your PC.

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 and 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.

Unable to store to the specified configuration file

Cause: The file name you specified cannot be stored.

Action: Make sure you entered the file name correctly or that the file is not write-protected.

Unable to write to disk ... Disk may be full

Action: Either insert a new disk or delete unnecessary files from the current disk to free memory and try again.

Unexpected EOF in section <section> of configuration file <name>

Cause: The configuration file lacks a closing section header and is probably corrupt.

Action: Compare it to a configuration file stored with the **C**onfig **S**tore command. Correct any errors and try again.

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 **B**reakpoints **A**dd command.

Action: Remove the breakpoints inserted in your code before assembly and link, and then reinsert them using the **B**reakpoints **A**dd command. If this message was received after you enabled the MMU, read "Execution Breakpoint Problems" in the chapter titled, "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 the chapter titled, "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 the chapter titled, "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.

Window load in progress

Cause: This status message means that the contents of the file you specified are currently being loaded in the window you specified. No action is required.

Window specified is not a hidden window

Cause: You tried to use **W**indow **U**tility **V**iew to return a hidden window to the PC Interface display, but the window you specified was not hidden.

Action: If you entered the wrong window name, then try again, making sure to enter the correct window name. Otherwise, no action is necessary.

Window specified is not a user-defined window

Cause: The **W**indow command that you specified could not be executed because you entered a window name that was not user-defined.

Action: You can try the command again with a user-defined window name.

Window specified is not a visible window

Cause: The **W**indow command that you specified could not be executed because you entered the name of a hidden window.

Action: You can use **W**indow **U**tility **V**iew to return the hidden window to the display. Then try the command again.

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

Data File Formats

Software development tools (in other words, compilers, assemblers, linkers, etc.) generate absolute files that contain program code. Once these absolute files are loaded into emulation and target system memory, the emulator can execute the program code.

The Absolute File

The file format reader creates an absolute file (<file>.hpa). This absolute file is a binary memory image, which is optimized for efficient downloading into the emulator.

The ASCII Symbol File

The ASCII symbol file (<file>.hps) produced by the reader contains global symbols, module names, local symbols, and, when using applicable development tools such as a "C" compiler, program line numbers. Local symbols evaluate to fixed (static, not stack relative) address.

You must use the required options for your specific language tools to include symbolic ("debug") information in the absolute and symbol files.

The symbol file contains symbol and address information in the form:

```
 module_name1
 module_name2
 ...
 module_nameN
global_symbol1  0100234
global_symbol2  0100678
...
global_symbolN  0100BCD
|module_name1|# 1234         0200872
|module_name1|local_symbol1  0200653
|module_name1|local_symbol2  0200872
...
|module_name1|local_symbolN  0200986
```

Symbols are sorted alphabetically in the groups: module names, global symbols, and local symbols.

Line numbers will appear similar to a local symbol except that "local_symbolX" will be replaced by "#NNNNN" where NNNNN is a five digit decimal line number. The addresses associated with global and local symbols are specific to the processor for which the absolute files were generated.

You access line number symbols by entering the following on one line in the order shown:

module name
colon (:)
space
the word “line”
space
the decimal line number

For example:

```
MAIN.C: line 23
```

If your emulator can store symbols internally, symbols will appear in disassembly. When the line number symbol is displayed in the emulator, it appears in brackets. Therefore, the symbol “MODNAME: line 345” will be displayed as “MODNAME:[345]” in mnemonic memory and trace list displays.

The space preceding module names is required. Although formatted for readability here, a single tab separates symbol and address.

The local symbols are scoped. This means that to access a variable named “count” in a function named “Foo” in a source file module named “MAIN.C,” you would enter “MAIN.C:COUNT” as shown below.

How to Access Variables

Module Name	Function Name	Variable Name	You Enter:
MAIN.C	Foo	count	MAIN.C:Foo.count
MAIN.C	bar	count	MAIN.C:bar.count
MAIN.C		line number 23	MAIN.C: line 23

Command File

A command file is an ASCII file containing PC Interface commands. You can create command files from within the interface by logging commands to a command file as you execute the commands. Or, you can create command files outside the interface with an ASCII text editor. You can send a command file to the PC Interface and have it execute the commands found there as if you typed them directly into the interface command line.

With a single command file, you can implement a complete test procedure. For example, you could start the interface and execute your command file. The command file could load a configuration, load an absolute file, modify registers or memory, set up a trace specification, start the program, capture the trace, and save the trace listing to a file. (The ability to capture information from the emulator may be limited, and depends on the host computer configuration.)

You can put most PC Interface commands into a command file. The only things that you cannot do in a command file are:

- Define function key macros.
- Set up the analyzer trace specification.
- Change the emulator configuration.

These things must be done in a configuration file. See the section “Using Configuration Files.”

As with any source file, comments in command files help to explain the operation of the command file and can also contain creation and modification information for the command file. You can put comments in command files by using a text editor; a “#” character anywhere in a line means that the rest of the text on the line is a comment.

Command files may be nested up to 8 levels. (Nesting means that one command file calls another.)

Configuration File

The configuration file is an ASCII file having several different sections. These sections begin and end with a section name; between the names are the current configuration settings for that section.

$SYSWIN	Contains settings for the PC Interface system-defined windows.
$USERWIN	Contains settings for user-defined windows (if any).
$MISC	Contains color settings for display monitors.
$KEYMAC	Contains function key macro definitions.
$SYMDB	Defines the current symbol database.

$EMUL	Contains the current emulator configuration as a series of Terminal Interface configuration commands.
$STLABINT	Lists the analyzer trace format and label information.
$STPATINT	Lists pattern and range specifications for the emulation analyzer.
$STSEQINT	Lists sequencer term definitions for the emulation analyzer.
$STLABEXT	Lists the analyzer trace format and label information for the external state.
$STPATEXT	Lists pattern and range specifications for the emulation analyzer for the external state.
$STSEQEXT	Lists sequencer term definitions for the emulation analyzer for the external.
$TIMSPEC	Contains the trace specification for the external timing analyzer.
$TIMWAVE	Contains the datafrom the lase captured timing analyzer measurement.
$LABSPEC	Contains the analyzer trace format for the external timing analyzer.
$LABWAVE	Contains the signal display format for the external timing analyzer.

You can modify a configuration file using an ASCII text editor, then load it into the PC Interface. However, the simplest and safest way to change a configuration file is to load an existing configuration, modify it using the PC Interface commands, then save it to a new file.

Function Key Macro Configuration File

You can create a function key macro outside the PC Interface by using a text editor to create a configuration file. The key macro definitions must be preceded and followed by the $KEYMAC separator. Each macro definition begins with the key combination that represents the macro. A colon separates the macro assignment from the definition. The definition itself appears as the sequence of keystrokes in the macro.

- All keystroke sequences that are not part of the standard printable ASCII character set must be enclosed in brackets. This includes characters in the range 0..31 decimal and 127..255 decimal. For example, function key 3 is represented as **<F3>**.
- The control key (<**Ctrl**>) is represented using the circumflex symbol (^) when the character that follows it is part of the ASCII character set, and is represented by the string “Ctrl” when the following character is not part of the ASCII character set. For example, **<Ctrl>M** (<**Enter**>) is shown as **<^M>**; **<Ctrl>F5** is shown as **<CtrlF5>**.The following shows the default function key macros defined by the PC Interface, as represented in a configuration file:

```
$KEYMAC
<F1>:psp1<M^>
<F2>:\>>
<F10>:sel
$KEYMAC
```

Nesting is limited to 16 levels. Direct or indirect recursion of macros is not permitted, except as a chain.

14

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 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 AVEC, BCLK, BG, CDIS, MDIS, IPLx, PCLK, RSTI, SCx, TBI, TLNx, TCI, TCK, TEA	I_{IL} I_{IH}	–250 —	— 25	µA
Hi-Z (Off-State) Leakage Current @ 0.5/2.4 V An, CIOUT, Dn, LOCK, LOCKE, SIZx, TDO, TMx, TLNx, TTx, UPAx BB, R/W, TIP, TS TA	I_{TSI}	–50 –100 –200	50 100 200	µA
Output High Voltage I_{OH} = –32 mA: An, Dn, SIZx, TTx, UPAx, LOCK, LOCKE, TLNx, CIOUT, TMx, PSTx, RSTO, BR, MI, BG,reset flying lead I_{OH} = –3.2 mA: R/W, TS, TIP, BB, TA, IPEND	V_{OH}	2.0 2.4	— —	V
Output Low Voltage I_{OL} = 64 mA An, Dn, SIZx, TTx, UPAx, LOCK, LOCKE, TLNx, CIOUT, TMx, PSTx, RSTO, BR, MI, BG, reset flying lead I_{OL} = 24 mA R/W, TS, TIP, BB, TA, 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	I_{CC}			
f = 25 MHz		—	1.4	A
f = 33 MHz		—	1.8	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 $\overline{\text{BR}}$, $\overline{\text{BB}}$ Valid	9	25	6.5	22.5	5.25	21	ns
43	BCLK to $\overline{\text{MI}}$ Valid	9	25	6.5	22.5	5.25	21	ns
48	BCLK to $\overline{\text{TA}}$ Valid	9	25	6.5	22.5	5.25	21	ns
50	BCLK to $\overline{\text{IPEND}}$, PSTx, $\overline{\text{RSTO}}$ 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	$\overline{\text{TA}}$ Valid to BCLK (Setup)	15	—	15	—	13	—	ns
22b	$\overline{\text{TEA}}$ Valid to BCLK (Setup)	15	—	15	—	14	—	ns
22c	$\overline{\text{TCI}}$ Valid to BCLK (Setup)	15	—	15	—	14	—	ns
22d	$\overline{\text{TBI}}$ Valid to BCLK (Setup)	15	—	15	—	14	—	ns
23	BCLK to $\overline{\text{TA}}$, $\overline{\text{TEA}}$, $\overline{\text{TCI}}$, $\overline{\text{TBI}}$ Invalid (Hold)	2	—	2	—	2	—	ns
24	$\overline{\text{AVEC}}$ Valid to BCLK (Setup)	10	—	10	—	10	—	ns
25	BCLK to $\overline{\text{AVEC}}$ 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

HP 64783A/B Input AC Timing Specifications

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	$\overline{\text{BB}}$ Valid to BCLK (Setup)	12	—	12	—	12	—	ns
41b	$\overline{\text{BG}}$ Valid to BCLK (Setup)	12	—	12	—	12	—	ns
41c	$\overline{\text{CDIS}}$, $\overline{\text{MDIS}}$ Valid to BCLK (Setup)	13	—	13	—	13	—	ns
41d	$\overline{\text{IPL}}$x Valid to BCLK (Setup)	8	—	8	—	8	—	ns
42	BCLK to $\overline{\text{BB}}$, $\overline{\text{BG}}$, $\overline{\text{CDIS}}$, $\overline{\text{IPLx}}$, $\overline{\text{MDIS}}$ 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/$\overline{\text{W}}$ 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/$\overline{\text{W}}$, SCx Invalid (Hold)	2	—	2	—	2	—	ns
46	$\overline{\text{TS}}$ Valid to BCLK (Setup)	14	—	14	—	12	—	ns
47	BCLK to $\overline{\text{TS}}$ Invalid (Hold)	2	—	2	—	2	—	ns
49	BCLK to $\overline{\text{BB}}$ 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

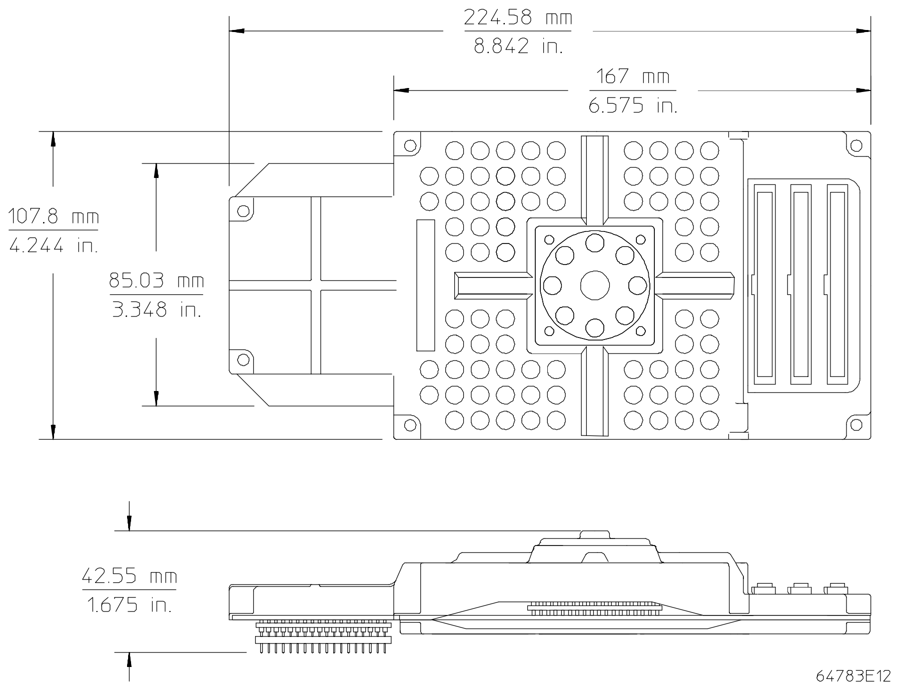
HP 64783A/B, 8.2 kg (18 lb). Any component used in suspending the emulator must be rated for 30 kg (65 lb) capacity.

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

Installation and Service Guide

Installation and Service Guide

In This Part

Part 4 of this book describes how to install the emulator in the card cage, how to install the demo board power cable, SRAM modules, rivets and covers, and the emulator probe cable. It also shows you how to connect the probe to the demo board, verify performance of the hardware, and use the progflash program to ensure software compatibility.

15

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 the chapter titled "Specifications and Characteristics" in this manual.

In the ideal case, you would use the emulator specifications listed in this manual when designing your target system instead, of 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.

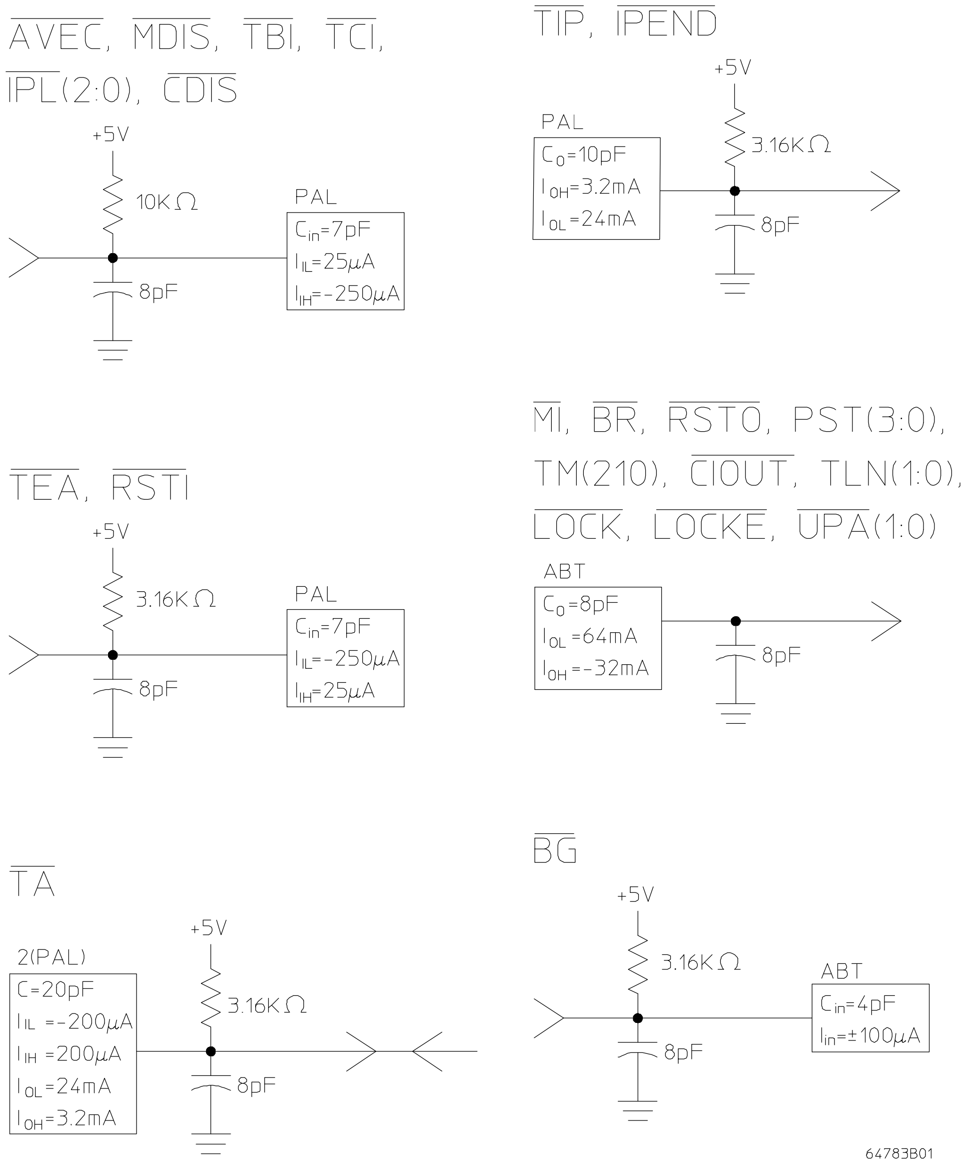
The BCLK clock is the most important signal to the emulator because all system timing is derived from this signal. The BCLK clock signal must have clean edges; the duty cycle of this clock is not particularly important. The emulator regenerates an internal BCLK from this signal with a 50% duty cycle. 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.

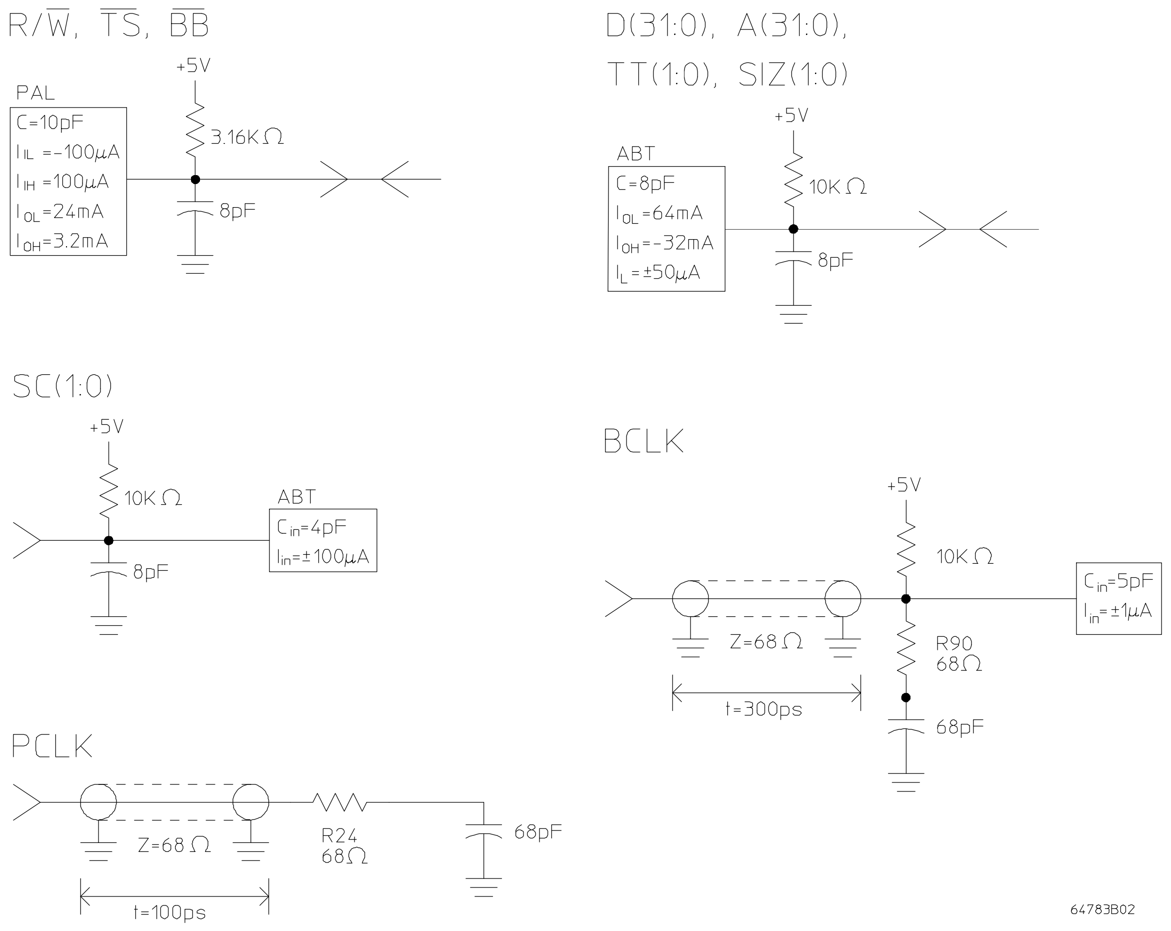
Both the BCLK and PCLK signals are terminated on the emulator. The terminations are placed on these signals, even though the emulator causes only a short electrical stub, so that accessories such as the flexible cable can be used to connect the emulator probe to your target system. The terminations on 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/W̄, T̄S̄, B̄B̄
+5V
PAL
C=10pF
I_{IL} =-100μA
I_{IH} =100μA
I_{OL}=24mA
I_{OH}=3.2mA
3.16KΩ
8pF
D(31:0), A(31:0),
TT(1:0), SIZ(1:0)
+5V
ABT
C=8pF
I_{OL}=64mA
I_{OH}=-32mA
I_L=±50μA
10KΩ
8pF
SC(1:0)
+5V
10KΩ
ABT
C_{in}=4pF
I_{in}=±100μA
8pF
BCLK
+5V
10KΩ
C_{in}=5pF
I_{in}=±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, obtain the terminal interface through its terminal window. Type the following command:

```
System Terminal
```

Command line prompts in this interface indicate the state of the emulator, and end with "$". For example:

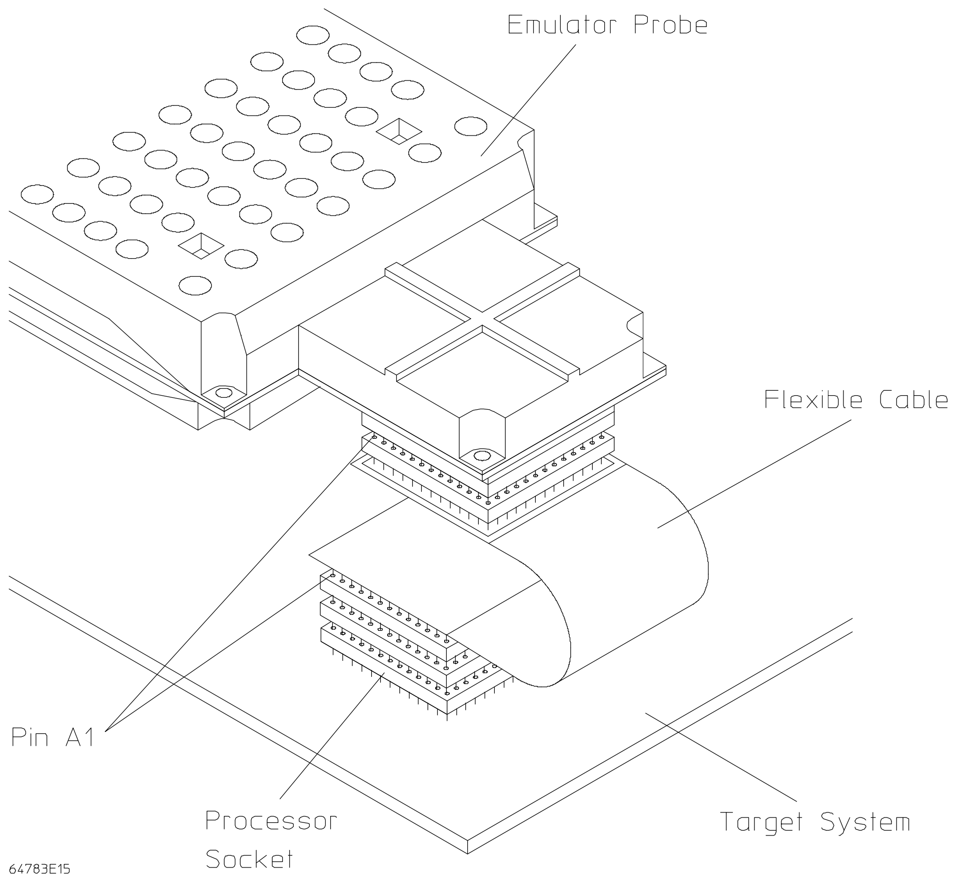
R$ indicates the emulator is reset.

M$ indicates the emulator is running the monitor program.

Connecting the emulator to the target system

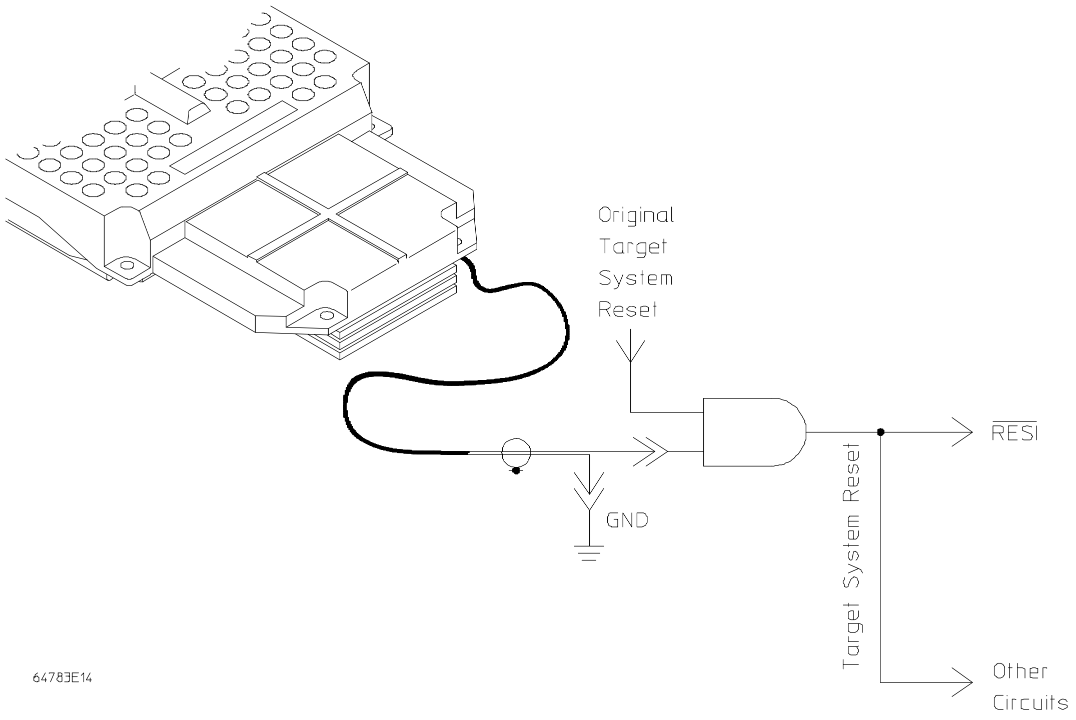
Plugging the emulator into a target system can be difficult because of mechanical constraints. If the mechanical constraints cannot be removed so that the emulator can be plugged directly into the target socket, there are several accessories available to help with the connection. These accessories are:

- 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 the Installation and Service Chapter in this manual.

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 <RETURN>.

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 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 <RETURN>, 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 the clock looks suspect, 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 $\overline{\text{TA}}$, $\overline{\text{TEA}}$, and $\overline{\text{TBI}}$, 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 $\overline{\text{TS}}$ signal during alternate bus master activity to see if it needs to intervene in the cycle (snooping). If the processor sees a $\overline{\text{TS}}$ signal but misses the corresponding $\overline{\text{TA}}$ signal, the processor may hang, waiting for this bus cycle to complete, even though the bus was granted to the MC68040 and released.

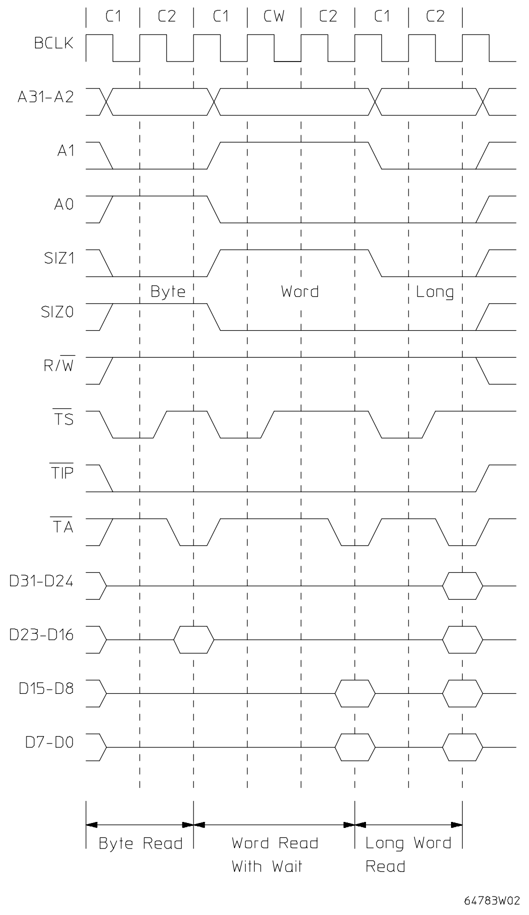
If bus cycles are occuring, then the "b$" prompt only indicates that bus cycles are infrequent. A type of system that would exhibit this behavior would be an interrupt-driven system. When done processing an interrupt, the system could execute a STOP instruction to wait for the next interrupt. If the interrupts were infrequent a "b$" 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, $\overline{\text{TS}}$. The $\overline{\text{TS}}$ 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 and/or a transfer error acknowledge TEA 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.

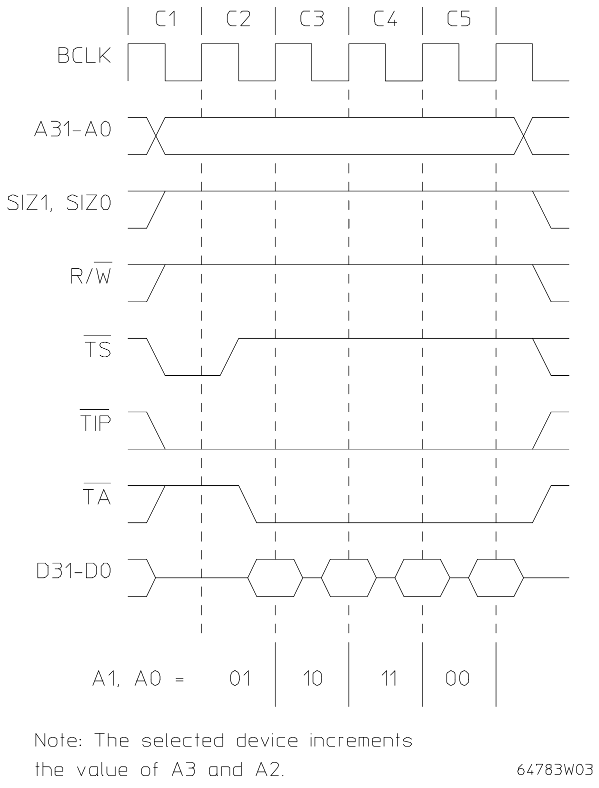
tg addr=00badad00
tp c
tgout trig2
bnct -r trig2
t

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.

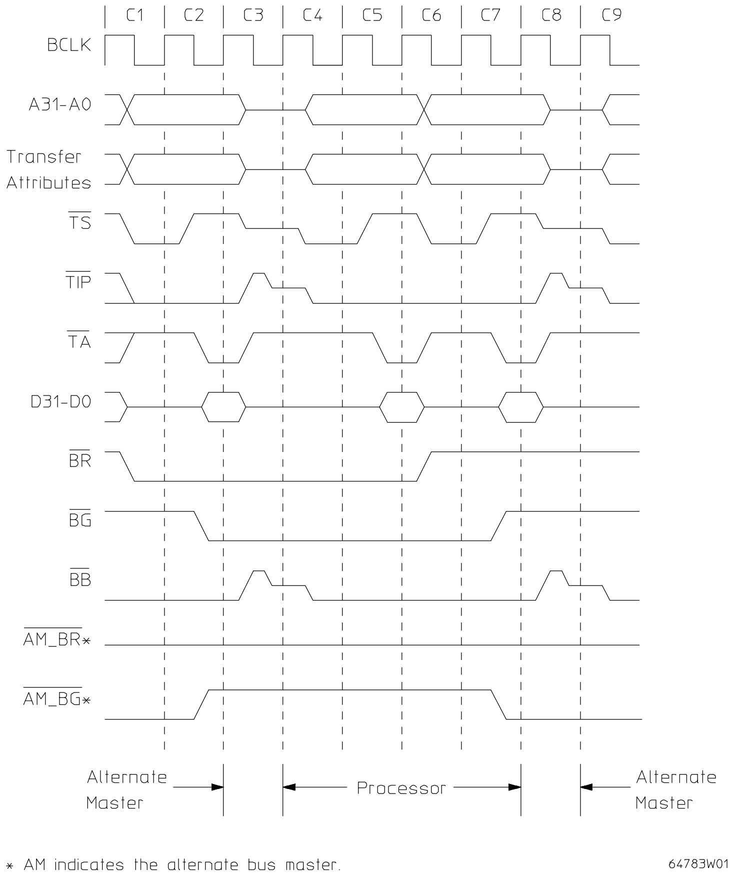
If the cycle where processing stops is part of a burst cycle, as indicated by the line access type in the status display, there are several things to check.

```
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 $\overline{\text{TA}}$ 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.



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.



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 thru 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 <RETURN>.
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.

1 Run your target program following the procedure in the previous steps.

2 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

3 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 $\overline{\mathrm{TS}}$ and $\overline{\mathrm{TIP}}$ 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 $\overline{\mathrm{TA}}$, interlock the monitor to the target strobes. The target system must not respond with $\overline{\mathrm{TEA}}$ 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 the chapter on 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 $\overline{\text{TA}}$ and $\overline{\text{TEA}}$ 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
```

16

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.
- Verifying the installation and performance of the emulator.
- Ensuring software compatibility
- List of replaceable parts.

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 64704A 80-Channel Emulation-Bus Analyzer card.
- HP 64700 Card Cage.

Optional parts are:

- HP 64172A 256-Kbyte Memory Modules or HP 64172B 1-Mbyte Memory Modules (which provide emulation memory).

Equipment and tools needed

In order to install and use the MC68040 emulation system, you need:

- Flat-blade screwdriver.

Support Services information is at the back of each manual binder

Installation overview

The steps in the installation process are:

1. Connect the HP 64783A/B emulator probe to the HP 64748C emulator control card.
2. Install cards into the HP 64700 card cage.
3. Install emulation memory modules on the emulator probe.
4. Connect the emulator probe to the demo target system.
5. Apply power to the HP 64700 Card Cage.

Your emulation and analysis system may already be assembled (depending on how parts of the system were ordered).

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 are provided in the HP 64700 Card Cage Installation/Service manual. Instructions for installing and updating emulator firmware are covered in the chapter titled "Installing/Updating Emulator Firmware" in this manual.

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 "To verify the performance of the emulator" 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.

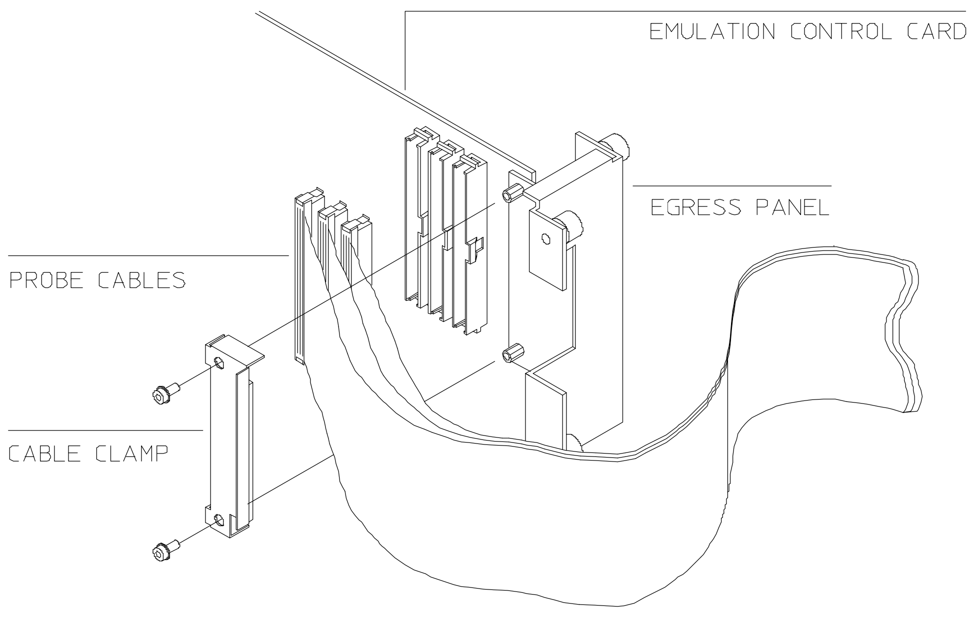
Step 1. Connect the Emulator Probe Cables

Three ribbon cables connect the HP 64748C emulation control card to the HP 64783A/B emulator probe.

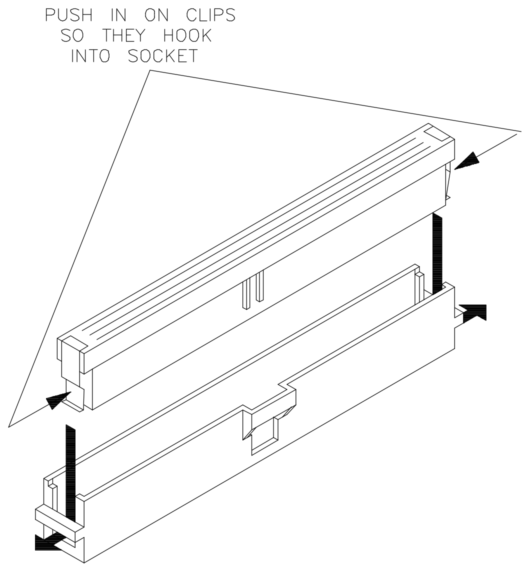
The shortest cable connects from J1 of the emulation control card to J3 of the emulator probe. The medium length cable connects from J2 of the emulation control card to J2 of the emulator probe. The longest cable connects from J3 of the emulation control card to J1 of the emulator probe.

Make sure the cable connectors are seated. There are stainless steel clips on the cable conntectors; 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).

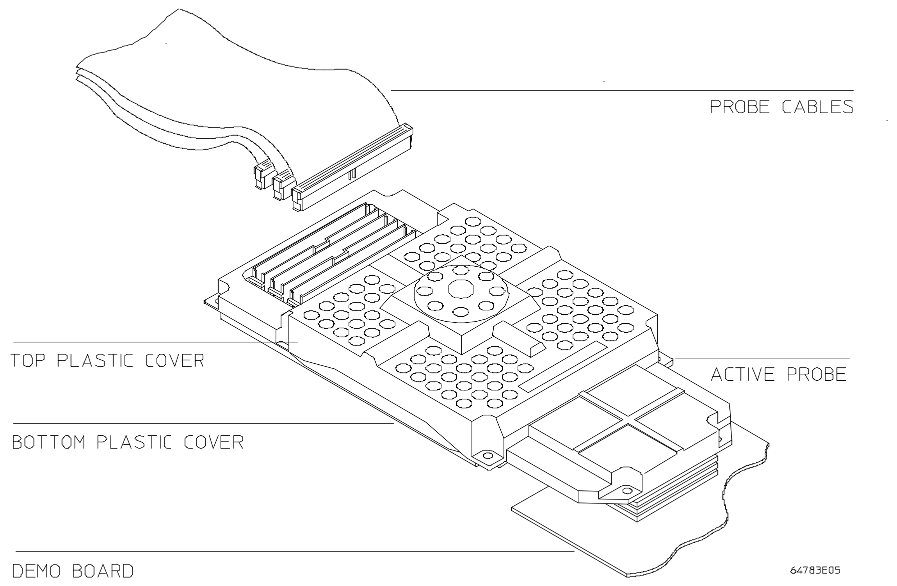
1 Connect the emulator probe cables to the emulation control card.



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 2. Install Boards into the HP 64700 Card Cage

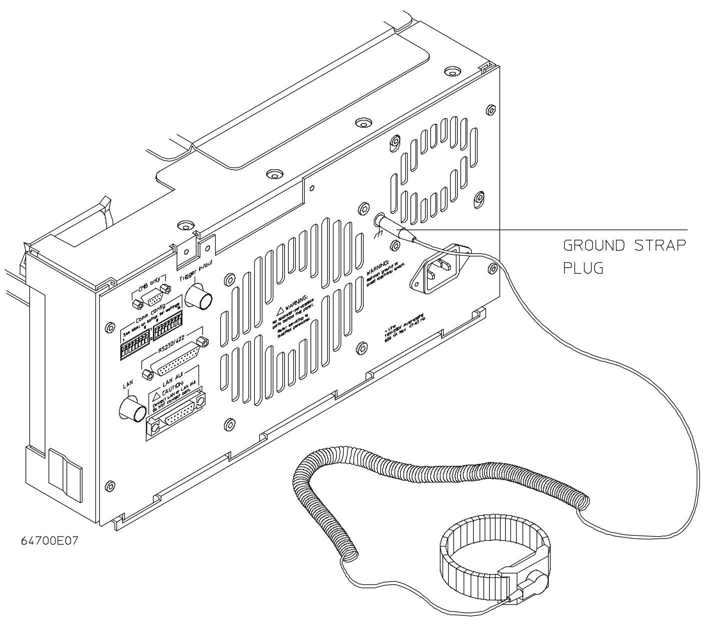
WARNING

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.

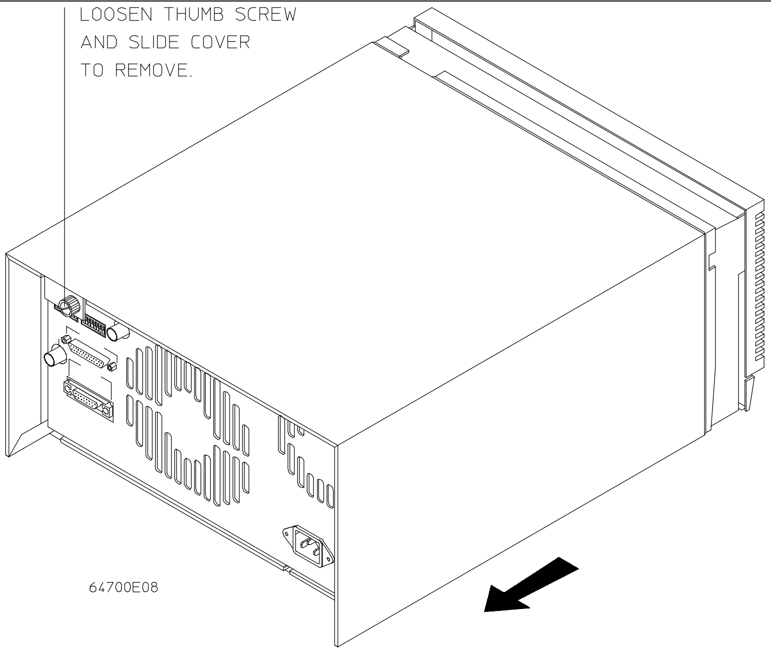
CAUTION

Do NOT stand the HP 64700 Card Cage on the rear panel. You could damage the rear panel ports and connectors.

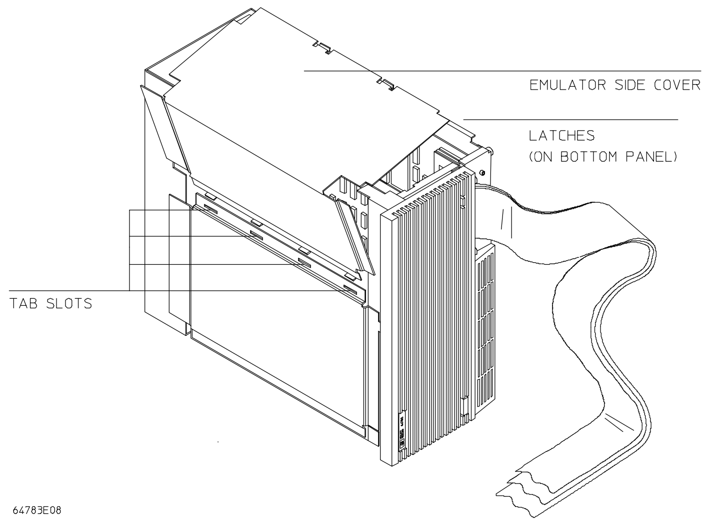
1 Use a ground strap when removing or installing boards into the HP 64700 Card Cage to reduce the risk of damage to the circuit cards from static discharge. A jack on the rear panel of the HP 64700 Card Cage is provided for this purpose.



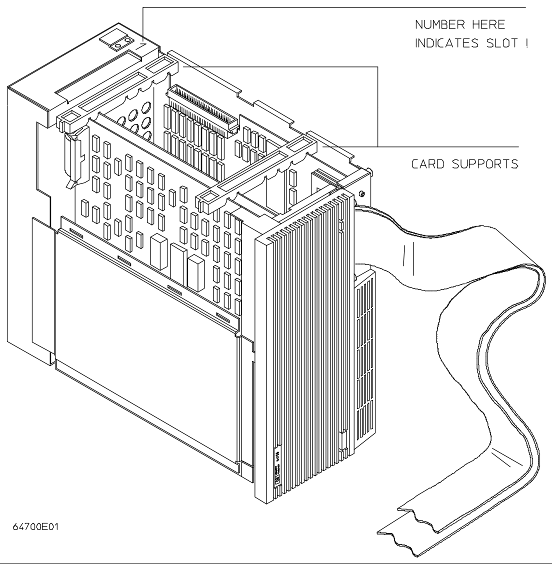
2 Turn the thumb screw and remove the top cover by sliding the cover toward the rear and up.



3 Remove the side cover by unsnapping the two latches and lifting off.

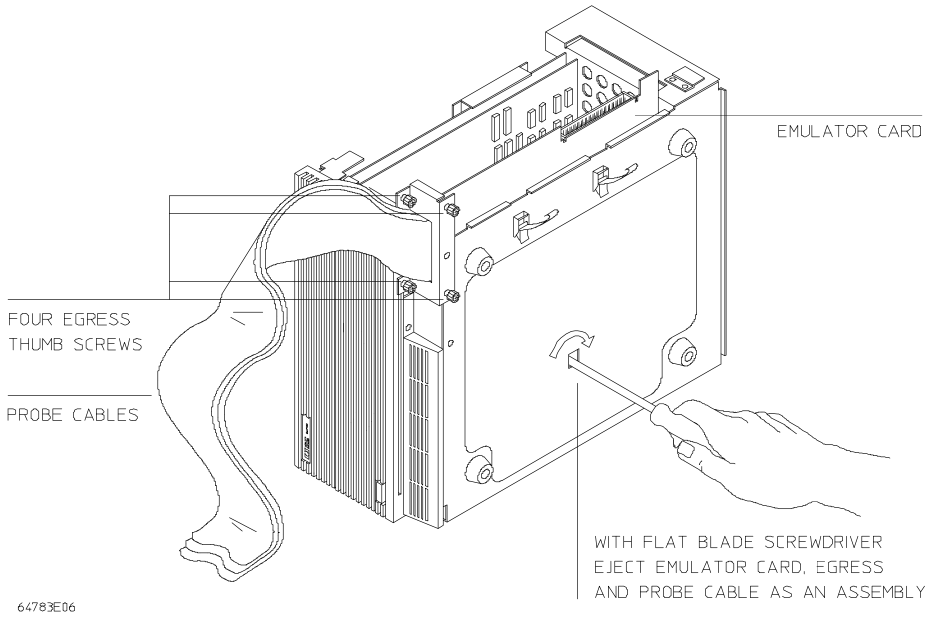


4 Remove the card supports.



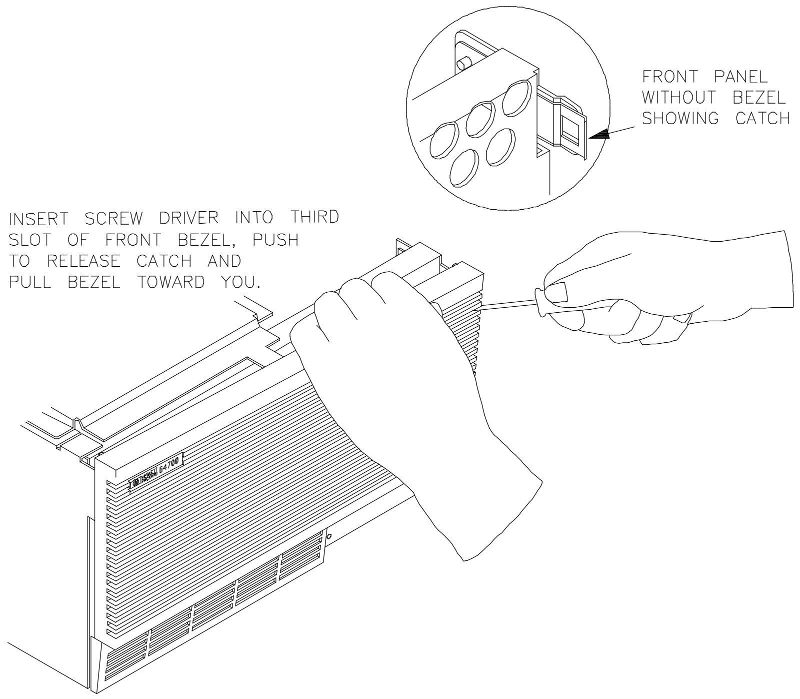
5 First, completely loosen the four egress thumb screws.

To remove emulator cards, insert a flat blade screwdriver in the access hole and eject the emulator cards by rotating the screwdriver.

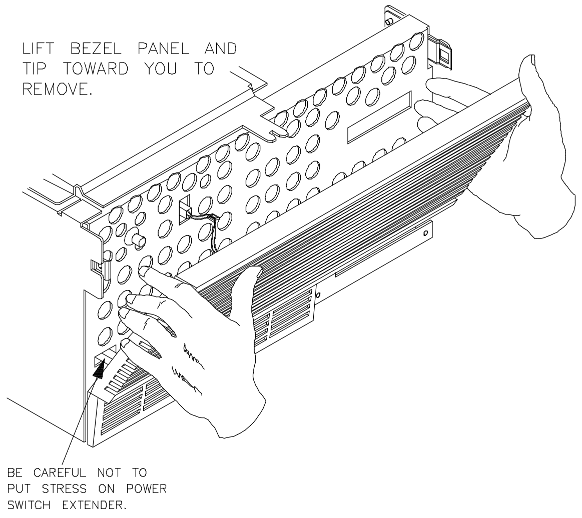

EMULATOR CARD
FOUR EGRESS THUMB SCREWS
PROBE CABLES
WITH FLAT BLADE SCREWDRIVER EJECT EMULATOR CARD, EGRESS AND PROBE CABLE AS AN ASSEMBLY
64783E06

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.

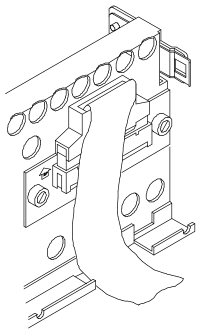
Be careful because the plastic ears are easily broken on the front bezel.


FRONT PANEL WITHOUT BEZEL SHOWING CATCH
INSERT SCREW DRIVER INTO THIRD SLOT OF FRONT BEZEL, PUSH TO RELEASE CATCH AND PULL BEZEL TOWARD YOU.

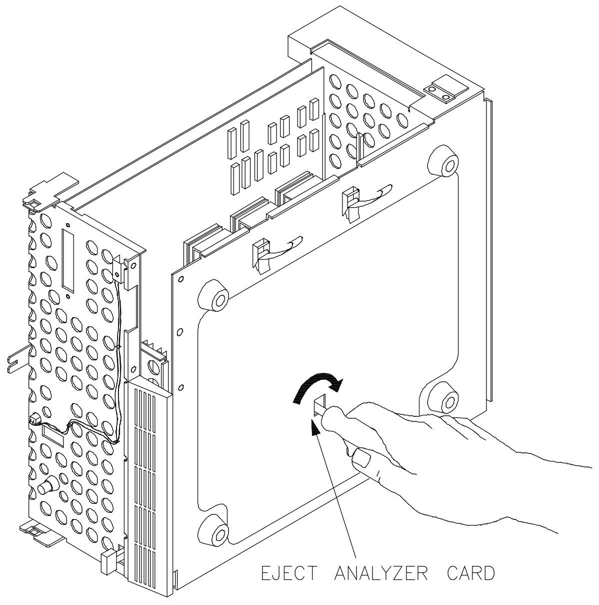
7 Lift the bezel panel to remove. Be careful not to put stress on the power switch extender.



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.

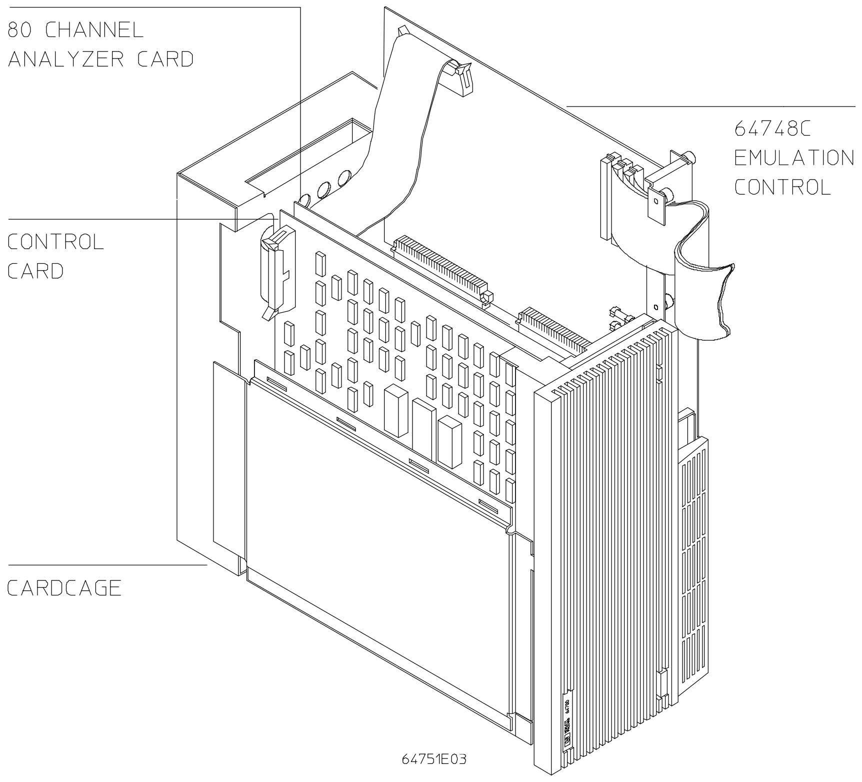


Do not remove the system control board. This board is used in all HP 64700 emulation and analysis systems.

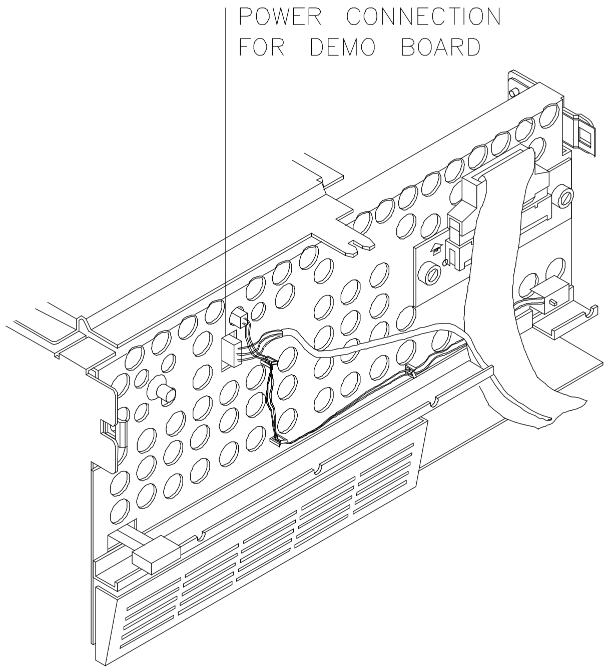
10 Install the analyzer and emulation control cards. The analyzer is installed in the slot next to the system control card. The emulation control card is installed in the second slot from the bottom of the card cage. These cards are identified with labels that show their model numbers and serial numbers. 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 mother board sockets. Check to ensure that the cards are seated all the way into the sockets. If the cards can be removed with your fingers, the cards are NOT seated all the way into the mother board socket.

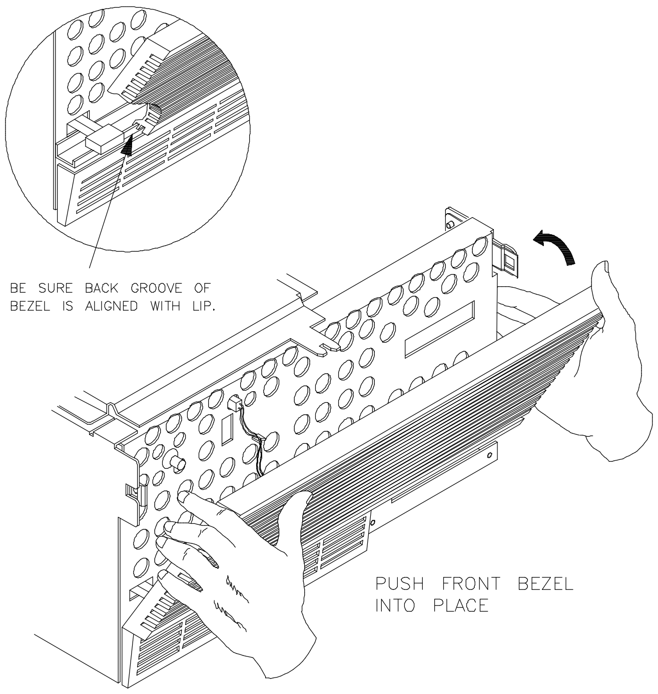
Attach the cable from the emulation control card to the analyzer card, and to the software performance analyzer, if installed. Tighten the thumbscrews that hold the emulation control card to the cardcage frame.



11 Connect the +5 V power cable to the connector in the HP 64700 front panel.

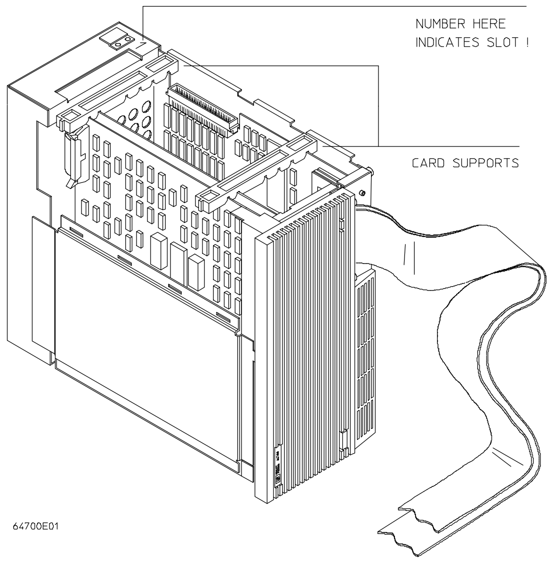


12 To reinstall the front bezel, be sure that the bottom rear groove of the front bezel is aligned with the lip as shown below.

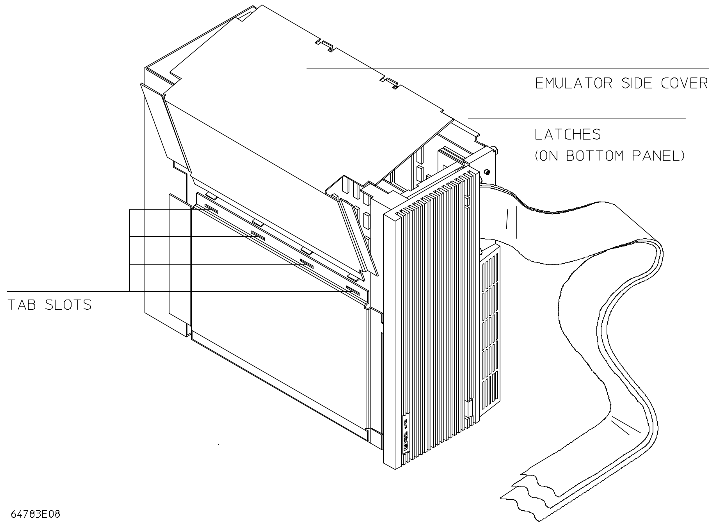

BE SURE BACK GROOVE OF BEZEL IS ALIGNED WITH LIP.
PUSH FRONT BEZEL INTO PLACE

13 If you wish to install the Flash card (used for updating firmware, see the Installing/Updating Emulator Firmware chapter), refer to the diagram above. Install the flash card in any available slot between the HP 64704A and the HP 64748C cards in the cardcage. Insert the flash card in the plastic guides. Make sure the connectors are properly aligned. Then press the card into the mother board sockets. Make sure the card is seated all the way into the sockets.

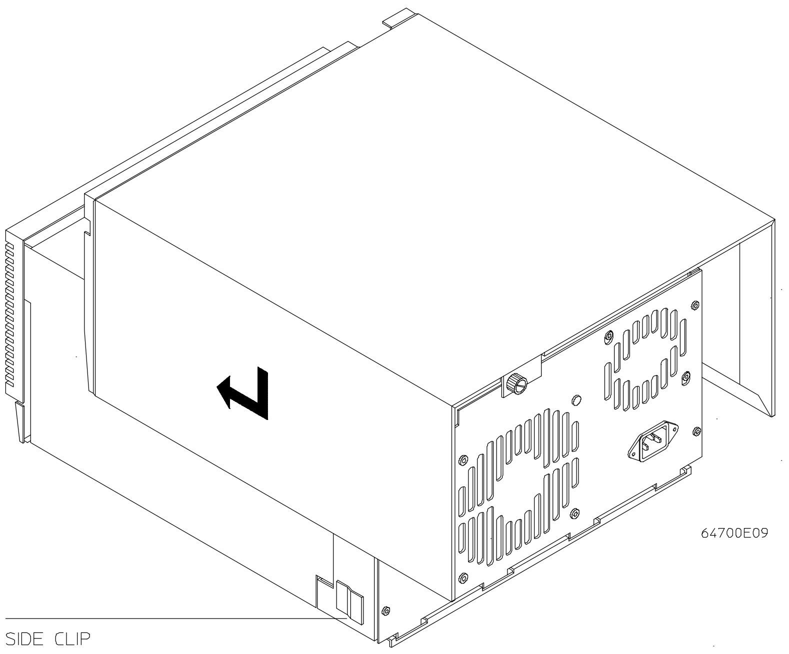
14 Install the card supports.



15 To install the side cover, insert the side cover into the tab slots and fasten the two latches.



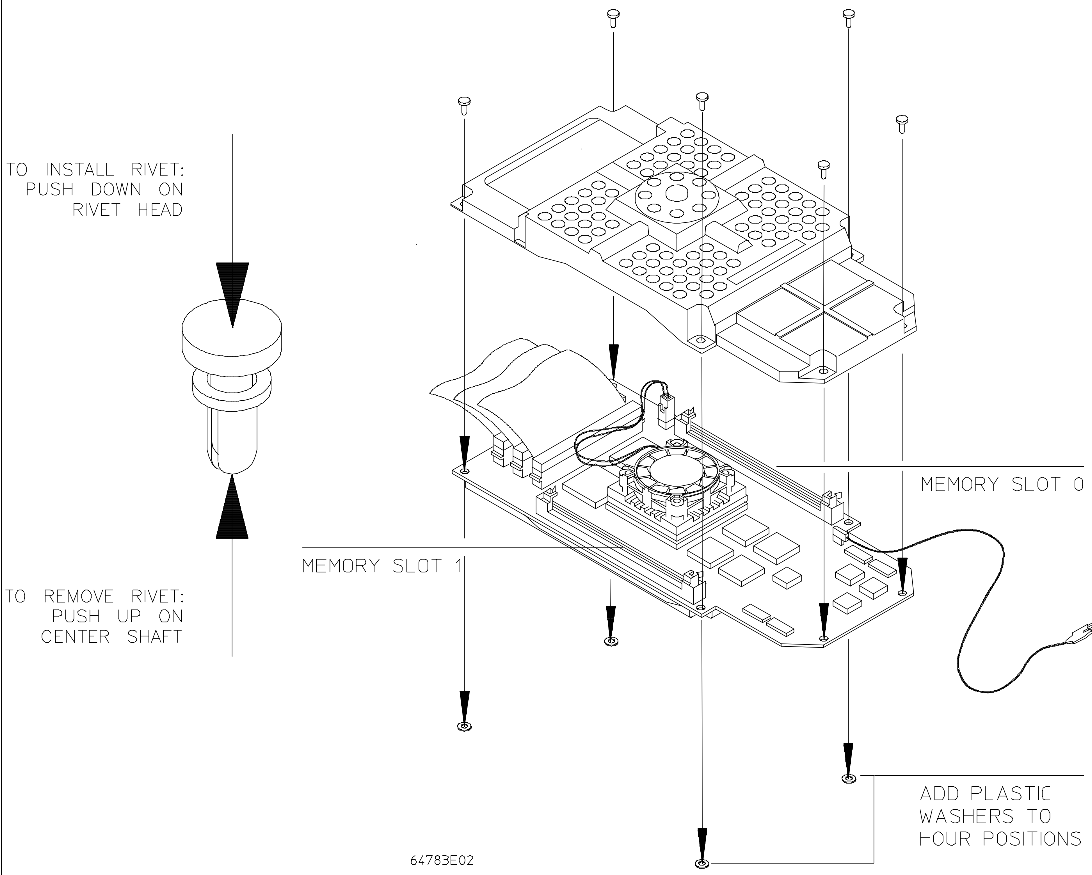
16 Install the top cover in reverse order of its removal, but make sure that the side panels of the top cover are attached to the side clips on the frame.



Step 3. 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.

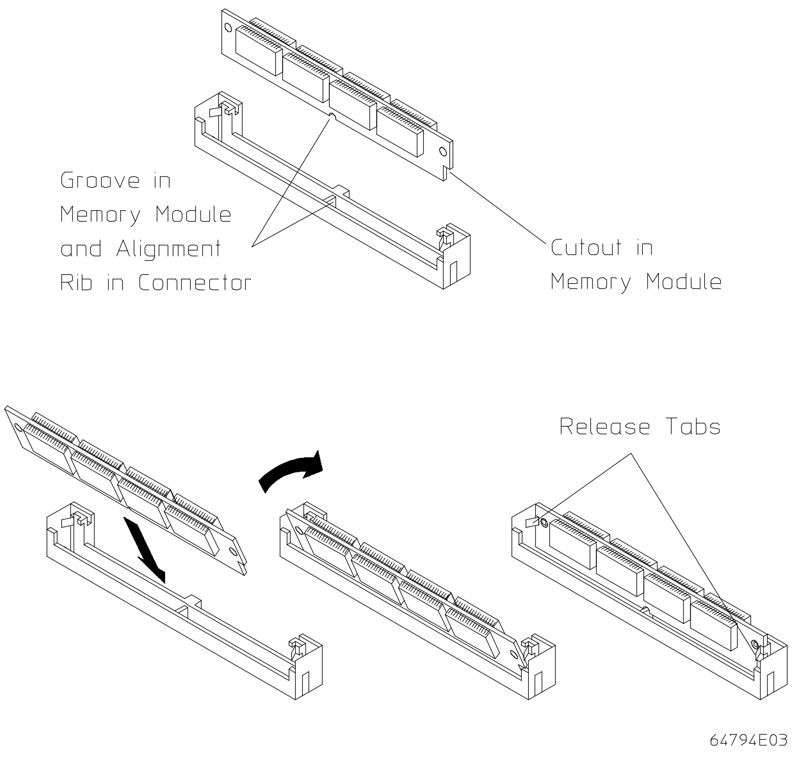
If you have two memory modules of different sizes, place the memory module with the greatest capacity in memory slot 0 to take advantage of the way memory slot 0 and memory slot 1 are divided by the emulator. For example, if you install a 1-Mbyte module in memory slot 0 and a 256-Kbyte module in memory 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.

Refer to the step called "To assign memory map terms" in the chapter titled "Configuring the Emulator" for details of the combinations of memory module installations.

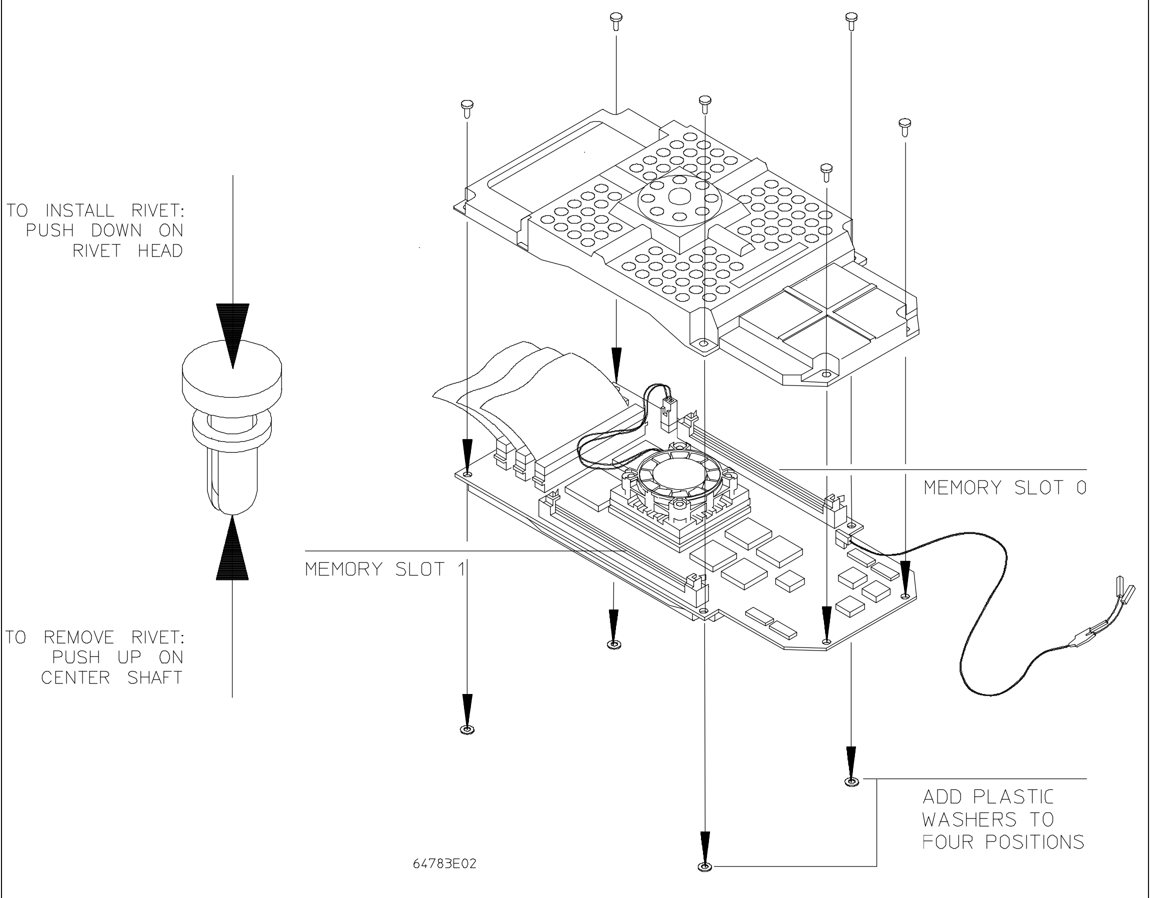
3 Install emulation memory modules on the emulator probe. There is a cutout at one end of the memory modules so they can only be installed the correct way.

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.

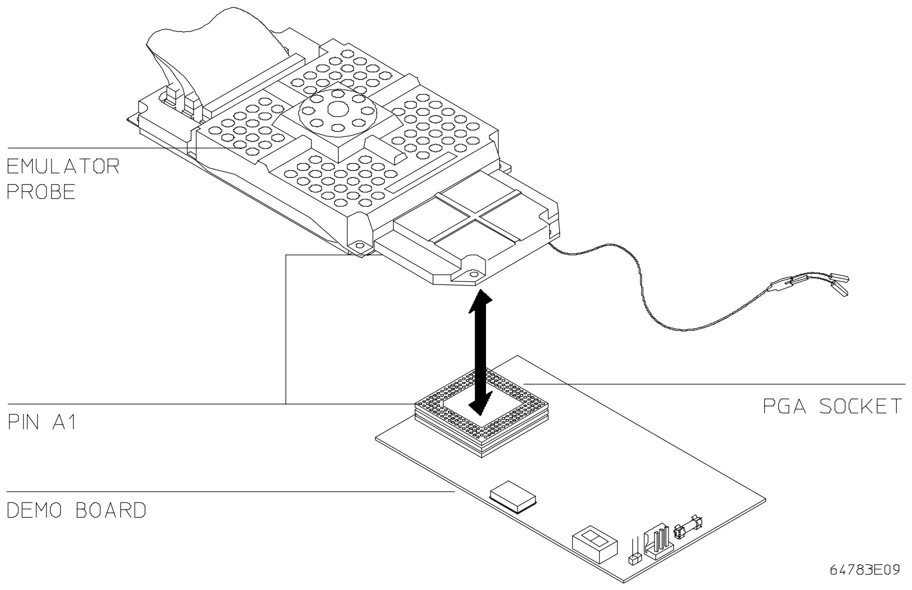


4 Replace the plastic cover, and insert new plastic rivets (supplied with the emulator) to secure the cover.

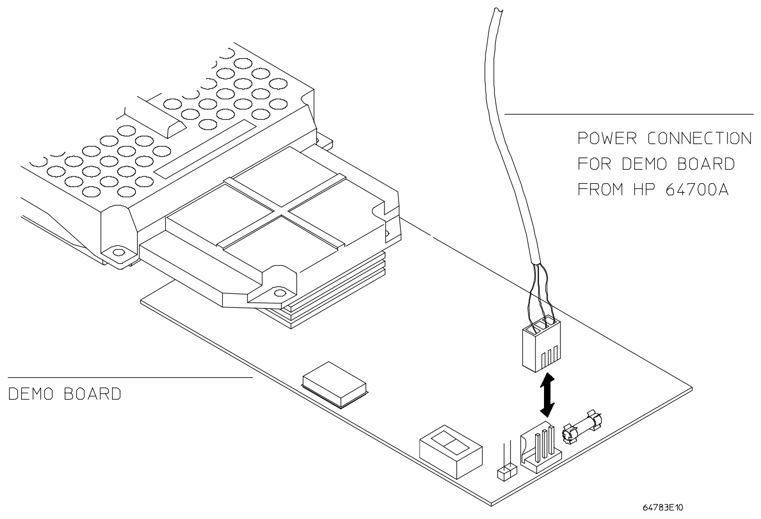

TO INSTALL RIVET: PUSH DOWN ON RIVET HEAD
TO REMOVE RIVET: PUSH UP ON CENTER SHAFT
MEMORY SLOT 0
MEMORY SLOT 1
ADD PLASTIC WASHERS TO FOUR POSITIONS
64783E02

Step 4. Connect the emulator probe to the demo target system

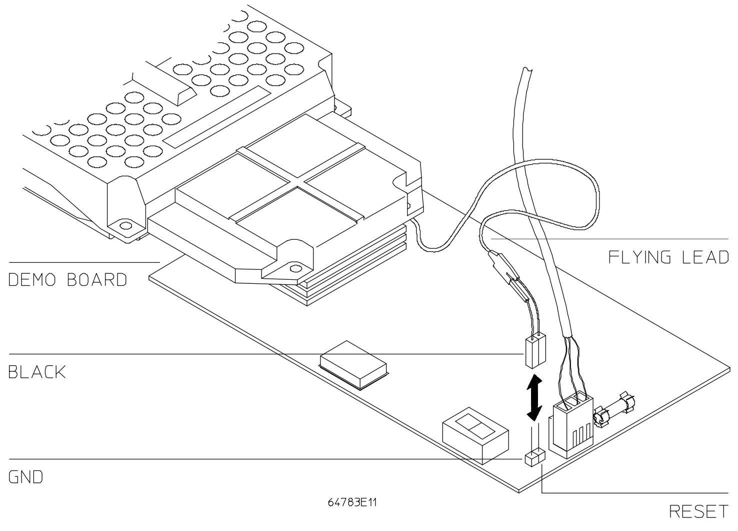
1 With HP 64700 power OFF, connect the emulator probe to the demo target system. When you install the probe into the demo board, be careful not to bend any of the pins. Do not insert the probe of the MC68040 emulator into the demo board socket incorrectly. Be very careful.



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.



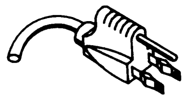
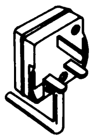
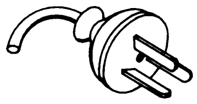
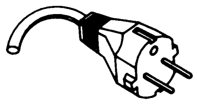
Step 5. Apply power to the HP 64700

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.

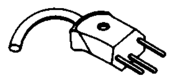
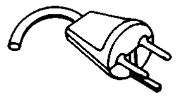
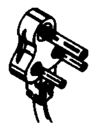
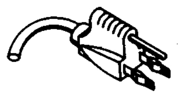
If the cable you received is not appropriate for your electrical power outlet type, contact your Hewlett-Packard sales and service office.

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^{o}	90/228 90/228	Jade Gray Jade Gray
Opt 900 250V 	8120-1351 8120-1703	Straight * BS136A 90^{o}	90/228 90/228	Gray Mint Gray
Opt 901 250V 	8120-1369 8120-0696	Straight * NZSS198/ASC 90^{o}	79/200 87/221	Gray Mint Gray
Opt 902 250V 	812001689 8120-1692 8120-2857	Straight * CEE7-Y11 90^{o} 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.

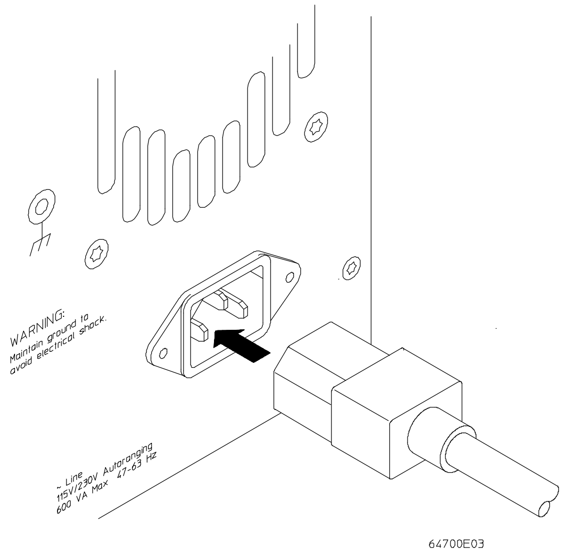
Power Cord Configurations (Cont'd)

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^o	79/20 79/200	Mint Gray Mint Gray
Opt 912 220V 	 8120-2957	Straight *DHCK107 90^o	79/200 79/200	Mint Gray Mint Gray
Opt 917 250V 	8120-4600 8120-4211	Straight SABS164 90^o	79/200 79/200	Jade Gray
Opt 918 100V 	8120-4753 8120-4754	Straight Miti 90^o	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 push button located at the lower, left-hand corner of the front panel. To turn ON power to the HP 64700, push the line switch button in to the ON (1) position. The power light at the lower, right-hand corner of the front panel will be illuminated.



To 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.

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 system terminal, and obtain an "R" or "R$" prompt.

8 Enter the command: **S***ystem* **T***erminal* **tcf -e** <return>. This sets the analyzer to easy configuration, which is required for performance verification.

9 Enter the command: **pv 1 <return>.**

Examples

Start the performance verification test routines from the PC Interface with the command:

S*ystem* **T***erminal* **pv 1**

A message similar to the following should appear:

```
Testing:  HP 64783 Motorola 68040 Emulator
     PASSED:
  Number of tests:  1          Number of failures:  0
Testing:  HP 64740  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 Emulator 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.

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.

To ensure software compatibility

There are various sets of firmware resident in the assemblies contained in the HP 64700 Card Cage. It is important to ensure that all the versions are compatible between the group of products you have installed. You can determine which versions of firmware you have by entering the Terminal Interface **ver** command, as follows: **S***ystem* **T***erminal* **ver**

There are at least three assemblies that have firmware in the HP 64700 Card Cage. These assemblies are the:

- Host Controller card
- Emulator card
- Analyzer card

If you purchased a complete Emulation/Analysis System from HP, you can be assured that all the products contained in the HP 64700 Card Cage contain compatible firmware at the time of sale. Software compatibility problems can occur when you swap the host controller card, emulator card, or analyzer card from one HP 64700 Card Cage to another, or from a recently purchased subassembly.

For example, you might purchase only the emulator subassembly (Emulation Control Card, Probe, and interconnecting ribbon cable) and replace the original emulator subassembly with the one you just purchased. In this case, the host controller may contain a version of firmware that is older than required to operate the new emulator; hence, compatibility problems can be caused by a newer emulator. All emulators will work with the latest software versions. The emulator software will warn you of incompatible software.

The HP 64700B Card Cage has Flash EPROM for the assemblies that may be installed.

The latest versions of firmware for the host controller card and analyzer card, along with a program called *progflash*, are part of the B1471 software for the HP 9000 workstation and Sun SPARCsystems and the HP 64700 Option 006 software for PCs.

When you load all your new versions of software onto your host computer, you are now ready to load the new version of firmware from your host computer to the assemblies that are in the HP 64700 Card Cage.

To load the new firmware, you use the *progflash* command. The progflash command displays a list of card cages, and subassemblies in each card cage on your system. From these lists, you can select which product to update. For information on using the *progflash* command, and updating your HP 64700 Series firmware, refer to the chapter titled "Installing/Updating Emulator Firmware" in this manual.

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
Analyzer Card HP 64740 with 1K memory depth	64740-66526	64740-69526
34-pin ribbon cable	64772-61602	

17

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 HP 64783A/B.

However, if you ordered the HP 64783A/B and the HP 64748C separately, or if you are using an HP 64748C that has been previously used with a different emulator probe, you must download the firmware for the HP 64783A/B into the emulation control card.

The firmware, and the program that downloads it into the control card, are included with the MC68040 emulator probe on the following MS-DOS format floppy disks:

- MC68040 EMULATION FIRMWARE 64783
- 64700 SW UTIL

The steps to install or update the emulator firmware are:

1. Connect the HP 64700 card cage to an IBM PC AT compatible computer's RS-232 serial port.

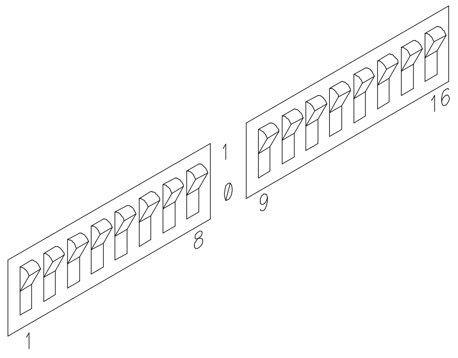
2. Install the firmware update utility and the 64783 emulator firmware.

3. Run "progflash" to update emulator firmware.

Step 1. Connect the HP 64700 to a PC host computer

1 Set the HP 64700 data communications configuration switches.

Set all "COMM CONFIG" (communications configuration) switches on the rear panel of the HP 64700 to the zero or open position.

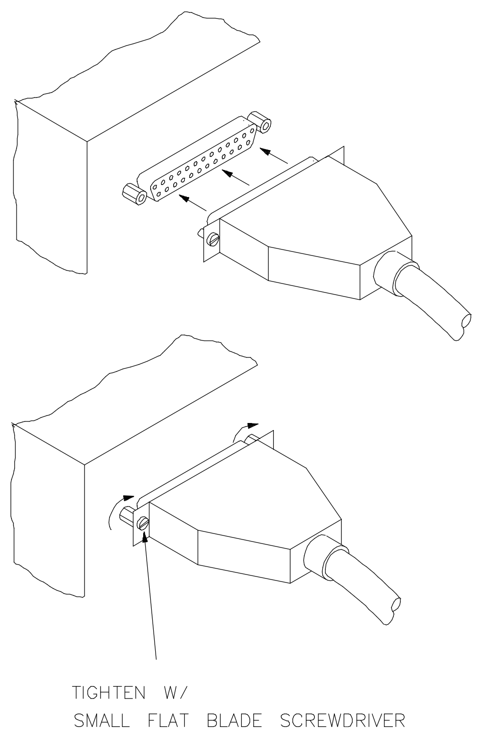


Note that switch settings are read as part of the HP 64700 power-up procedure. Any changes made to the switches after power-up will not be read until you turn the HP 64700 off and back on again.

2 Connect the RS-232 cable.

Recommended cables are HP 13242N (25-pin male to 25-pin male) or HP 24542M (9-pin female to 25-pin male) which are equivalent to a MODEM cable.

To connect cables to the HP 64700, simply align the cable with the serial port and insert the 25-pin male connector of the cable until it is firmly seated. Then tighten the holding screws on each side of the cable with a small, flat blade screwdriver. This will ensure that the cable pins and shield hood make good contact with the HP 64700 connector and will also guard against accidental disconnection of the cable.



Step 2: Install the firmware update utility

Your HP Vectra PC or IBM PC AT compatible computer must have MS-DOS 3.1 or greater and a fixed disk drive. The firmware update utility and the 64783 firmware require about 300 Kbytes of disk space.

1 Insert the 64700 SW UTIL disk into drive A.

2 Change MS-DOS prompt to drive A: by typing "A:" at the MS-DOS prompt.

For example:

```
C> A: <RETURN>
A>
```

3 Type "INSTALL" at the MS-DOS prompt.

For example:

```
A> INSTALL <RETURN>
```

After confirming that you want to continue with the installation, the install program will give you the option of changing the default drive and/or subdirectory where the software will reside. The defaults are:

```
Drive = C:
Directory Path = C:\HP64700
```

Follow the remaining instructions to install the firmware update utility and the 64783 firmware. These instructions include editing your CONFIG.SYS and AUTOEXEC.BAT files. Details follow in the next steps.

4 After completing the install program, use the PC editor of your choice and edit the \CONFIG.SYS file to include these lines:

```
BREAK=ON
FILES=20
```

BREAK=ON allows the system to check for two break conditions: <CTRL><Break>, and <CTRL>c.

FILES=20 allows 20 files to be accessed concurrently. This number must be at LEAST 20 to allow the firmware update utility to operate properly.

5 Edit the AUTOEXEC.BAT file to add:

```
C:\HP64700\BIN (to the end of the PATH variable)
SET HPTABLES=C:\HP64700\TABLES (as a new line)
SET HPBIN=C:\HP64700\BIN (as a new line)
```

Part of an example AUTOEXEC.BAT file resembles:

```
ECHO OFF
SET HPTABLES=C:\HP64700\TABLES
PATH=C:\DOS;C:\HP64700\BIN
```

6 If you are using the COM3 or COM4 ports, you will need to edit the \HP64700\TABLES\64700TAB file. The default file contains entries to establish the communications connection for COM1 and COM2. The content of this file is:

```
EMUL_COM1 unknown COM1 OFF 9600 NONE ON 1 8
EMUL_COM2 unknown COM2 OFF 9600 NONE ON 1 8
```

Either add another line or modify one of the existing lines. For example:

```
EMUL_COM3 unknown COM3 OFF 9600 NONE ON 1 8
EMUL_COM4 unknown COM4 OFF 9600 NONE ON 1 8
```

The "unknown" field usually specifies the processor type (which is "m68040" for the HP 64783 emulator), but you don't need to change this field in order to update the emulator firmware.

Software installation is now complete. The PC will need to be rebooted to enable the changes made to the CONFIG.SYS and AUTOEXEC.BAT files. To reboot, press the <CTRL><ALT><DEL> keys simultaneously.

Step 3: Run "progflash" to update emulator firmware

- Enter the PROGFLAS [-V] [EMUL_NAME] [PRODUCT] command.

The PROGFLAS command downloads code from files on the host computer into Flash EPROM memory in the HP 64700.

The -V option means "verbose". It causes progress status messages to be displayed during operation.

The EMUL_NAME option is the logical emulator name as specified in the \HP64700\TABLES\64700TAB file.

The PRODUCT option names the product whose firmware is to be updated.

If you enter the PROGFLAS command without options, it becomes interactive. If you don't include the EMUL_NAME option, PROGFLAS displays the logical names in the \HP64700\TABLES\64700TAB file and asks you to choose one. If you don't include the PRODUCT option, PROGFLAS displays the products which have firmware update files on the system and asks you to choose one. (In the interactive mode, only one product at a time can be updated.) You can abort the interactive PROGFLAS command by pressing <CTRL>c.

PROGFLAS will print "Flash programming SUCCEEDED" and return 0 if it is successful; otherwise, it will print "Flash programming FAILED" and return a nonzero (error).

You can verify the update by displaying the firmware version information.

Examples

To install or update the HP 64783 emulator firmware in the HP 64700 that is connected to the COM1 port:

```
C> PROGFLAS <RETURN>
```

```
              HP64700S006 A.05.00 03Jan94
                    64700 SW UTIL

           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 EMUL_COM1           unknown
         2 EMUL_COM2           unknown

Number of Emulator to Update? (intr (usually cntl C or DEL) to abort)
```

To update firmware in the HP 64700 that is connected to the COM1 port, enter "1".

```
           Product
         1 64783

Number of Product to Update? (intr (usually cntl C or DEL) to abort)
```

To update the HP 64783A/B MC68040 emulator firmware, enter "1".

```
Enable progress messages? [y/n] (y)
```

To enable status messages, enter "y".

```
Checking System firmware revision...
Mainframe is a 64700B

Reading configuration from '/hp64700/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: /hp64700/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
C>
```

You could perform the same update as in the previous example with the following command:

```
C> PROGFLAS -V EMUL_COM1 64783 <RETURN>
```

Glossary

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 the chapter on Using the Emulator in this manual 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 emulator because all analysis bits are used by the emulation-bus analyzer.

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.)

Command File

a file containing a sequence of commands to be executed.

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.

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.

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 the chapter on Using the Emulator in this manuyal 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.

Fork

temporary diversion from one program to start another program or process.

Form

the part of the PC interface screen that allows you to enter data for modifying various parameters.

Guarded Memory

an address range that is to be inaccessible to the emulation processor. The emulator will generate a break and display an error message if an access to guarded memory occurs.

Handshaking

a process that involves receiving and/or sending control characters which indicate a device is ready to receive data, that data has been sent, and that data has been accepted.

Host Computer

a computer to which an HP 64700 Series emulator can be connected. A host computer may run interface programs which control the emulator. Host computers may also be used to develop programs to be downloaded into the emulator.

Inverse Assembler

a program that translates absolute code into assembly language mnemonics.

Linker

a program that combines relocatable object modules into an absolute file which can be loaded into the emulator and executed.

Locked Exit

one of two methods used to leave the PC interface and return to the host computer operating system. A locked exit command allows you to exit the PC interface and re-enter later with the current configuration. (See also Unlocked Exit.)

Logging Commands

the process of automatically storing entered commands into a file. This file can later be used as a command file.

Logical Address Space

the addresses assigned to code during the process of compiling, assembling and linking to generate absolute files. Refer to Chapter 10 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.

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 10 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.

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.

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.

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.

Uploading

the transfer of emulation or target system memory contents to a host computer.

Unlocked Exit

one of two methods used to leave the PC interface and return to the host computer operating system. An unlocked exit command allows you to exit the PC interface and re-enter later with the default configuration. (See also Locked Exit.) This is not available in the Terminal Interface.

User Program

Another name for your target program (the program you are developing for your product.

Viewport

see Window.

Wait States

extra microprocessor clock cycles that increase the total time of a bus cycle. Wait states are typically used when slower memory is implemented.

Window

a specified rectangular area of virtual space shown on the display in which data can be observed.

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.