HEWLETT PACKARD

User's Guide for the Terminal Interface

68020/030/EC020/EC030 Emulators/Analyzer (HP 64748 and HP 64747)

Notice

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

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

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

Printing History

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

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

Edition 1	64748-97000, April 1991
Edition 2	64748-97004, July 1991
Edition 3	64748-97009, May 1992
Edition 4	64748-97012, December 1993
Edition 5	64748-97015, March 1994

The HP 64747 and HP 64748 Emulators

Description

The HP 64747A emulator supports the Motorola 68EC030 up to 40 MHz. The HP 64747B emulator, when used with the HP 64748C Emulation Control Card, supports the Motorola 68030. The HP 64748A/HP 64748D emulator supports the Motorola 68020 and 68EC020 microprocessors up to 33 MHz. The only difference between the HP 64748A and HP 64748D is that the HP 64748D can accept installation of 4-Mbyte SIMMs in its probe sockets and the 64748A cannot.

HP Emulator	**Control Card 64748B**	**Control Card 64748C**
64748 (68020/EC020)	Complete support.	Complete support.
64747A (68EC030)	Complete support.	Complete support.
64747B (68030/EC030)	Only MC68EC030 support.	Complete support.

The emulators plug into the modular HP 64700 instrumentation card cage and offer 80 channels of processor bus analysis with the HP 64794 or 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 the LAN or RS-422 interface. An RS-232 port and a firmware-resident interface allows 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 64748A/64748D

- 33 MHz active probe emulator
- Support for MC68020 and MC68EC020 (probe adapter required)

HP 64747A

- 40 MHz active probe emulator
- Supports MC68EC030
- Supports burst and synchronous bus modes

HP 64747B

- 40 MHz active probe emulator
- Supports MC68EC030 and MC68030
- Supports burst and synchronous bus modes

Both Emulators

- Symbolic support
- Execution (software) breakpoints
- 36 inch cable and 219 mm (8.8") x 102 mm (4") probe, terminating in PGA package
- Optional adapter for PQFP
- 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.)

Both Emulators (continued)

- Multiprocessor emulation
  - synchronous start of 32 emulation sessions
  - cross triggerable from another emulator, logic analyzer, or oscilloscope
- Coprocessor support - allows display and modification of FPU registers
- Demo board and self test module included

Emulation-bus analyzer

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

Emulation memory

- Several memory configurations can be obtained (from 256 Kbyte to 8 Mbyte) by installing optional memory modules on the emulation probe
- 4 Kbytes of dual-ported memory available if you use the background monitor
- Mapping resolution is 256 bytes
- HP 64748 (MC68020) :
  - No wait states out of dual-ported memory up to 33 MHz
  - No wait states out of target memory
  - No wait states out of emulation memory up to 25 MHz
  - 1 wait state out of emulation memory above 25 MHz
- HP 64747A (MC68EC030), and HP 64747B (MC68030/EC030):
  - No wait states for target system accesses up to 25 MHz
  - Three-cycle asynchronous or synchronous target accesses and two-cycle burst accesses above 25 MHz
  - Dual-port/monitor memory matches target system access speeds
  - Three-cycle asynchronous or synchronous emulation memory accesses and two-cycle burst accesses at all speeds.

In This Book

This manual covers the HP 64747 and HP 64748 emulators. All information in the manual applies to both emulators unless it is marked with the processor name (MC68020, MC68EC020, MC68030, or MC68EC030).

Emulator	Supports . . .	Also supports . . .
HP 64747A	MC68EC030	MC68030 (with MMU disabled)
HP64747B	MC68030	MC68EC030
HP 64748	MC68020	MC68EC020 (probe adapter required)

Part 1, “Quick Start,” shows you how to make some simple measurements with the emulator, using the built-in demo program. A short chapter in this part shows you how to fix the most common problems you might encounter when you first use the emulator.

Part 2, “Using the Emulator,” tells you how to use all the standard emulator commands to perform various measurement tasks with the emulator. Use this part of the manual after you have worked through the tutorials in part 1.

Part 3, “Reference Information,” is the place you should turn to when you are familiar with the emulator and want to make advanced measurements, such as using the 68030 MMU, or simply want to look up detailed syntax information for a command.

Part 4, “Installation and Service,” tells you how to install the emulator, connect it to the target system, and verify that it works correctly.

You should read the book *Concepts of Emulation and Analysis* when you have the chance to do so; it contains a good conceptual introduction to the emulation process, and also describes how an emulation monitor works. The book *HP 64700 Card Cage Installation/Service Guide* tells you more about installation and configuration of the HP 64700 Card Cage. If you have a problem with the emulator and don’t understand how to fix it, a listing for your local HP Sales and Service office is in the *Support Services Guide.*

Contents

10 Emulator Commands

11 Expressions

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 downloaded 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.

Configure the Emulator

Each target system has different resource requirements for memory and I/O locations. 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, “Quick Start Guide,” tells how to set up the emulator and how to begin making simple measurements. The chapter is organized as a practice tutorial, so that you can use the built-in demo program of the emulator to learn about emulator operation.

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

If you’re looking for more detailed information on emulator operation, see part 2.

Part 1

1

Quick Start

How to get the emulator ready for use and make a few simple measurements

The MC68020 and MC68030/EC030 emulators have many powerful features to help you debug and integrate target system software and hardware. The tutorial in this chapter will guide you through the steps of setting up the emulator and making a few basic measurements with the built-in demonstration (demo) program. The steps are a general guide to using the emulator, but you might not follow them every time that you use the emulator.

You can use the tutorial in a variety of ways. You may simply want to skim the material, then set up the emulator and use part 2 of this manual to learn about the different tasks you can do with the Terminal Interface, or you can work through the tutorial by following the instructions in each step. The first part of each step gives you some general information on the command or action. You can enter the commands given in the "Examples" part of each step to work through a measurement using the Terminal Interface.

The tutorial (and many examples throughout this book) uses a simple demo program. This program emulates a simple command interpreter. It has a one-byte input buffer for commands, and recognizes the ASCII characters "A" and "B." All other values are considered invalid. When you input a command to the buffer, the program calls a subroutine that interprets the command and writes a corresponding message to an output buffer. In the tutorial examples, you will enter a command by modifying memory, then view the results in the output buffer.

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

Step 1. Install the emulator and analyzer into the Card Cage

If you purchased the MC68020/68030/EC030 emulator as a complete system, the emulator and analyzer boards are preinstalled in the HP 64700 Card Cage at HP. However, if you purchased the MC68020/68030/EC030 emulator and analyzer boards separately, you must install them in the HP 64700 Card Cage before you can use the emulator.

- Install the emulator and analyzer boards according to the instructions in Chapter 15 of this manual.

  If you have any questions regarding the installation that are not answered by the instructions in Chapter 15, see the *HP 64700 Series Card Cage Installation/Service Guide.*

Step 2. Connect the HP 64700 Card Cage to the host computer

To communicate with the Terminal Interface, you need to connect the HP 64700 Card Cage to a terminal or a host computer that can emulate a terminal.

- Connect the emulator to a terminal, PC, or workstation by following the instructions in the *HP 64700 Series Card Cage Installation/Service Guide.*

Step 3. Connect the emulator to the demo board

1 Plug the PGA male connector on the emulator probe into the PGA female connector on the demo board, taking care to align pin A1 of each connector.

2 Connect power from the emulator to the demo board power socket using the cable provided.

The emulation probe must be connected to a target system (the demo board, in this procedure) to use the MC68020 or MC68030/MC68EC030 emulator. The emulator needs a powered-on system with a clock oscillator to function correctly.

The demo program examples in this manual will work correctly when you use the demo board for your target system. They may not work correctly with another target system due to the system memory map and the emulator configuration.

See Chapter 15, “Installation and Service,” for more information on the demo board and its power cable connections.

Step 4. Build a program

For this tutorial, you don’t need to build the demo program because it’s already resident in the emulator. When you’re ready to debug a target system using your own program, you can build the program in a format usable by the emulator by following these steps:

1 Create source files in “C” or MC68020/MC68030/EC030 assembly language using a text editor.

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

Translate the assembly source files to relocatable object files using a compatible MC68020/MC68030/EC030 cross assembler.

3 Link all relocatable object files with the linker/loader to produce an absolute object file in HP64000 (HP-OMF) format or Motorola S-record file format.

See Chapter 4, "Using the Emulator," for more information.

Example

To build the demo program with the HP 64870 Assembler/Linker:

```
as68k -h demo
as68k -h handle_msg
ld68k -h -c demo.k -o demo.x
```

To build the demo program with the Microtec Research assembler/linker on a PC:

```
asm68k -fd -fh -fcase -ft -L demo.s
asm68k -fd -fh -fcase -ft -L handler.s
lnk68k -c demo.k -fi -fh -fd -fs -L -o demo.x
```

Step 5. Apply power

Caution

Possible damage to the emulator! When you use the emulator with your own target system, always turn on power to the emulator before turning on power to your system. Otherwise, the emulator or your system might be damaged.

You must apply power to the HP 64700 Card Cage before you can use the emulator and the Terminal Interface.

- Apply power to the emulator by pressing the power ON button located on the front panel of the HP 64700 Card Cage.

Example

When you apply power to the emulator, you will see the following appear on your terminal screen:

```
                  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

  HP64748 Motorola 68020 Emulator
  HP64740 Emulation Analyzer
```

The emulator executes the powerup initialization procedure, then presents the Terminal Interface command prompt.

Step 6. Enter commands

To use the emulator, you enter various commands at the Terminal Interface prompt. These commands instruct the emulator to execute some action or to report information.

- Enter a command by typing it at the Terminal Interface prompt and pressing <**RETURN**> or <**Enter**>. (Use the key on your system that sends a carriage return).

Example

Display the emulator status:

```
R>es
```

Step 7. Get command help

If you need to know more about a Terminal Interface command, you can use the built-in **help** facilities.

- To display the main **help** menu, type: **help**

  The main help menu lists general syntax information for the Terminal Interface. It also lists the various command groups that have more help information available.

- To display help information for a particular command group, type: **help <group_name>**

  When you request help on a command group, the Terminal Interface displays a list of all commands in that group and a brief definition of each command.

- To display help information for a particular command, type: **help <command_name>**

  Help information for a specific command includes a list of command options with brief explanations. Some help modules also contain a series of short command examples.

  You can enter the **?** symbol for the word "**help**."

Examples

Display main help information:

```
R>help
```

Display help information specific to the MC68020/68030/EC030 emulator:

```
R>help proc
```

Step 8. Configure the emulator

Before you use the emulator to debug your target system hardware and software, you must configure the emulator for the way you use the MC68020 or MC68030/EC030 processor. This includes information on memory map and processor configuration. There are two primary commands that affect the emulator interface to your system.

- Use the **map** command to define memory resource usage.

  The emulator contains a memory mapper that is programmed to allocate emulation memory and control access to both emulation and target system memory. You use the **map** command to define the memory ranges that you want to use.

- Use the **cf** command and its options to set other emulator configuration items, including processor configuration and the target system interface configuration.

  The **demo** command automatically sets the proper emulator configuration for the demo program. Therefore, if you're using the demo program to learn about the emulator, you won't need to make any changes to the configuration. For any other system or program, you may need to make changes to the configuration. When you are ready to use the emulator to debug your system, see Chapter 7, "Configuring the Emulator."

Step 9. Load a program

If you're using emulation memory or your target system RAM to hold your user code, you can download programs from a host computer. This simplifies development; you can change your program on the host and load it into the emulator for immediate testing and verification.

- Use the **load** command to download absolute files (containing programs and/or data) from a host computer into the emulator and target system memory.

The **demo** command automatically loads the demo program into emulation memory from the emulator. Therefore, if you're using the demo program to learn about the emulator, you won't need to use the **load** command.

- Type **demo** to configure the emulator and load the demo program.

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

  See Chapter 4, "Using the Emulator," for more information on loading programs. See Chapter 10, "Emulator Commands," for more information on the **demo** command and the demo program.

Step 10. Run the program

Once you have built and loaded a program and configured the emulator, you are ready to run the program to test its functionality.

- To run a program from reset, type: **r rst**
- To run a program from the current program counter (PC) value, type: **r**
- To run a program from a specific address, type: **r <address>**

  <address> is a 32-bit expression. See <ADDRESS> in the "Expressions" chapter for information.

Examples

Run the demo program from reset:

```
R>r rst
```

The prompt changes to U> to show processor running in user mode.

Step 11. Modify memory

The demo program has a global variable called Cmd_Input that is used as the command input buffer. You can use the emulator features to modify this memory location, which effectively enters a command for the program to interpret.

- Enter the command "A" into the demo program by typing **m -db demo:Cmd_Input=41**.

  The emulator can handle symbolic address arguments, such as **demo:Cmd_Input**. These symbols must be defined to the emulator by using the **sym** command or by downloading a symbol table. The **demo** command automatically defines the correct symbols for the demo program. See the "Symbols" section in Chapter 4 of this manual for details.

  The **-db** argument to the **m** (memory) command says that the modification value (41 hex) is to be interpreted as a byte value. (For example, if you used the argument **-dw** instead, the value would be interpreted as 0041 hex.)

Step 12. Display memory

In Step 11, you modified the command input buffer variable to an ASCII "A." You can view the output buffer of the program by displaying memory to verify that the command was correctly interpreted by the program.

1 Display the demo program's message output buffer in byte format by typing: **m -db handle_msg:Msg_Dest..Msg_Dest+1f**

You will see:

```
000000533  43 6f 6d 6d 61 6e 64 20 41 20 65 6e 74 65 72 65
000000543  64 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

When you load a new section of program code, or modify code using the **m** command, you may want to verify that the code is what you expected. The emulator allows display of memory locations in mnemonic format.

2 Display the message handling routine for the demo program in mnemonic format by typing: **m -dm Int_Cmd..47e**

You will see:

```
00000042a  Int_Cmd          CMPI.B   #$41,D0
00000042e  -                BEQ.W    handle_msg:Cmd_A
000000432  -                CMPI.B   #$42,D0
000000436  -                BEQ.W    handle_msg:Cmd_B
00000043a  -                BRA.W    handle_msg:Cmd_I
00000043e  andle_msg:Cmd_A  LEA      handle_msg:Msg_A,A0
000000444  -                MOVEQ    #$00000010,D1
000000446  -                BSR.W    handle_msg:Print_Msg
00000044a  -                RTS
00000044c  andle_msg:Cmd_B  LEA      handle_msg:Msg_B,A0
000000452  -                MOVEQ    #$00000010,D1
000000454  -                BSR.W    handle_msg:Print_Msg
000000458  -                RTS
00000045a  andle_msg:Cmd_I  LEA      handle_msg:Msg_I,A0
000000460  -                MOVEQ    #$0000000E,D1
000000462  -                BSR.W    handle_msg:Print_Msg
000000466  -                RTS
000000468  e_msg:Print_Msg  LEA      handle_msg:End_Msgs,A1
00000046e  andle_msg:Again  MOVE.B   (A0)+,(A1)+
000000470  -                DBEQ     D1,handle_msg:Again
000000474  e_msg:Fill_Dest  MOVE.B   #$00,(A1)+
000000478  -                CMPA.W   #$0553,A1
00000047c  -                BNE.B    handle_msg:Fill_Dest
00000047e  -                RTS
```

Notice that you can mix symbols and numeric values in ranges.

When you use the **-dm** 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.

Step 13. Start and display a trace

The *emulation-bus analyzer* is a powerful tool that allows you to view the execution of your program in real-time. You can have the analyzer trigger on a specific state so that it captures only the information you need. That way, you won't spend time searching through trace lists of unimportant information.

1 Define a simple trigger on the address value of the symbol Print_Msg by typing: **tg addr=handle_msg:Print_Msg**

Often, you'll want the analyzer to recognize its trigger when a certain program location is reached. You can use either a simple address expression, or one that includes symbols.

The analyzer doesn't begin examining data until you start a trace.

2 Begin an emulation-bus analyzer trace by typing: **t**

When you start a trace, the analyzer begins searching for the trigger specification while saving data according to your storage specifications. Once the trigger has been found, you can display any states that are stored.

The program code at Print_Msg is executed only when the program interprets a command. To set up this condition, you'll need to enter a command. (The processor must be running, enter **r** if it isn't.)

3 Enter the command "B" for the sample program by typing: **m -db demo:Cmd_Input=42**.

Once you enter the command, the analyzer will find the trigger condition and fill the trace buffer with program states.

After the analyzer completes the trace measurement, you can display the trace to view the execution of the program.

4 Display the disassembled and dequeued trace list with symbols and addresses by typing: **tl -e -d -od 0**

You will see:

```
 Line   addr,H    68020 Mnemonic                              count,R  seq
-----   --------  ----------------------------------------  ---------  ---
    0   rint_Msg  LEA      handle_msg:End_Msgs,A1                ---   +
    2   00000ff8   $0000          supr data long wr (ds16)    0.240 uS   .
    3   00000ffa   $0458          supr data word wr (ds16)    0.120 uS   .
    5   sg:Again  MOVE.B   (A0)+,(A1)+                        0.360 uS   .
       =sg:Msg_B   src  sdata rd:$45
       =End_Msgs   dest sdata wr:$45
    6   00000470  DBEQ     D1,handle_msg:Again TAKEN          0.120 uS   .
   13   sg:Again  MOVE.B   (A0)+,(A1)+                        1.040 uS   .
       =00000514   src  sdata rd:$6E
       =00000534   dest sdata wr:$6E
   14   00000470  DBEQ     D1,handle_msg:Again TAKEN          0.120 uS   .
```

The trace list is a 512 or 1024 state deep buffer (depending on whether you time stamp the states with **tcq**). You can selectively display portions of the buffer using the **tl** command. See Chapter 5, “Using the Analyzer.”

Step 14. Break to monitor

The emulation monitor is a program that provides various emulation functions, including register access and target system memory manipulation. The emulator must break program execution into the monitor before executing certain emulation commands, such as those accessing registers, or target system memory. You also can use the break command to pause target program execution.

- Break the emulation processor into the monitor by typing: **b**

The prompt changes to M> to show that the processor is running in the monitor.

You can use either a foreground or background monitor. See Chapter 7, “Configuring the Emulator,” for more information.

Step 15. Reset the emulator

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, “Configuring the Emulator,” for more information.)

Sometimes you may want to reset the emulation processor. This may be done from the emulator or the target system.

- To reset the emulation processor from the emulator, type: **rst**

  The prompt will change to R>.

2

Troubleshooting

Finding out what’s wrong and fixing it

This chapter explains how to diagnose and solve simple problems you might encounter when you first get started using the emulator. It doesn't explain how to solve more complex problems or how to interpret error messages. See Part 2 of this manual for more comprehensive problem solutions.

If the demo program won't work

- ☐ Check to be sure that you have the emulator plugged into the demo board, with power connected to the demo board from the emulator. (The demo program will not work with target systems other than the demo board.)
- ☐ Make sure the switches on the demo board are set to the OCE position (Out-of-Circuit Emulation, away from TEST).
- ☐ Make sure you initialized the emulator (**init -p**), and then executed the **demo** command to load the program and configure the emulator.

If you don't see a prompt

- ☐ Make sure that the power cable is connected to the Card Cage and that the front panel power switch is ON.
- ☐ Make sure that the communications channel settings are correct for the data communications setup and cabling that you are using.
- ☐ Make sure that you are using the correct data communication cable and that it is properly connected from your terminal or host computer to the HP 64700 Series Card Cage.

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

If you see an unfamiliar prompt

The emulator uses several different characters before the prompt string to provide status information (for example, R> means emulator reset). A complete list of these prompts is in Chapter 3, "Using the Terminal Interface." If you are unable to find the problem from that information, check the following items:

- ☐ To function correctly, the emulator must be plugged into a powered on target system with a clock signal. (Apply power to the emulator before applying power to the target system.) You must use the demo board supplied with the emulator for the demo program.
- ☐ Make sure your system is not holding the emulator in a wait or hold state.
- ☐ Make sure that the emulator is properly configured for your system requirements. See Chapter 7, "Configuring the Emulator."
- ☐ Make sure that the emulation monitor is configured correctly. If you want to use a background monitor, choose **cf mon=bg**. If you want to use a foreground monitor, choose **cf mon=fg**. Choose **cf mon=<address>** to set the starting address of the monitor. **<address>** must be on a 4K boundary. The foreground monitor **<address>** range cannot be used by your programs.
- ☐ Try running performance verification (pv) to verify that the emulator and emulation controller are functioning correctly. See Chapter 15, "Installation and Service."

If the emulator displays a prompt, but doesn't respond to commands

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

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 find the source of the problem, contact your local HP Sales and Service Office for assistance.

If you can't load the demo program

- ☐ Make sure that the emulator is connected to the demo board, not some other system.
- ☐ Try reinitializing the emulator (**init -p**), then reenter the **demo** command.

If you can't load a program

- ☐ Make sure that the configuration is correct. See Chapter 7, "Configuring the Emulator," for more information.
- ☐ Make sure that the memory map is defined correctly for your program resource needs. See Chapter 7, "Configuring the Emulator," for more information.
- ☐ Make sure that the emulator is properly connected to your target system (the demo board, in this case) and that the system is powered-on. Also, if the memory map references target system resources, there must be target system hardware in the ranges defined by the map.
- ☐ Check the **load** command syntax and the absolute file format to make sure that you are using the correct options.
- ☐ Make sure that you are using the correct load procedure for the emulator communications configuration. See Chapter 4, "Using the Emulator," for examples of different configurations and the appropriate load procedures.

If the emulator won't run the program

- ☐ Make sure that you have configured the emulation monitor correctly. If you want to use a background monitor, choose **cf mon=bg**. If you want to use a foreground monitor, choose **cf mon=fg**. Choose **cf mon=<address>** to set the starting address of the monitor. **<address>** must be on a 4K boundary. The foreground monitor **<address>** range cannot be used by your programs.
- ☐ Check the general emulator configuration, including chip and bus configuration. See Chapter 7, "Configuring the Emulator," for more information.
- ☐ Check the emulator memory map to verify that it matches the resource needs of the program. If the program and map rely on resources in your target system, make sure that the emulator is properly connected to a powered-on target system.

- ☐ Check to make sure that you have correctly specified the address for the run command.

If you can’t break to the monitor

- ☐ Make sure that you have configured the emulation monitor correctly. If you want to use a background monitor, choose **cf mon=bg**. If you want to use a foreground monitor, choose **cf mon=fg**. Choose **cf mon=<address>** to set the starting address for the monitor.
- ☐ Try initializing the emulator (**init -p**), or cycle power to the emulator.
- ☐ Run performance verification (**pv**) to test the emulation controller.

If the emulator won’t reset

- ☐ Use the **es** command to see if the target system is holding the processor in the reset state.
- ☐ Try initializing the emulator (**init -p**), or cycle power to the emulator.
- ☐ Run performance verification (**pv**) to test the emulation controller.

Part 2

Using the MC68020/MC68EC020 or MC68030/MC68EC030 Emulator

Making Measurements

When you've become familiar with the basic emulation process (see part 1 of this manual), 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 view program activity.

In This Part

Chapter 3, "Using the Terminal Interface," tells you how to use the Terminal Interface commands.

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

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

Chapter 6, "Making Coordinated Measurements," shows you how to use multiple emulators, analyzers, oscilloscopes or other measurement tools to make complex measurements.

Chapter 7, "Configuring the Emulator," explains how to use the Terminal 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.

If you're looking for a general introduction to using the emulator, see part 1. Reference information on the emulator is in part 3.

3

Using the Terminal Interface

How to set up the emulator and enter commands in the terminal interface

The Terminal Interface provides all the commands you need to make emulation and analysis measurements. The interface includes tools for emulator initialization, command entry and recall, and command help.

The steps in the emulation process are as follows:

1 Develop your program as described in Chapter 4.

2 Set up the emulator hardware and software as described in Chapter 15.

3 Connect the emulator to the demo board or other system. (See Chapter 15 and Chapter 5).

4 Apply power to the emulator.

5 Configure the emulator as needed for your system and programs. See Chapter 7.

6 Use the Terminal Interface commands to load, run and debug your programs. See Chapters 3, 4 and 5.

Using the Interface

The Terminal Interface is a command-line interface. By using the Terminal Interface commands, you can control HP 64700 Card Cage system functions and the emulator-specific functions.

This section tells you how to enter, recall and edit Terminal Interface commands. It also explains a few system commands that you may want to use.

The Terminal Interface displays different prompts to show you the current emulator status. The prompts are shown in the following table.

Command Prompt	Meaning
c>	No clock source from the emulated system.
R>	The processor is being reset from the emulator.
r>	The processor is being reset from the emulated system.
g>	The processor has granted the bus to the emulated system (BG or BGACK is asserted).
h>	In the MC68020 emulator, either an external device is asserting HALT, or the processor is asserting HALT (due to a double bus fault). In the MC68030/EC030 emulator, the processor has double bus faulted.
b>	No bus cycles are occurring.
U>	The processor is executing a target (user) program.
M>	The processor is executing the emulation monitor.
p>	No power from the emulated system.
W>	The emulator is waiting for a CMB READY signal. See Chapter 6.

w>	The processor is waiting for cycle termination from the target system.
?>	The emulator is in an unknown state. You will probably need to use the **rst** or **init** command or cycle power to reinitialize the emulator.

To apply power

- Apply power to the emulator by pressing the power ON button located on the front panel.

Examples

Apply power to the MC68020 emulator. You will see the following appear on your terminal screen:

```
                  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

  HP64748 Motorola 68020 Emulator
  HP64704 Emulation Analyzer
```

(The display will be slightly different for the MC68030/EC030 emulator.) The emulator executes the powerup initialization procedure, and then presents the Terminal Interface command prompt. See “To Initialize the Emulator.”

To initialize the emulator

- To do a limited initialization of the emulator, type: **init**

- To do a complete initialization of the emulator, without verification of system controller and memory, type: **init -c**

- To do a complete initialization of the emulator, with verification of memory and system controller, type: **init -p**

  The **init** command does the following:

  - Resets the memory map.
  - Resets the emulation configuration items.
  - Resets the break conditions.
  - Clears software breakpoints.
  - Reloads the background monitor.

  The **-c** and **-p** options to the **init** command allow a more complete initialization of the emulator, as follows:

  - The **init -c** command does a cold-start initialization, except that system controller performance verification tests are not executed.
  - The **init -p** command performs a powerup initialization, which is the same as cycling power. This includes emulator, analyzer, host controller, and communications port initialization, and host controller performance verification. It breaks the LAN connection before reporting the results of the initialization.

To enter commands

- Enter a command by typing it at the Terminal Interface prompt and pressing <**RETURN**> or <**Enter**>. (Use the key on your system that sends a carriage return).

- Recall commands in the reverse of the order that they were entered by pressing <**Ctrl**>**R**.

- Combine multiple commands on one command line by separating them with semicolons: **command1;command2;command3**

- Repeat a set of commands a certain number of times by typing: **rep <repeat_count> {<command_set>}**

  where <**repeat_count**> is an integer specifying the number of times to repeat the set of commands listed in <**command_set**>.

Examples

To load the demo program, enter:

```
R>demo
```

To enter a run command, enter:

```
R>r
```

To display several memory locations in mnemonic format and display registers, enter:

```
R>m -dm 0400..040f;reg
```

To display the emulator status and analyzer trace status, enter:

```
R>es;ts
```

To display various analyzer settings, enter:

```
R>tcf;tck;tsto;tg;tgout
```

To load the demo program, then execute the command **s 1;reg** three times, enter:

```
R>demo;rep 3 {s 1;reg}
```

This loads the demo program, and then causes the emulation processor to step and display registers three times. The first step is from the current program counter address.

To recall commands

- To recall commands in the same order that they were entered, press **<Ctrl>B**.
- To recall commands in the reverse of the order that they were entered, press **<Ctrl>R**.

The command line buffer allows you to recall previously entered commands to the command line. To execute the command, press **<RETURN>** or **<Enter>**.

If you want to edit the commands that you recall from the buffer, it's easiest to use the command line editing feature of the Terminal Interface. See "To enable command line editing." The command line editing feature has different recall commands. See "To edit a command."

To repeat commands

- Repeat a set of commands a certain number of times by typing: **rep <repeat_count> {<command_set>}**

where **<repeat_count>** is an integer specifying the number of times to repeat the set of commands listed in **<command_set>**. (A **<repeat_count>** of 0 continues repeating the commands until you enter a **<Ctrl>c**.)

Example

Load the demo program, then execute the command **s 1;reg** three times:

```
R>demo;rep 3 {s 1;reg}
```

This loads the demo program, then causes the emulation processor to step and display registers three times. The first step is from the current program counter address.

To enable or disable command line editing

1 To install the command line editing feature, type: **cl -e**

2 To remove the command line editing feature, type: **cl -d**

The command line editing feature allows you to use a simple set of commands to modify the command line. See “To edit a command.”

To edit a command

1 Press **<Esc>** to enter command editing mode. (Command line editing must be enabled. See “To enable or disable command line editing.”)

2 Use the commands listed in the following table to edit the command line. You can either edit an existing command or recall one to the command line using **<Esc>j** or **<Esc>k**. Or, you can use the commands shown in the table to search for a command in the buffer.

Command	Action
i	Insert before current character.
A	Append to end of line.
dd	Delete command line.
$	Move cursor to end of line.
^	Move cursor to start of line.
l	Move right one character.
j	Fetch next command.
a	Insert after current character.
x	Delete current character.
D	Delete to end of line.

Command	Action
0	Move cursor to start of line.
h	Move left one character.
k	Fetch previous command.
r	Replace current character.
/<string>	Find previous command matching <string>.
n	Fetch previous command matching <string>.
N	Fetch next command matching <string>.

3 When finished editing the command, press **<Enter>** to execute the command.

To get on-line help

- Display the main **help** menu by typing: **help**

  or

- Type: **?**

- To display help information for a particular command group, type: **help <group_name>**

- To display help information for a particular command, type: **help <command_name>**

  If you need quick reference information about a command or a set of commands, you can use the built-in **help** facilities. You can enter the **?** symbol in place of the word "**help**."

Examples

To display main help information, enter:

```
R>help
```

To display help information for the emulation command group, enter:

```
R>? emul
```

To display help information for the load command, enter:

```
R>help load
```

To display the emulator status

- Display the emulator status by typing: **es**

The emulation prompts can usually tell you most information about the emulator's status: whether the emulator is reset, running a user program, or running in monitor. (See "Using the Interface" for information on the different command prompts.) If you need more information than is given by the prompt, you can use the **es** command.

To set the date and time

- To display the current date and time setting, type: **dt**
- To set the date, type **dt <yymmdd>**, where **yy** is the last two digits of the year, **mm** is the month, and **dd** is the day of the month.
- To set the time, type **dt <hh:mm:ss>**, where **hh** is the hour in 24 hour format, **mm** is the minute of the hour, and **ss** is the second.

The HP 64700 Card Cage has a system clock that you can set using the **dt** command. The clock is reset when power is cycled. You can use the system clock for a variety of applications. For example, if you're logging the output of analyzer traces to a printer, you might want to insert the **dt** command at intervals so that the date and time will be printed with the trace listings.

Examples

Set the date to September 5, 1991:

```
R>dt 910905
```

Set the time to 1:05 P.M.:

```
R>dt 13:05:00
```

Display the date and time settings:

```
R>dt
```

To change the prompt

- Change the Terminal Interface command prompt to the string given by **<string>** by typing: **po -p "<string>"**

The standard command prompt is ">." You can change the prompt to suit your needs. Remember that the prompt is always preceded by a status character that identifies the emulator state. This character is not affected when you redefine the prompt.

Examples

The emulator prompt appears as follows when the emulator is reset:

```
R>
```

Change the prompt string to "<myemulator>":

```
R>po -p "<myemulator>"
R<myemulator>
```

To check the version of the Terminal Interface software

- Type **ver** to display the version numbers of the Terminal Interface system software and emulator software.

  The MC68020 or MC68030/EC030 emulator firmware must be used with the correct version of the emulation system and emulation analyzer firmware. See the paragraph titled, "To ensure software compatibility" in the Installation and Service chapter of this manual for more information.

To print strings on the output device

- Print a numeric expression or character string on the standard output device by typing: **echo <value>**

  where **<value>** may be a character string (enclosed in single or double quotes), a numeric expression, or a series of hex codes preceded by backslash (\) characters. (The hex codes are converted to their ASCII equivalents.)

  Occasionally, you may want to print a string on the standard output device (usually your terminal, but may be a printer or another device if you have redirected the standard output port.) The **echo** command allows you to do this.

  You can also use this command as a numeric calculator. The hex code character evaluation is useful for sending control strings to your terminal.

Examples

Send the string "Change the switch now" to your terminal:

```
R>echo "Change the switch now"
```

For an HP 2392A terminal, send the commands to home the cursor and clear the screen:

```
R>echo \1b "H" \1b "J"
```

Find the result of the bitwise AND operation on 08 hex and 28 hex:

```
R>echo 08&28
```

To insert delays in command processing

- To delay execution of the next operation until the next keystroke occurs (on the standard input port), type: **w**
- To delay execution of the next operation until the current measurement is completed, type: **w -m**
- To delay execution of the next operation for a time, type: **w <NN>**

  where **<NN>** is the number of seconds that you want to delay.

  Command delays are especially useful when you're using repeat loops, macros, or command files, and you need to modify some target system condition or write down results before the next command begins.

Examples

Initialize the emulator, and then load the demo program:

```
R>init -c

R>demo
```

Now use a command repeat: start a trace, wait for trace completion, display the resulting trace list, and then wait for any keystroke before the next iteration of the loop:

```
U>rep 0 {t;w -m;tl;w}
```

Cancel the repeat by typing **<Ctrl>C**.

Building and Using Macros

Macros can simplify repetitive command sequences. You can enter the command sequence once; then use the macro for the command sequence. Macros simplify trace measurements that require many run and trace commands, or setting up a particular emulator configuration each time you start a new measurement.

To create macros

- To create a macro referenced by **<name>**, type: **mac <name>={<cmd_list>}**

  where **<cmd_list>** is a series of Terminal Interface commands that are separated by semicolons (;).

  You can add parameters to macros. See the **mac** command in Chapter 10, "Emulator Commands" for more information.

Example

Define a macro that resets the emulator, loads the demo program, runs the demo program, and then modifies the Cmd_Input buffer:

```
R>mac setup={rst;demo;r;m -db demo:Cmd_Input=41}
```

Execute the set of commands in this macro:

```
R>setup
```

List the predefined configuration macros:

```
R>mac
```

To execute a macro

- To execute a macro, type the macro name at the command prompt.
- To prevent command information display during macro execution, type: **mac -q**
- To have macros execute with complete information on the commands in the macro, type: **mac -v**

Example

Execute the "setup" macro defined in the previous section:

```
R>setup
```

If you don't want the commands in the macro displayed, enter **mac -q** before entering the macro command:

```
R>mac -q
R>setup
```

Reenable command display during macro execution:

```
R>mac -v
```

To delete macros

- To delete a macro given by **<name>**, type: **mac -d <name>**
- To delete all macros, type: **mac -d ***

When you're finished using a macro, you should delete it. This frees emulator system memory for symbols, equates, and new macro definitions.

Example

Delete the macro named setup:

```
R>mac -d setup
```

Using Command Files

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

Building Command Files

To build and use a command file in the Terminal Interface, the HP 64700 Series Card Cage must be connected to a PC, workstation, or other host computer with secondary storage.

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. If you log commands, the way in which you build the command file depends on the configuration of the connection. This section shows how to build and use command files for three of the possible setups.

Commands to be logged can be classified into two categories: those that take an action and those that list status. Many commands fit into both categories. For example, the **tsq -i <number>** command deletes a trace sequence term. The **tsq** command lists all current trace sequencer settings.

You can use this action-status division to your advantage when logging commands. For example, if you want to log the configuration of the emulator to a command file, including the trace settings and so on, it's best to reassign the emulator's standard output to a file. Thus, the file will capture the lists output from the various commands. The lists can be used directly for the command file. If you want to log several action commands, it's usually best to log only the command inputs and reassign the standard output to another port so that the output isn't captured (trace and memory lists, for example).

Editing Command Files

Because the command file is an ASCII text file, you may use an ASCII editor to add, modify, or remove commands.

Comments in Command 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. You can put comments in command files by using a text editor or by entering the comment as a "command" in the interface command line entry area when logging commands to a file. The same mechanism that allows you to enter comments directly into the command line when logging commands also prevents the interface from trying to execute the comment as an interface command. See "To create a command file with a text editor" for more information.

To create a command file with a text editor

- Type in a series of commands in a text editor (one to a line or multiple commands on one line, separated by semicolons) and save the file to disk.

You can create and edit command files with any text editor that will write and edit ASCII files. To insert a comment in a command file, precede the comment text with the **#** character. Anything after that character is ignored by the Terminal Interface command interpreter.

Example

Create the following text file using an editor and save it as **comfile**:

```
demo                          # loads demo program
r                             # runs program
# Now give the program a command
m CMD_INPUT=41                # command "A"
m -db MSG_DEST..MSG_DEST+20   # display output buffer
# Set up a trace and execute it
tg addr=int_serv:FILL_DEST    # trigger on blank fill routine
t                             # start a trace
m -db CMD_INPUT=42            # command "B"
tl -e                         # list trace with symbols and addresses
b                             # break
rst                           # reset the emulation processor
```

To log a command file from a PC host

1 Start the terminal emulation software on your PC (such as HP AdvanceLink).

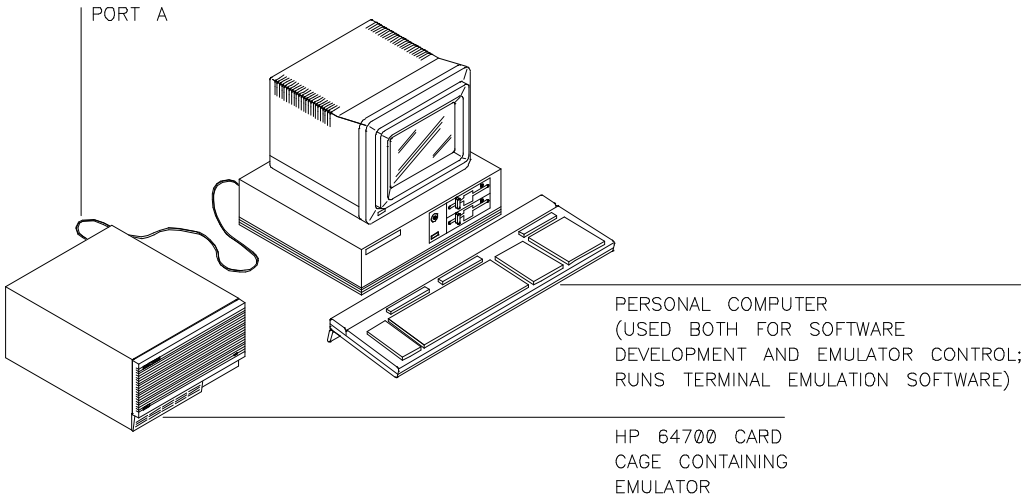
2 Enable the file logging capability of your terminal emulation software.

3 Type the series of Terminal Interface commands that you want to save to the command file.

4 Disable the file logging capability of the terminal emulation software.

5 Edit the disk log file as needed to remove extraneous information such as command prompts and command responses. Save the file under the name that you want to use for the command file.

The above procedure allows you to save a series of Terminal Interface commands in a file that you can later edit into a form suitable for use as a command file.



To log a command file on a UNIX host (emulator on different port)

1 Connect to the emulator using the UNIX **cu** command.

2 Begin logging commands to a file named **<filename>** on the host by typing: **~%><filename>**

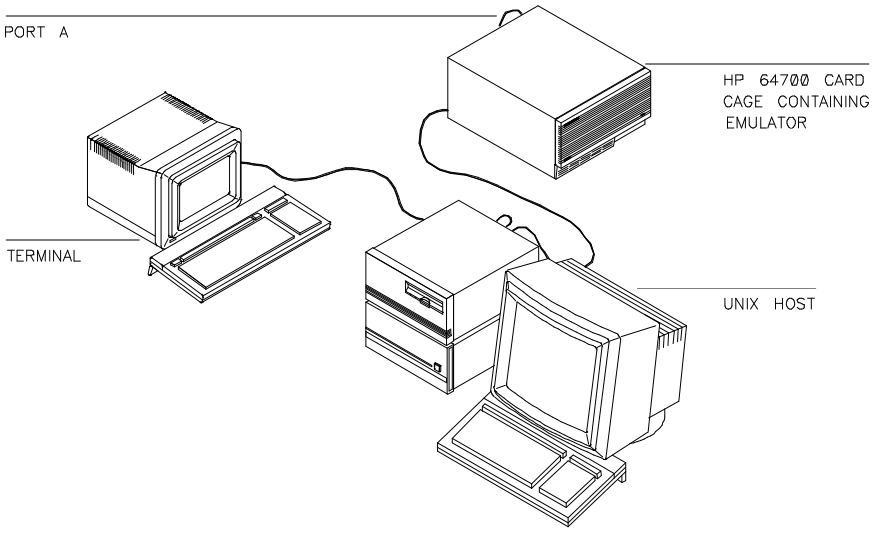
3 Type in the series of commands that you want to save in the command file.

4 End command logging by typing: **~%>**

5 Exit **cu** by typing: **~.**

6 Edit the file you saved to remove command prompts, add comments or change commands.

You can use the file redirection capability of **cu** to log commands to a file on the host.



When you type the ~ character during **cu** program execution, **cu** prints the host name of your system after the tilde.

Example

Connect to the emulator using **cu**:

```
$ cu -l /dev/tty01
```

Redirect the command inputs to a command file:

```
R>~%>cfile
```

Type in the commands that you want to save in the command file. Now end input redirection:

```
R>~%>
```

Edit the command file **cfile** to remove command prompts and other unwanted information.

To use a command file on a PC host

- Use the ASCII upload feature of your terminal emulation software to send the command file to the HP 64700 Series Card Cage.

  By using an ASCII or text upload feature built into your terminal emulation program, you can send a command file on your PC's disk to the emulator. You must use an upload feature because it will ship the file out to the serial connection. Many "disk read" or "file read" functions simply display the file's contents on the PC display without sending the data to the serial port.

To use a command file on a UNIX host (emulator on different port)

1 Connect to the emulator using the UNIX **cu** command.

2 Download the command file named **<filename>** to the emulator by typing: **~%<filename**

You can use the input redirection capability of **cu** to send a file on the UNIX host to the emulator.

Example

Connect to the emulator:

```
$ cu -l /dev/tty01
```

Initialize the emulator:

```
R>init -c
```

Download the command file named **cfile**:

```
R>~%<cfile
```

(Note: the name of your host computer will usually be printed by **cu** after you type the tilde (~).)

4

Using the Emulator

How to use the Terminal Interface to control the processor and view system resources

The emulator has many commands and features that allow you to control execution of your program. It also has facilities for entering and recalling commands.

To configure the emulator

- Set up the emulator for use by configuring it as described in Chapter 7, "Configuring the Emulator."

The emulator has several configuration items that adapt it to specific system designs and program requirements. You should check the configuration and modify it for your needs before using the emulator. This will ensure correct operation of all emulator functions.

To build programs

1 Create source files in "C" or MC68020/MC68030/MC68EC030 assembly language using a text editor.

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

Translate the assembly source files to relocatable object files using a compatible MC68020/MC68030/MC68EC030 cross assembler.

3 Link all relocatable object files with the linker/loader to produce an absolute object file in HP64000 (HP-OMF) format or Motorola S-record file format.

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 emulator Terminal Interface. Usually, this means that you'll want your files in Motorola S-record or HP64000 (HP-OMF) absolute format. The HP language tools for the HP 9000 can produce these formats.

Processor	C Compiler	Assembler
68020/68EC020	HP 64903	HP 64870
68030/68EC030	HP 64907	HP 64874

You may use other language tools, such as the Microtec Research™ or Intermetrics™ compilers and assemblers, if they produce the Motorola S-record format or HP64000 absolute file format. (These are the preferred formats, but the Terminal Interface will also accept Intel hex and Tektronix hex file formats.)

Example

To build the demo program with the HP 64870 Assembler/Linker:

```
as68k -h demo
as68k -h handle_msg
ld68k -h -c demo.k -o demo.x
```

To build the demo program with the Microtec Research assembler/linker on a PC:

```
asm68k -fd -fh -fcase -ft -L demo.s
asm68k -fd -fh -fcase -ft -L handler.s
lnk68k -c demo.k -fi -fh -fd -fs -L -o demo.x
```

To load the demo program

- Load the emulator demo program by typing: **demo**

The emulator contains a simple demo program that acts as a primitive command interpreter. You can use this program to learn more about the MC68020/MC68030/MC68EC030 emulator.

The demo command maps memory and sets the configuration items as needed. It also initializes the trace vector in the vector table so that the **s** (step) command will work properly.

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

More information on the demo program is in Chapter 1, “Quick Start.”

Loading Programs

The Terminal Interface provides commands that allow you to move files into emulation or target memory from a host computer through the serial or LAN ports of the HP 64700 Card Cage.

Many different absolute file formats are supported. The primary one discussed in this section is Motorola S-record format. If you have a host computer connected to the same LAN as the HP 64700, you can move files in the HP64000 absolute format.

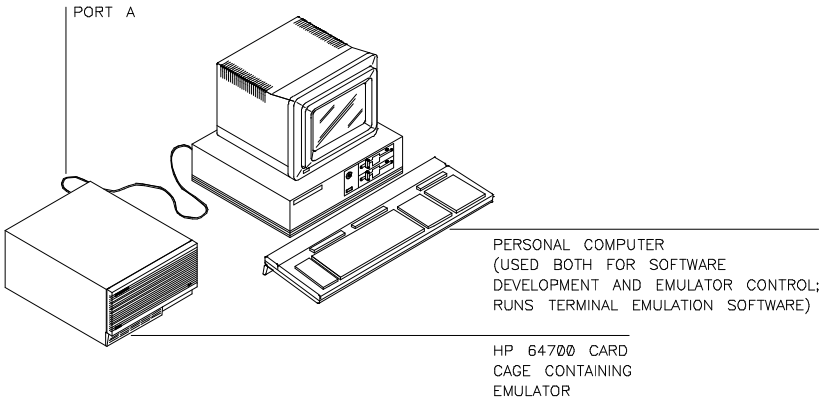
The **load** command has other options that allow you to control the load process. See **load** in Chapter 10, "Emulator Commands," for details.

To load a program from a PC host (PC controls emulator)

1 Build an absolute file in the Motorola S-record format (see "To build programs" in this chapter).

2 Start the terminal emulation software on the PC (such as HP AdvanceLink).

3 At the Terminal Interface prompt, type: **load -m**



4 Exit the terminal emulation software.

5 At the MS-DOS prompt, type: **copy <filename> <com_port>**

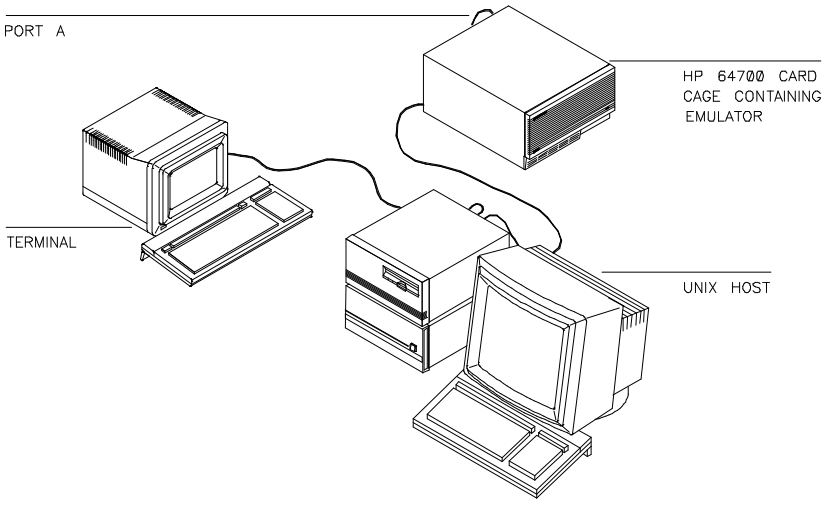
where **<filename>** is the name of the Motorola S-record file you want to load, and **<com_port>** is the name of the PC communications port (COM1..COM4) to which the emulator is connected.

6 Restart the terminal emulation software.

To load a program from a UNIX host (emulator on different port)

1 Build an absolute file in the Motorola S-record format (see “To build programs” in this chapter).

2 To connect to the emulator, type the command **cu -l /dev/ttyXX** at the HP-UX prompt .



where **XX** is the device number of the UNIX system serial port connected to the HPC emulator.

3 Type: **load -im**

4 Type: **~%< <filename>**

where **<filename>** is the name of the Motorola S-record file that you want to load.

To load programs over the LAN

- Use the **ftp** command on your local host computer to transfer files to the remote HP 64700.

  When connecting to the HP 64700's ftp interface, you can use either the HP 64700's hostname or the Internet Protocol (IP) address (or internet address). When you use the HP 64700's hostname, the ftp software on your computer will look up the internet address in the hosts table, or perhaps a name server will return the internet address.

Examples

To connect to the emulator's ftp interface, enter the following command (use any name and password):

```
$ ftp 15.35.226.210
Connected to 15.35.226.210.
220 User connected to HP64700
Name (15.35.226.210:guest):
Password (15.35.226.210:guest):
230-
                              NOTICE

   This utility program is unsupported. It is provided at no cost.
   Hewlett-Packard makes no warranty on its quality or fitness for
   a particular purpose.

 FTP on the HP64700 serves as a means for downloading absolute files to the
 emulation environment. The file transfer can be be performed as follows:

 1. The data mode type must be set to IMAGE (binary)

 2. Store the file using options to indicate the file format.  The following
 example uses PUT as the host command for sending the file. This may be
 different for your ftp implementation.
```

```
 put <file_name> <options>
     <file_name> - host file to be loaded.
     <options>   - The options are preceeded by a minus (-). The available
     options vary for individual emulators. All support HP OLS, Intel hex,
     Motorola S-records, and Extended Tek Hex.  Emulator specific options can
     be viewed by issuing a Terminal Mode help for the load command.

     put hpfile.X  -h  #to download an HP OLS file
     put intelfile -i  #to download an Intel Hex file
     put motfile   -m  #to download a Motorola S-record file
     put tekfile   -t  #to download an Extended Tek Hex file

230
```

To set up ftp for binary file transfers:

```
ftp> binary
200 Type set to I
```

To download the HP 64000 format absolute file into the emulator:

```
ftp> put program.X -h
200 Port      ok
150
226-
R>
226 Transfer completed
3332 bytes sent in 0.20 seconds (16.27 Kbytes/sec)
```

To exit out of the ftp interface:

```
ftp> quit
221 Goodbye
$
```

Symbols

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

The symbols you enter and the corresponding address information is stored in an emulator system table. When you display memory in mnemonic form, step the processor, or display trace results, the emulator retrieves the symbol information from the table and displays it. This makes the measurement results easier to read.

In the Terminal Interface, you can only define global and local symbols by downloading a symbol file from a host computer (see “To load program symbols”). Otherwise, you can define your own (user) symbols by adding them within the Terminal Interface (see “To add user symbols”).

To load program symbols over the LAN

- Use the **ftp** command on your local host computer to transfer files to the remote HP 64700.

  Loading symbol files over the LAN is the same as loading absolute files over the LAN, except that a different option is used with the "put" command in ftp.

  Symbol files are ASCII files in a special format. Chapter 13, “Data File Formats,” describes the symbol file format.

Examples

To connect to the emulator's ftp interface, enter the following command (use any name and password):

```
$ ftp 15.35.226.210
Connected to 15.35.226.210.
220 User connected to HP64700
Name (15.35.226.210:guest):
Password (15.35.226.210:guest):
230-
                                NOTICE

   This utility program is unsupported. It is provided at no cost.
   Hewlett-Packard makes no warranty on its quality or fitness for
   a particular purpose.

.
.
.
```

To set up ftp for binary file transfers:

```
ftp> binary
200 Type set to I
```

To download the symbol file into the emulator:

```
ftp> put program.sym -S
200 Port      ok
150
226-
R>
226 Transfer completed
1789 bytes sent in 4.78 seconds (0.37 Kbytes/sec)
```

To exit out of the ftp interface:

```
ftp> quit
221 Goodbye
$
```

To add user symbols

- Add a user symbol by typing: **sym <name>=<address>**

You can define user symbols to help you while you're making measurements. For example, you might find that you're repeatedly entering a particular address for memory display commands. If you define this address as a symbol, you can use the symbol in the memory display command. Also, the symbol will be displayed in analyzer measurements and memory mnemonic displays.

To remove symbols

- To delete all symbols in the emulator's symbol table, type: **sym -d**
- To delete all user symbols, type: **sym -du**
- To delete a specific user symbol, type: **sym -du <symbolname>**
- To delete all global symbols, type: **sym -dg**
- To delete all local symbols for all modules, type: **sym -dl**
- To delete all local symbols for a specific module, type: **sym -dl <modname>:**

The emulator symbol table uses system memory so you might need to delete symbols or sets of symbols to free memory while you're using the emulator. You also might want to delete symbol sets that you're no longer using so they don't clutter the symbol display.

Examples

To delete the symbols for the handle_msg module in the demo program, enter:

```
R>sym -dl handle_msg:
```

To delete all global symbols, enter:

```
R>sym -dg
```

To display symbols

- To display all symbols in the emulator's symbol table, type: **sym**
- To display all user symbols, type: **sym - u**
- To display a specific user symbol, type: **sym -u <symbolname>**
- To display all global symbols, type: **sym -g**
- To display a specific global symbol, type: **sym -g :<symbolname>**
- To display all local symbols for all modules, type: **sym -l**
- To display all local symbols for a specific module, type: **sym -l <modname>:**
- To display a specific local symbol, type: **sym -l <modname>:<symbolname>**

Examples

To display the local symbols for the module demo in the demo program, enter:

```
R>sym -l demo:
```

To display the value of all global symbols, enter:

```
R>sym -g
```

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.

To display memory

- To display a range of memory in the format set by the mode command, type: **m <address_range>**
- To display a range of memory in byte format, type: **m -db <address_range>**
- To display a range of memory in word format, type: **m -dw <address_range>**
- To display a range of memory in long word format, type: **m -dl <address_range>**
- To display memory in MC68020/MC68030/MC68EC030 mnemonic format, type: **m -dm <address_range>**

  The display mode is initialized by the **mo** (mode) command. You can change the display mode setting using the options shown above.

  When you use the **-dm** 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.

  Only emulation memory mapped with the **dp** (dual-port) attribute may be displayed or modified while a user program is running. Dual-port memory may also be displayed while the emulator is reset. For other memory, the emulator must use the monitor to access it. Use the **b** command to begin running in the monitor.

Examples

Before using the following examples, reload the demo program:

```
R>demo
```

To display program memory for the main part of the demo program in the current display mode, enter:

```
R>m demo:Main..EndLoop
```

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

```
R>m -dl 0..3ff
```

To display the demo program's message output buffer in byte format, enter:

```
R>m -db handle_msg:Msg_Dest..Msg_Dest+1f
```

To display the Print_Msg routine for the demo program in mnemonic format, enter:

```
R>m -dm handle_msg:Print_Msg..Fill_Dest+0a
```

To modify memory

- Modify a single memory location to a single value by typing: **m <address>=<value>**
- Modify a range of memory locations to a single value by typing: **m <lower>..<upper>=<value>**
- Modify a range of memory locations with a list of values by typing: **m <lower>..<upper>=<value1>,<value2>, . . .**
- Change whether **<value>** is interpreted as a byte, word, or long word data type by adding the **-d<mode>** parameter before the address range.

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 accessed. <**value**> represents the data value to which the contents of memory are to be modified.

If you don't use the **-d**<**mode**> parameter, the current display mode is used to interpret the data type of <**value**>. Otherwise, the display mode you specify is used to interpret the data type. See the examples and "To Initialize Display and Set Access Modes" for more information.

Examples

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

```
R>m 0e1f=43
```

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

```
R>m -db 0e1f=43
```

To modify the command input buffer of the demo program to 43 hex, enter:

```
R>m -db demo:Cmd_Input=43
```

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

```
R>m 0e00..0e38=0
```

To modify the range of locations from 0e00 through 0e38 to "ABC", enter:

```
R>m -db 0e00..0e38=41,42,43
```

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:

```
R>m -db 0f01=03
```

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

```
R>m -dw 0f00=03
```

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 MC68020/MC68030/EC030 processor architecture).

To search memory

- To search a memory range for a particular expression, type: **ser <lower>..<upper>=<expr>**
- To search a memory range for a character string, type: **ser <lower>..<upper>=<string>**
- To change the mode that determines matching characteristics for the search, add the **-d<mode>** parameter before the address range.

Searching memory for values or character strings can help you determine whether a program is functioning correctly. For example, in the emulator demo program, you can enter a command, then search memory for the output message to see if the program responded 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 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.

If you don't use the **-d<mode>** parameter, the current display mode is used. Otherwise, the display mode you specify is used to determine how data is matched during the search. See “To initialize display and set access modes” in this chapter for more information.

Examples

Suppose that memory location f00 contains 03 and f01 contains 00 hex. Then the word spanning both locations contains 0300 hex.

To search these locations for 3 hex by words, enter:

```
R>ser -dw 0f00..0f01=3
```

The search will fail since the value 3 hex doesn't lie on a word boundary. To search for the same value by bytes, enter:

```
R>ser -db 0f00..0f01=3
```

The match is found at address 0f00.

To search the message area of the emulator demo program for the string "ommand," enter:

```
R>ser -db handle_msg:Msg_A..End_Msgs="ommand"
```

Three matches are found.

To copy memory blocks

- Copy a memory block from an address range specified by **<lower>** and **<upper>** to the destination address range having lower bound **<destination>** by typing: **cp <destination>=<lower>..<upper>**

The **cp** command allows you to move blocks of code or data to different locations in 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

Suppose that you need to modify the exception vector table located in your target system ROM. The following are the initial conditions for the memory map:

To copy memory blocks

```
R>map

map 0000..0ffffh # 64 Kbytes target ROM; vector table/program code
map 10000..18fff # 32 Kbytes target RAM;program data
map 19000..19fff # Other guarded memory
map 80000..80fff # Emulation RAM (foreground monitor)
```

To modify the vector table, first create a new emulation memory term:

```
map 20000..203ff # Emulation RAM (1K block for
exception vector table)
```

Copy the exception table from target ROM to emulation RAM:

```
R>b
R>cp 20000=0..3ff
```

Now you can modify the table. To have the processor use the new table in emulation RAM, enter:

```
R>r vbr=20000
```

To initialize display and set access modes

- Initialize the global display mode by typing: **mo -d<disp_mode>**

  where **<disp_mode>** is **b** for byte, **w** for word, **l** for long word, or **m** for mnemonic.

- Set the global access mode by typing: **mo -a<access_mode>**

  where **<access_mode>** is **b** for byte, **w** for word, or **l** for long word.

- Check the mode settings by typing: **mo**

  The display mode setting affects your interaction with memory displays, modifications, and searches. The display mode determines whether the emulator interprets data values as bytes, words, or long words. You can use the mode command to set the mode that you need initially. If you use the mode parameters to the individual commands, the global display mode is changed.

  The access mode has a different function. When you display or modify target system memory or emulation memory that is not dual-port, the emulator uses the monitor to read or write target memory locations. The access mode determines whether the emulator uses byte, word, or longword sizing for the memory accesses.

Using Processor Run Controls

When you don't use an emulator, run control can be difficult. Usually, you're limited to starting the processor from reset, and then entering data values that vector program execution to the routines you want to test. Reaching those routines may be difficult or impossible if the data values are boundary conditions or 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 **s** command to step through the program. You can break to the monitor program to examine on-chip resources such as registers. You can also reset the processor from the emulator.

To run a program

- To run a program from the current program counter (PC) value, type: **r** or
- Type: **r $**
- To run a program from a specific address, type: **r <address>**
- To run a program from target system reset, type: **r rst**

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

<address> is a 32-bit address expression. You can include function codes to specify the memory space to which the address applies. See <ADDRESS> in Chapter 11, "Expressions," for more information.

The **r rst** command pulses the processor reset line. The processor fetches the values at addresses 0 and 4 and loads these values into the interrupt stack pointer and program counter registers. It then begins running from the program counter address value.

If the emulator is in the reset state (R> prompt), the **r** command (with no parameters) acts the same as **r rst**. Otherwise, **r** runs from the current program counter value.

However, if you reset the emulator, break to the monitor, and then run the emulator, the stack pointer and program counter values will not be initialized. Therefore, the run will fail. The **cf rv** configuration item allows you to define initial values for the program counter and stack pointer in this instance. See Chapter 7, "Configuring the Emulator," for more information. Typically, you will want to put the values found at addresses 0 and 4.

Examples

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

```
R>r demo:Main
```

or

```
R>r 400
```

To break to monitor

- Break the emulation processor into the monitor by typing: **b**

The emulation monitor is a program that provides various emulation functions, including register access and target system memory manipulation. If the emulator is reset, it will enter the monitor before executing certain emulation commands, such as those accessing registers, emulation memory that is not dual-port, or target system memory. (The emulator breaks to the monitor temporarily if you enter these commands during user program execution, unless you restrict the emulator to real-time runs. See Chapter 7, "Configuring the Emulator," for more information.) You also can use the break command to pause execution of your user program.

The prompt changes to M> to show that the processor is running in the monitor.

You can use either a foreground or background monitor. See Chapter 7, "Configuring the Emulator," for more information.

To step the processor

- To step the processor one instruction from the current program counter value, type: **s**

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

- To step the processor one instruction from an address given by <address>, type: **s 1 <address>**

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

- To inhibit display of information about the steps, add the **-q** parameter before the **<count>** and **<address>**.

- To display only the next program counter value when the step is complete, add the **-w** parameter before the **<count>** and **<address>**.

  The **s** (step) 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. 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. You must supply a step count if you supply an address. Otherwise, the emulator will interpret the address as a step count.

  The default base for **<address>** is hex. If you omit the address, the current program counter value is used. You can use **$** to mean the same thing as the current program counter value.

  Both emulation monitors use the trace exception vector (located at offset 24 in the vector table) to implement the **s** (step) command. Therefore, you must initialize this vector properly before using the step command.

  - If you're using the background monitor, the emulator may try to modify the trace vector as needed to complete the step function. If you have trouble with

this (for example, the vector table is in target ROM and the trace vector is uninitialized) you might want to set the trace exception vector to an even value that points to a mapped memory area. Or, you can set the vector to point at the start of the vector table (usually address 0 unless you have relocated the vector table by modifying the vector base register (VBR).

- If you're using the foreground monitor, the trace exception vector must point to the TRACE_ENTRY address in the foreground monitor. In the default foreground monitor, this address will equal the setting of **cf monaddr** plus 800h. For example, if **cf monaddr=1000**, then the trace exception vector should be set to 00001800h.

The emulator verifies the trace vector each time that you enter a **s** command. During execution of the command, it does not recheck the value for each instruction step. So, if you relocate the vector table by modifying the VBR, the emulator will not recheck the trace vector in the new location. If you modify the trace vector value after this relocation (through program execution), stepping may fail.

The memory area used by the vector table must point to readable memory (that which is not guarded memory). Usually, the vector will be in emulation RAM or target RAM. The emulator modifies the trace vector to do the step, but restores the original value when the step completes. If it is in target ROM, the emulator cannot modify the trace vector properly. Therefore, the ROM must contain the correct trace vector value for stepping to work correctly.

Examples

To step the processor one instruction, enter:

```
M>s
```

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

```
M>s 3
```

To step the processor five instructions from the demo:Loop symbol in the sample program, enter:

```
M>s 5 demo:Loop
```

To step once and disable step display, enter:

```
M>s -q
```

To step twice from the start address of the demo program and display only the resulting program counter value, enter:

```
M>s -w 2 400
```

To reset the processor

- To reset the emulation processor from the emulator, type: **rst**
- To reset the emulation processor, and then begin running in the emulation monitor, type: **rst -m**
- To reset the emulator from the target system, assert the RESET signal in your target system.

  When you apply power to the emulator, the initialization process leaves the emulator in the reset state. Changing some configuration items also resets the processor. (See Chapter 7, “Configuring the Emulator,” for more information.)

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

  Both the MC68020 and MC68030/EC030 emulators 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 emulators are running a user program when the target system reset occurs, they behave as if a **r rst** command were issued.

  If the MC68020 emulator is in the background monitor when the target reset occurs, it will reenter the monitor when the reset is released. (As if a **rst -m** command were given.) The MC68030/MC68EC030 emulator behaves this way for both the foreground and background monitors.

Viewing and Modifying Processor Registers

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

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

To display registers

- To display an individual register, type **reg** and the register name.
- To display all registers in a class, type: **reg <reg_class_name>**
- where **<reg_class_name>** is the name of a class of registers. The available registers and register classes are shown in the following table:

Register Class	Register Names
*** (basic)**	**pc, st, usp, isp, msp, cacr, caar,d0..d7, a0..a7, vbr, dfc, sfc**
fpu	**fpcr, fpsr, fpiar, fp0..fp7**
mmu (MC68030) **OR** **acu (MC68EC030)**	**tt0, tt1, mmusr, tc, srp, crp** **ac0, ac1, acusr**

The processor must be running to allow register displays. If it's running in the monitor, the emulator does the display directly. If it's running the target system program, the emulator forces a break to the monitor, gets the register data, and then returns to the user program. (If you restrict the emulator to real-time runs, the **reg** command isn't allowed while you're running a user program. See Chapter 7, "Configuring the Emulator," for more information.)

You can combine displays of multiple registers and register classes by listing all the arguments on the same command line.

Your target system must have an active floating-point coprocessor in order to display the fpu register class (floating-point registers in the coprocessor). If your target system does not have a floating-point coprocessor, attempting to access an fpu register will cause an error message to appear.

The mmu register class of the M68030 is only accessible when the mmu is enabled (**cf mmu=en**). The acu register class is only accessible when the mmu is disabled (**cf mmu=dis**). Registers ac0, ac1, and acusr are nearly identical to tt0, tt1, and mmusr.

Examples

To display the processor's A0 register, enter:

```
M>reg a0
```

To display the D5 and USP registers, enter:

```
M>reg d5 usp
```

To display the PC and the CCR register, enter:

```
M>reg pc st
```

(The CCR register is part of the status register st).

To modify registers

- To modify a register to a new value, type: **reg <regname>=<value>**

where **<regname>** is the name of a processor register, and **<value>** is an expression matching the data type of the register (byte, word, or longword). (You can't use symbols in the expression.)

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 "To Display Registers" above for more information.

You can modify several registers on the same command line. You can also display and modify registers on the same command line.

To modify an MMU register in the MC68030 emulation processor, the MMU must be enabled (**cf mmu=en**).

To modify registers in the fpu class, your target system must have a floating-point coprocessor. You can enter values into the three fpu control registers, and into the eight floating-point registers, in the hexadecimal number base.

Examples

To modify the PC register to the Loop address of the demo program, enter:

```
M>reg pc=040e
```

Notice that you can't use a symbol in the expression when modifying a register.

To modify the D3 register to 0 and the A6 register to 80a5, enter:

```
M>reg d3=0 a6=80a5
```

To modify the A4 register to 4a and display the CACR and CAAR registers, enter:

```
M>reg a4=4a cacr caar
```

To modify the tt0 register of the MC68030 to 00000110, enable the mmu (**cf mmu=en**), and enter:

```
M>reg tt0=110
```

To modify the ac1 register of the MC68EC030 to 00000000, disable the mmu (**cf mmu=dis**), and enter:

```
M>reg ac1=0
```

Using Software Breakpoints

Software breakpoints give you a way to stop program execution when the processor reaches a certain instruction. Suppose that you find a problem with execution of a certain instruction. You can set a breakpoint at that location. When the processor reaches that location, the emulator will force a break to the monitor. You can then display registers or memory to determine the cause of the problem.

You can insert, enable, disable or remove software breakpoints. When you insert and enable a breakpoint, the emulator replaces the instruction at the target location with a BKPT instruction. The BKPT acknowledge cycle is read by the emulator, and the emulator causes a monitor break. The emulator replaces the BKPT instruction with the original instruction when the breakpoint is hit, or when you disable or remove the breakpoint.

Before you use breakpoints, you must choose the breakpoint vector, and enable the breakpoints feature (with the **bc -e bp** command). You will normally do these things only once for every emulation session, no matter how many breakpoints you use. To do these things, see Chapter 7, “Configuring the Emulator.”

You should only set a software breakpoint at the first word of a program instruction. You can set up to 32 different breakpoints at a time.

You can perform a single operation on a set of breakpoints by specifying all the breakpoint addresses for that operation on the command line. See the examples below.

To insert a software breakpoint

- Insert a new software breakpoint at a location given by **<address>** by typing: **bp <address>**

When you set a software breakpoint, the emulator replaces the code at the location you specified with a BKPT instruction. It also makes an entry in the system breakpoint table. When the processor executes the breakpoint code, the breakpoint is disabled, and the original code at those locations is replaced. The software breakpoint table entry is marked as “disabled.”

The emulator uses the monitor to insert the breakpoint.

Examples

Enable the software breakpoints feature by entering:

```
M>bc -e bp
```

To insert a breakpoint at the symbol Cmd_A, enter:

```
M>bp handle_msg:Cmd_A
```

To insert breakpoints at Print_Msg and Fill_Dest, enter:

```
M>bp handle_msg:Print_Msg Fill_Dest
```

The handle_msg: local symbol module specification sets the default local symbol module. Therefore, you don't need to specify the local symbol module with Fill_Dest.

To insert a breakpoint at address 42a, enter:

```
M>bp 42a
```

To enable a software breakpoint

- To enable an existing software breakpoint at a location given by **<address>**, type: **bp -e <address>**
- To enable all existing software breakpoints, type: **bp -e ***

When a breakpoint is hit, it is disabled. If you want to reenable the breakpoint, you use the **-e** option to the **bp** command. The emulator will search the breakpoint table for the address you specify. If there is a breakpoint entry for that location, the entry is marked as "enabled" and the BKPT instruction is 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 **b** command to begin running in the monitor.

Examples

To enable an existing breakpoint at Cmd_A, enter:

```
M>bp -e handle_msg:Cmd_A
```

To enable existing breakpoints at Print_Msg and Fill_Dest, enter:

```
M>bp -e handle_msg:Print_Msg Fill_Dest
```

To enable an existing breakpoint at address 42a hex, enter:

```
M>bp -e 42a
```

To disable a software breakpoint

- To disable an existing software breakpoint at a location given by **<address>**, type: **bp -d <address>**
- To disable all existing software breakpoints, type: **bp -d ***

Sometimes you will want to temporarily disable a software breakpoint without removing it. The **-d** option to the **bp** command lets you do this.

When you disable a software breakpoint, the emulator replaces the BKPT instruction at the breakpoint location with the original instruction. It marks the software breakpoint table entry as "disabled." Then the processor won't break to 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 **b** command to begin running in the monitor.

Examples

To disable an existing breakpoint at Cmd_A, enter:

```
M>bp -d handle_msg:Cmd_A
```

To disable existing breakpoints at Print_Msg and Fill_Dest, enter:

```
M>bp -d handle_msg:Print_Msg Fill_Dest
```

To disable an existing breakpoint at address 42a, enter:

```
M>bp -d 42a
```

To remove a software breakpoint

- To remove an existing software breakpoint at a location given by <**address**>, type: **bp -r** <**address**>
- To remove all existing software breakpoints, type: **bp -r ***

When you're finished using a particular breakpoint, you should remove the breakpoint table entry. The **-r** option to the **bp** command lets you do this. The original instruction is restored to memory, and the breakpoint table entry is removed.

The emulator uses the monitor to remove the breakpoint. Therefore, you can't remove a breakpoint when the emulator is reset. Use the **b** command to begin running in the monitor.

Examples

To remove an existing breakpoint at Cmd_A, enter:

```
M>bp -r handle_msg:Cmd_A
```

To remove existing breakpoints at Print_Msg and Fill_Dest, enter:

```
M>bp -r handle_msg:Print_Msg Fill_Dest
```

To remove an existing breakpoint at address 42a hex, enter:

```
M>bp -r 42a
```

To display software breakpoints

- To display all existing software breakpoints, type: **bp**

Using the Emulator In-Circuit

Out-of-circuit emulation is useful for debugging your program code. You can use the emulation-bus analyzer and other emulator features to test and evaluate your code.

As the design of your target system progresses, you'll want to test features of your program that interact with your target system hardware instead of the emulation memory.

You connect the emulator probe to your target system to do in-circuit emulation. Then you can make analyzer measurements and have the memory display and other capabilities of the emulator to debug system problems.

To prepare the emulator for in-circuit emulation, take the following steps:

1 Study your system design, especially its memory configuration.

2 Study Chapter 7 of this manual to determine how to set configuration items for the best results with your system.

3 Install the emulation probe in your target system.

4 Set configuration items as determined by the results of step 2.

When you use the emulator in-circuit, you need to carefully consider the configuration of the emulator and its relationship to your system design. See Chapter 7, "Configuring the Emulator," for details.

To install the emulation probe

Caution

Possible damage to the emulator probe. The emulation probe contains devices that are susceptible to damage by static discharge. You should take precautions before handling the probe, to avoid damaging the internal components of the probe with static electricity.

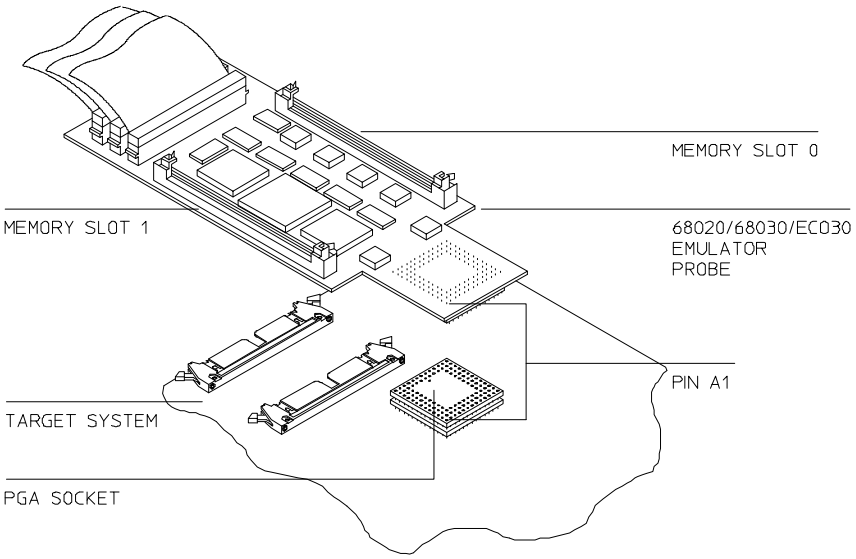
To install the emulation probe

Caution *Possible damage to the emulator*. Make sure both your target system and emulator power are OFF before installing the emulator probe into your 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 your target system socket. Note the location of pin A1 on the processor and on your target system socket. Store the processor in a protected environment (such as antistatic foam).

2 Insert the emulator probe into your target system socket. Make sure to align pin A1 of the emulator probe and your system socket.



To power-on the emulator and your system

Caution

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

1. Apply power to the emulator.
2. Apply power to your target system.

To probe other types of target system sockets

- Adapters for special target system probing needs are available, as shown in the following table.

Probe type	HP part number
68020 PGA to PGA extender	64748-61604
68020 PGA to PQFP extender	E2426-61601
68020 PGA to 68EC020 PGA adapter	64748-87602
68030/EC030 PGA to PGA extender	64747-61601
68030/EC030 PGA to PQFP surface mount adapter (low profile)	E2406-61602
68030/EC030 PGA 90° CCW rotator	64700-87620
68030/EC030 PGA 90° CW rotator	64700-87619

Using The MC68030 Emulator With MMU Enabled

When you enable memory management in the MC68030 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 MC68030 emulator with the MMU enabled. Chapter 9 provides detailed information to help you use the MC68030 MMU most efficiently.

To enable the processor memory management unit (MC68030 only)

- To turn on the MMU in the MC68030 emulation processor, enter: **cf mmu=en**

Once enabled, the MMU of the MC68030 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 of the MC68030 will manage memory. The MMU of the MC68030 must be enabled by this configuration question before the operating system can establish those control values.

The target system will control the MMU during program execution by using the /MMUDIS signal. If you disable the MMU with this configuration question, the /MMUDIS signal from the target system will be ignored.

A foreground monitor must be used when the MMU of the MC68030 is enabled. If the background monitor is selected when you type in the **cf mmu=en** command, the foreground monitor will be selected automatically.

Note

Make sure the foreground monitor is mapped to memory space that has a 1:1 translation. Refer to Chapter 7 for instructions on how to map the foreground monitor to 1:1 address space in the MC68030 MMU.

Examples

To enable the MC68030 MMU so that the operating system can set it up to manage memory, enter the command:

M>**cf mmu=en**

To disable the MC68030 MMU, enter the command:

M>**cf mmu=dis**

To see the present state of the MMU, enter the command:

M>**cf mmu**

To obtain additional information about the MMU, enter the command:

M>**help cf mmu**

To view the present logical-to-physical mappings

- Enter the command: **mmu**

The display will show the logical-to-physical address translations defined by the current MMU registers and translation tables.

Examples

To see all of the logical-to-physical mappings (one display line for each mapped page), enter the command:

U>mmu

This will display translations for all function codes.

To see all of the logical-to-physical mappings for logical addresses from 0 through 0ffff, enter the command:

U>mmu 0..0ffff

By default, the list of mappings you get when you include an address with the **mmu** command shows all mappings available through the supervisor program function code. The first command in this set of examples did not include an address; it showed all logical-to-physical mappings for all function codes.

To see the logical-to-physical mapping for the page that contains logical address 40F0, enter the command:

U>mmu 40f0

To see only the mappings for supervisor space in the address range from 0 through 0ffff, enter the command:

U>mmu 0..0ffff@s

To see only the mappings under the user program function code in the address range from 0 through 0ffff, enter the command:

U>mmu 0..0ffff@up

Note that the function codes are **ud**, **up**, **sd**, and **sp**.

To show all of the valid mappings in the mapping tables for selected values of the TC, SRP, and CRP registers, ignoring the present values of those registers, enter a command, such as:

U>mmu tc=81ff2000 srp=20800604bf9fffe7 crp=8000000200001020

To see translation details for a single logical address

- Enter the command: **mmu -t <address>**

Examples

To see how logical address 40F0 (in supervisor program space) is mapped through the translation tables to its corresponding physical address, enter the command:

U>mmu -t 40f0

To see how logical address 1000 under the user program function code is mapped through the translation tables, enter the command:

U>mmu -t 1000@up

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

- Enter the command: **mmu -t<table> <address>**

  Where **<table>** is the table level you want to see (either **a**, **b**, **c**, **d**, or **f**, if function codes are used), and **<address>** is the logical address that uses the table at the point to be shown.

  There may be several base addresses for Table A. 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 1250, enter the command:

M>mmu -ta 1250

5

Using the Analyzer

How to view 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. Powerful triggering and sequencing capability ensures that the analyzer captures only the information you need, so you don't spend time searching through detailed trace lists for the information that's of interest.

Making Basic Analyzer Measurements

You can use just a few analyzer commands to make most measurements, such as these:

- 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.

The analyzer has powerful triggering, storage and trace list display capability. These features are described in other sections of this chapter.

To create an 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, “Expressions,” for more details on numeric expressions and the available logic operators.

Example

The following are valid numeric expressions:

```
1XXX0Y<<3
(340q*7)/2
0ffa^32T
52T*7a
```

To start a trace measurement

- Begin an emulation-bus analyzer trace by typing: **t**

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

To stop a trace measurement

- Halt an emulation-bus analyzer measurement by typing: **th**

Sometimes you need to halt a trace because an examination of the analyzer status shows that the trace isn’t capturing the data you expect. Then, you’ll want to halt the analyzer and reconfigure your trigger and storage terms to capture data.

To display the trace status

- Display the emulation-bus analyzer trace status by typing: **ts**

  The trace status display shows whether the trace is running or complete. It also shows the current sequencer state (whether triggered or still looking for the next sequence term) and shows the number of states captured. You will usually use this command if you can't display the trace because no data has been captured. The trace status will help you find the problem.

To display the trace list

- Display the trace list using the default parameters by typing: **tl**

  The trace list buffer is 512 or 1024 states deep (depending on whether or not you turn on the state/time count). You can selectively display portions of the buffer using the **tl** command.

To define a simple trigger qualifier

- Define a simple trigger on an address value, by typing: **tg addr=<value>**

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

Example

To trigger the analyzer when the demo program reaches the location Fill_Dest, enter:

```
M>tg addr=handler:Fill_Dest
```

To define a simple storage qualifier

- Store only the bus cycles that reference a particular address by typing: **tsto addr=<value>**

If you want to store only the accesses to a certain location, you can use the trace storage qualifier. More complex patterns of data and status qualification can be made, as well as range specifications.

Example

Measure the amount of time between writes of message characters to the Msg_Dest area:

```
M>init -c
M>demo
M>tg addr=handle_msg:Msg_Dest and stat=write
M>tsto addr=handle_msg:Msg_Dest..Msg_Dest+1f
M>t
M>r
M>m -db demo:Cmd_Input=41
M>tl
```

To set the trigger position

- To position the trigger term at the start of the trace list, type: **tp s**
- To position the trigger term at the end of the trace list, type: **tp e**
- To position the trigger term at the center of the trace list, type: **tp c**
- To position the trigger in the trace list with N number of states before it, type: **tp -b N**
- To position the trigger in the trace list with N number of states after it, type: **tp -a N**

The trigger position can help make the trace list more readable. For example, you might want to see all the program events leading to a particular access. You can define that access as the trigger term, and then position the trigger at the end of the trace (**tp e**).

The way the analyzer processes a trigger-position specification is by adding a count to trigger recognition. For example, if you specify **tp e**, the analyzer sets up to capture trigger plus zero states. If you specify **tp s**, the analyzer sets up to capture trigger plus 511 or 1023 states (depending on whether or not you turn on state/time count). When trigger plus count is captured, "trace complete" is shown, state capture stops, and you can view the content of trace memory.

Example

To position the trigger 10 states after the beginning of the trace, enter:

```
M>tp -b 10
```

Displaying the Trace List

The Terminal Interface allows you to present the analysis trace buffer in the manner most useful to you. You can rearrange the display columns or change their width. Also, you can create custom columns that represent certain groups of analyzer signal lines.

You can add various options to the **tl** (trace list) command to show specific state ranges or to specify disassembly modes.

To define analyzer labels

- To define a new analyzer trace label given by **<name>**, type: **tlb <name> <lower>..<upper>**

  where **<lower>** and **<upper>** represent the lower and upper boundaries of the group of analyzer signals that are to be included in the label definition.

- To define an analyzer trace label with negative polarity, type: **tlb -n <name> <lower>..<upper>**

You can define analyzer signal labels to focus on signals of interest. For example, you might want to define a label for a special subset of the data or address lines.

Example

Suppose that you want to see the individual bytes of the data bus.

```
M>tlb byte0 32..39
M>tlb byte1 40..47
M>tlb byte2 48..56
M>tlb byte3 57..63
```

You can use the **tf** command to add those columns to the trace list. See “To change the trace format.”

To delete analyzer labels

- Delete the analyzer label given by **<name>** by typing: **tlb -d <name>**

  If a label is in use (in the trace specification or trace format), it won't be removed until the specification or format is deleted or redefined without the label.

Example

Delete the analyzer's **byte 0** label (defined in the previous section):

```
M>tlb -d byte0
```

To display the analyzer labels

- To display the definition of a label given by **<name>**, type: **tlb <name>**
- To display the definition of all labels, type: **tlb**

Examples

Display the default analyzer labels:

```
U>tlb
```

You will see:

```
#### Emulation trace labels
tlb addr 0..31
tlb data 32..63
tlb stat 64..79
```

To change the trace format

- To display the analyzer input lines designated by **<LABEL>**, use the command: **tf <LABEL>, <BASE> [<WIDTH>]**

  where **<BASE>** specifies the radix for display (see “To Create an Expression”).

  <WIDTH> is an optional parameter that is valid only for the addr field. It specifies the width in characters (in the range 4..50) for the field.

- To display disassembled processor instruction mnemonics, add **mne** to the **tf** command line.

- To display count information (state or time) in relative format, add the **count,r** option to the command line; or display the count in absolute format by adding the **count,a** format to the command line. The count you can make is affected by the analyzer clock rate. See "To configure the analyzer clock" in this chapter.

- To display sequencer state change information, add the **seq** option to the **tf** command line.

- To display the current trace format, type: **tf**

  The **tf** command options specify how data is arranged on the screen when you display the trace list with the **tl** command. You can specify multiple options on the command line. The sequence of the options on the command line determines the sequence of the columns in the trace list display.

  When you enter a **tf** command with a new set of options, the previous trace format is destroyed and the options for the new command set the format.

Examples

The default trace format is the same as that obtained by entering the command:

```
R>tf addr,h mne count,R seq
```

View the resulting trace format:

```
R>init -c
R>demo
R>t
R>r
U>tl 0..5
```

You will see:

```
 Line   addr,H    68020 Mnemonic                              count,R  seq
-----   --------  ----------------------------------------  ---------  ---
    0   00000000   $0000         supr prgm long rd (ds16)         ---   +
    1   00000002   $1000         supr prgm word rd (ds16)    0.160 uS   .
    2   00000004   $0000         supr prgm long rd (ds16)    0.160 uS   .
    3   00000006   $0400         supr prgm word rd (ds16)    0.280 uS   .
    4   00000400   $2E7C         supr prgm long rd (ds16)    0.120 uS   .
    5   00000402   $0000         supr prgm word rd (ds16)    0.200 uS   .
```

To display the addresses in decimal, data in binary, count absolute, and omit the sequencer information, enter the command:

```
U>tf addr,t data,y count,a
U>tl 0..5
```

```
 Line  addr,T  data,Y                            count,A
-----  ------  --------------------------------  ---------
    0  00      00000000000000000000000000000000        0
    1  02      00010000000000000000000000000000   0.160 uS
    2  04      00000000000000000000000000000000   0.320 uS
    3  06      00000100000000000000000000000000   0.600 uS
    4  24      00101110011111000000000000000000   0.720 uS
    5  26      00000000000000000000000000000000   0.920 uS
```

You can also change the column order. For example, enter the commands:

```
U>tf count,a addr,t data,y
U>tl 0..5
```

```
 Line    count,A  addr,T  data,Y
-----  ---------  ------  --------------------------------
    0        0     00      00000000000000000000000000000000
    1   0.160 uS  02      00010000000000000000000000000000
    2   0.320 uS  04      00000000000000000000000000000000
    3   0.600 uS  06      00000100000000000000000000000000
    4   0.720 uS  24      00101110011111000000000000000000
    5   0.920 uS  26      00000000000000000000000000000000
```

To display the trace list

- To display the trace from the top of the list, type: **tl -t [<COUNT>]**

  where **<COUNT>** is an optional parameter specifying the number of states to be displayed. The default is to the last **<COUNT>** value.

- To display the next group of states from the trace (those previously undisplayed), type: **tl -n [<COUNT>]**

- To display the trace list beginning with the state numbered **<LOWER>**, type: **tl <LOWER>**

- To display the trace list states beginning with the state numbered **<LOWER>** and ending with the state numbered **<UPPER>**, type: **tl <LOWER>..<UPPER>**

- To display the complete trace buffer, type: **tl ***

  The **tl** command has many options that allow you to control trace display so that you can view only the states of interest. Many trace list options can be combined to increase the usefulness of the display.

Examples

To see how the trace list options are used, you need to capture a trace in the analyzer's memory. You can easily capture a trace by entering the following commands:

```
R>init -c
R>demo
R>t
R>r
```

Display the trace starting at state 11 by entering:

```
U>tl 11
```

To display the trace list

You will see:

```
 Line   addr,H    68020 Mnemonic                              count,R  seq
-----   --------  ----------------------------------------  ---------  ---
   11   0000040e   $1039         supr prgm word rd (ds16)    0.160 uS   .
   12   00000500   $00--         supr data byte wr (ds16)    0.120 uS   .
   13   00000410   $0000         supr prgm long rd (ds16)    0.120 uS   .
   14   00000412   $0500         supr prgm word rd (ds16)    0.200 uS   .
   15   00000414   $6600         supr prgm long rd (ds16)    0.120 uS   .
   16   00000416   $0006         supr prgm word rd (ds16)    0.160 uS   .
   17   00000500   $00--         supr data byte rd (ds16)    0.120 uS   .
   18   00000418   $6000         supr prgm long rd (ds16)    0.120 uS   .
   19   0000041a   $000E         supr prgm word rd (ds16)    0.200 uS   .
   20   0000041c   $6100         supr prgm long rd (ds16)    0.120 uS   .
```

Display three more states from the next states available by entering the command:

```
U>tl -n 3
```

You will see:

```
 Line   addr,H    68020 Mnemonic                              count,R  seq
-----   --------  ----------------------------------------  ---------  ---
   21   0000041e   $000C         supr prgm word rd (ds16)    0.120 uS   .
   22   00000428   $60E4         supr prgm long rd (ds16)    0.120 uS   .
   23   0000042a   $0C00         supr prgm word rd (ds16)    0.200 uS   .
```

Display the trace from the top by entering the command:

```
U>tl -t
```

You will see:

```
 Line   addr,H    68020 Mnemonic                              count,R  seq
-----   --------  ----------------------------------------  ---------  ---
    0   00000000   $0000         supr prgm long rd (ds16)         ---   +
    1   00000002   $1000         supr prgm word rd (ds16)    0.160 uS   .
    2   00000004   $0000         supr prgm long rd (ds16)    0.120 uS   .
```

Notice that only three states are displayed. This is because you reset the count parameter when you entered the command **tl -n 3**.

Display states 27 through 35 with the command:

```
U>tl 27..35
```

You will see:

```
 Line   addr,H    68020 Mnemonic                              count,R  seq
-----   --------  ----------------------------------------  ---------  ---
   27   0000040e   $1039         supr prgm word rd (ds16)    0.160 uS   .
   28   00000410   $0000         supr prgm long rd (ds16)    0.120 uS   .
   29   00000412   $0500         supr prgm word rd (ds16)    0.160 uS   .
   30   00000414   $6600         supr prgm long rd (ds16)    0.160 uS   .
   31   00000416   $0006         supr prgm word rd (ds16)    0.120 uS   .
   32   00000500   $00--         supr data byte rd (ds16)    0.120 uS   .
   33   00000418   $6000         supr prgm long rd (ds16)    0.120 uS   .
   34   0000041a   $000E         supr prgm word rd (ds16)    0.200 uS   .
   35   0000041c   $6100         supr prgm long rd (ds16)    0.120 uS   .
```

To prevent trace list header display

- Disable the display of the column headers in the trace list by typing: **tl -h <trace_opts>**

  where **<trace_opts>** are the other options you want for trace display.

  Disabling the column headers may be useful if you are saving the display output in a file on a host computer. Then the trace list displays can be concatenated to produce a continuous listing without interrupting headers.

Example

Capture a trace, then display it without headers:

```
R>init -c
R>demo
R>t
R>r
U>tl -h 0..5
```

You will see:

```
   0   00000000   $0000         supr prgm long rd (ds16)         ---   +
   1   00000002   $1000         supr prgm word rd (ds16)    0.160 uS   .
   2   00000004   $0000         supr prgm long rd (ds16)    0.120 uS   .
   3   00000006   $0400         supr prgm word rd (ds16)    0.320 uS   .
   4   00000400   $2E7C         supr prgm long rd (ds16)    0.120 uS   .
   5   00000402   $0000         supr prgm word rd (ds16)    0.200 uS   .
```

To control symbol and address display in the trace list

- To display only symbols in the address column of the trace list, type: **tl -s <list_opts>**
- To display only hexadecimal values in the address column of the trace list, type: **tl -a <list_opts>**
- To display both symbols and hexadecimal values for addresses in the address column of the trace list, type: **tl -e <list_opts>**

<list_opts> above are the other trace list options that you might select.

Display of symbols in the trace list's address column makes the list much easier to read and interpret. You must first download a symbol file to the emulator (see Chapter 1, "Quick Start"), or define some user symbols.

When you use the **-e** or **-s** options, symbols are also displayed in the **mne** disassembly field for operands.

Example

Capture a trace for the examples:

```
R>init -c
R>demo
R>t
R>r
```

Display only symbols:

```
U>tl -s 0..12
```

You will see:

```
 Line   addr,H    68020 Mnemonic                            count,R  seq
-----   --------  ---------------------------------------  ---------  ---
    0              $0000         supr prgm long rd (ds16)         ---   +
    1              $1000         supr prgm word rd (ds16)    0.160 uS   .
    2              $0000         supr prgm long rd (ds16)    0.120 uS   .
    3              $0400         supr prgm word rd (ds16)    0.320 uS   .
    4   emo:Main   $2E7C         supr prgm long rd (ds16)    0.120 uS   .
    5              $0000         supr prgm word rd (ds16)    0.200 uS   .
    6              $1000         supr prgm long rd (ds16)    0.120 uS   .
    7              $13FC         supr prgm word rd (ds16)    0.200 uS   .
    8              $0000         supr prgm long rd (ds16)    0.120 uS   .
    9              $0000         supr prgm word rd (ds16)    0.240 uS   .
   10              $0500         supr prgm long rd (ds16)    0.120 uS   .
   11   emo:Loop   $1039         supr prgm word rd (ds16)    0.160 uS   .
   12   md_Input   $00--         supr data byte wr (ds16)    0.120 uS   .
```

Display only hexadecimal values for addresses:

```
M>tl -a 0..12
```

You will see:

```
 Line   addr,H    68020 Mnemonic                            count,R  seq
-----   --------  ---------------------------------------  ---------  ---
    0   00000000   $0000         supr prgm long rd (ds16)         ---   +
    1   00000002   $1000         supr prgm word rd (ds16)    0.160 uS   .
    2   00000004   $0000         supr prgm long rd (ds16)    0.120 uS   .
    3   00000006   $0400         supr prgm word rd (ds16)    0.320 uS   .
    4   00000400   $2E7C         supr prgm long rd (ds16)    0.120 uS   .
    5   00000402   $0000         supr prgm word rd (ds16)    0.200 uS   .
    6   00000404   $1000         supr prgm long rd (ds16)    0.120 uS   .
    7   00000406   $13FC         supr prgm word rd (ds16)    0.200 uS   .
    8   00000408   $0000         supr prgm long rd (ds16)    0.120 uS   .
    9   0000040a   $0000         supr prgm word rd (ds16)    0.240 uS   .
   10   0000040c   $0500         supr prgm long rd (ds16)    0.120 uS   .
   11   0000040e   $1039         supr prgm word rd (ds16)    0.160 uS   .
   12   00000500   $00--         supr data byte wr (ds16)    0.120 uS   .
```

Display both symbols and hexadecimal values for addresses:

```
M>tl -e 0..12
```

You will see:

```
 Line   addr,H    68020 Mnemonic                              count,R  seq
-----   --------  ----------------------------------------  ---------  ---
    0   00000000   $0000         supr prgm long rd (ds16)         ---   +
    1   00000002   $1000         supr prgm word rd (ds16)    0.160 uS   .
    2   00000004   $0000         supr prgm long rd (ds16)    0.120 uS   .
    3   00000006   $0400         supr prgm word rd (ds16)    0.320 uS   .
    4   emo:Main   $2E7C         supr prgm long rd (ds16)    0.120 uS   .
    5   00000402   $0000         supr prgm word rd (ds16)    0.200 uS   .
    6   00000404   $1000         supr prgm long rd (ds16)    0.120 uS   .
    7   00000406   $13FC         supr prgm word rd (ds16)    0.200 uS   .
    8   00000408   $0000         supr prgm long rd (ds16)    0.120 uS   .
    9   0000040a   $0000         supr prgm word rd (ds16)    0.240 uS   .
   10   0000040c   $0500         supr prgm long rd (ds16)    0.120 uS   .
   11   emo:Loop   $1039         supr prgm word rd (ds16)    0.160 uS   .
   12   md_Input   $00--         supr data byte wr (ds16)    0.120 uS   .
```

To control trace list disassembly and dequeueing

- To disassemble the trace list, type: **tl -d <list_opts>**
- To display all bus cycles in the disassembled trace list, type: **tl -oa <list_opts>**
- To display only instruction cycles in the disassembled trace list, type: **tl -oi <list_opts>**
- To dequeue the trace list, type: **tl -od <list_opts>**
- To display the non-dequeued trace list, type: **tl -on <list_opts>**
- To disassemble the trace list from the lower word of a starting state, type: **tl -ol <list_opts>**
- To tell the analyzer software which operand belongs with a particular starting state, type:
  tl <list_opts> <instruction_state> <operand_state>

where **<instruction state>** is an instruction state in the trace list and **<operand_state>** is the first operand cycle for that instruction.

<list_opts> above are the other trace list options that you might select.

The MC68020 and MC68030/MC68EC030 trace lists display states in the order they were captured by the analyzer. The **-d** option causes disassembly of the trace-list content. The **-o**<options> control how the trace list is disassembled. Option **-oa** shows all bus cycles (instructions and operands). Option **-oi** shows only the instruction cycles. Option **-ol** starts disassembly with the low word of the specified trace list line number. Option **-od** dequeues the disassembled trace and **-on** calls for the non-dequeued trace list.

A captured state must be a long word (having a high word and a low word. An opcode can appear in either word (or both words). The disassembler starts with the high word in the trace list line number you specify in your command. If the disassembled trace list isn't what you expected, try using the **-ol** option to force disassembly to begin with the low word.

A dequeued trace list (option **-od**) shows operand cycles immediately following the instructions that caused them, and suppresses unexecuted instructions. If you choose a non-dequeued trace list (option **-on**), all emulation-bus activity is shown in the order it occurred, whether or not it was executed.

To help the dequeuer select the correct operand cycles to align with a particular opcode, use a command such as: **tl -d -od 50 62**, which means align the operand cycles on line 62 with the instruction on line 50. You can resynchronize the dequeuer at any point in the display if you see a problem.

Example

Capture a trace for the example:

```
R>init -c
R>demo
R>t
R>r
```

Display disassembled:

```
U>tl -d 0..12
```

You will see:

```
 Line   addr,H    68020 Mnemonic                              count,R  seq
-----   --------  ----------------------------------------  ---------  ---
    0   00000000  ORI.B    #$1000,D0                             ---   +
    1   00000002   $1000          supr prgm word rd (ds16)    0.160 uS   .
    2   00000004  ORI.B    #$400,D0                           0.120 uS   .
    3   00000006   $0400          supr prgm word rd (ds16)    0.280 uS   .
    4   00000400  MOVEA.L  #$00001000,A7                      0.160 uS   .
    5   00000402   $0000          supr prgm word rd (ds16)    0.200 uS   .
    6   00000404   $1000          supr prgm long rd (ds16)    0.120 uS   .
    7   00000406  MOVE.B   #$00,$00000500                     0.200 uS   .
    8   00000408   $0000          supr prgm long rd (ds16)    0.120 uS   .
    9   0000040a   $0000          supr prgm word rd (ds16)    0.200 uS   .
   10   0000040c   $0500          supr prgm long rd (ds16)    0.160 uS   .
   11   0000040e  MOVE.B   $00000500,D0                       0.160 uS   .
   12   00000500   $00--          supr data byte wr (ds16)    0.120 uS   .
```

The default is all cycles displayed (-oa).

Display only instructions:

```
U>tl -oi 0..12
```

```
 Line   addr,H    68020 Mnemonic                              count,R  seq
-----   --------  ----------------------------------------  ---------  ---
    0   00000000  ORI.B    #$1000,D0                             ---   +
    2   00000004  ORI.B    #$400,D0                           0.280 uS   .
    4   00000400  MOVEA.L  #$00001000,A7                      0.440 uS   .
    7   00000406  MOVE.B   #$00,$00000500                     0.520 uS   .
   11   0000040e  MOVE.B   $00000500,D0                       0.640 uS   .
```

Display part of the trace with all cycles shown and instructions dequeued:

```
U>tl -oda 11..30
```

```
 Line   addr,H    68020 Mnemonic                              count,R  seq
-----   --------  ----------------------------------------  ---------  ---
   11   0000040e  MOVE.B   $00000500,D0                       0.640 uS   .
       =00000500   src  sdata rd:$00
   15   00000414  BNE.W    $0000041C ?TAKEN?                  0.560 uS   .
   18   00000418  BRA.W    $00000428                          0.400 uS   .
   22   00000428  BRA.B    $0000040E                          0.560 uS   .
   27   0000040e  MOVE.B   $00000500,D0                       0.720 uS   .
       =00000500   src  sdata rd:$00
   30   00000414  BNE.W    $0000041C ?TAKEN?                  0.400 uS   .
```

?TAKEN? means the dequeuer was not able to determine whether or not the branch was taken. If you read down the trace list and decide that the branch was taken, use the **tl -d -od <Line number>** command to restart disassembly at the trace list line number of the branch destination. You will need to include the **-ol** option if the destination opcode is in the low word at the destination address. You may need to

resynchronize alignment of operand cycles with the instruction at the branch address, as described just before the examples. TAKEN would have been shown beside the branch if the dequeuer had determined that the branch was taken. NOT TAKEN would have been shown if the dequeuer had determined that the branch was not taken.

To change the trace depth

- Change the analyzer trace depth from 512 to 1024 states by typing: **tcq none**

  The analyzer's state/time counter uses half of the analyzer's state memory resources. Therefore, when you use time count (**tcq time**) or state count (**tcq any**, etc.) the analyzer's trace depth is 512 states. You can double the trace depth by disabling state/time counting.

Analyzing Program Execution in the MC68030 Emulator with its MMU Enabled

Most emulation and analysis commands that require an address as part of the command use logical addresses. When the MC68030 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

- Run your program to the point where you are sure the MMU is set up.
- Break to the monitor program with the command: **b**.
- Load the deMMUer with the command: **dmmu -l** or **dmmu -lv**.
- Enable the deMMUer with the command: **dmmu -e**.
- Continue execution of your target program with the command: **r**,
  or restart the program with the command: **r rst**.

To pick the place to load the deMMUer, you might set a software 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. When the software breakpoint is hit (emulator running in foreground monitor), you can load the deMMUer.

Whether you continue your program or restart it, the deMMUer will have the ability to reverse translate the physical addresses according to the MMU setup at the time you issued the load-deMMUer command. The deMMUer will remain loaded even if you reset the emulation processor.

If you restart your program, you can use the analyzer to see how the MMU tables are created and how the program operates.

Address ranges will be reverse translated correctly if they are translated by the setup of the MMU that existed when you issued the **dmmu -lv** 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 trace program execution in physical address space

Disable the deMMUer with the command: **dmmu -d**.

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.

Using the Trace Sequencer

The analyzer trace sequencer is the key to powerful analyzer measurements. You define a series of states that lead to the trigger condition and a set of conditions for analyzer storage of bus cycles. The trace sequencer hardware uses these definitions to control analyzer storage. Thus, you can capture only bus cycles that are relevant to your problem. The sequencer defines a filter, effectively removing bus cycles from the trace storage that aren't important to the current measurement.

The sequencer operates in either easy or complex configuration. The easy configuration has simpler setup but less power than the complex configuration. These configurations are described in later sections.

To change the trace configuration

- To change the trace configuration to easy, type: **tcf -e**
- To change the trace configuration to complex, type: **tcf -c**

After you initialize the emulator (by cycling power, or by using the **init** command), or the analyzer (with **tinit**), the analyzer is reset to easy configuration.

Change the trace configuration to complex if you need to specify a trigger sequence that includes more than four sequence states with multiway branches, or if you need different storage qualifiers at each level of the sequence. Change the trace configuration to easy if you need only one storage qualifier and have a simple trigger sequence (less than four sequence terms with global restart).

Using Easy Configuration

You use easy configuration to set up slightly more complex trigger specifications than the **tg** command will allow. You don't have access to the full power of the analyzer, but the command set is simplified.

In easy configuration, the analyzer has four sequence terms, a global restart term, and a global storage qualifier. The branch out of the last sequence term is the trigger.

Expressions in easy configuration are limited to simple (in)equalities of analyzer to integer values. The analyzer does allow you to count states or time and specify prestore qualifiers.

To create a simple expression

- To define a simple expression, create one or more equalities of the form **<label>=<expr>** joined by the operator **and** or one or more inequalities of the form **<label>!=<expr>** joined by the operator **or**.

- To define a range expression, create an equality of the form: **<label>=<expr>..<expr>**

  Simple expressions allow you to build qualifiers that have multiple conditions. Notice that the conditions must use the same logical operators; you can't mix **and** and **or** in an expression.

  There is only one range expression available. If you try to define a second range expression, you will see an error message.

  See "To create an expression" earlier in this chapter for more information on the **<expr>** parameter. See "To create analyzer labels" for more about the **<label>** parameter.

Example

Here are some valid simple expressions (using the demo program symbols and the predefined equates):

```
addr=handle_msg:Print_Msg
data=41
addr=demo:Cmd_Input and stat=read
addr!=handle_msg:Cmd_B or addr!=Cmd_I
```

You can't combine the **and** and **or** logical operators, nor can you mix the **and** operator with != or the **or** operator with =. Here are some invalid simple expressions:

```
addr!=handle_msg:Cmd_A and stat!=read
data=3e or stat=write
```

To build these types of expressions, you must use the analyzer's complex configuration.

To insert a sequence term

- Insert a new sequence term numbered <**TERM#**> by typing: **tsq -i** <**TERM#**>

There are only four sequence terms available in easy configuration. Therefore, <**TERM#**> must be in the range 1..4. If you specify a number that is already in use, that term and succeeding terms are incremented.

Examples

Initialize the analyzer and insert a new sequence term before term 1:

```
R>tinit
R>tsq -i 1
```

Now add a term after term 2:

```
R>tsq -i 3
```

To remove a sequence term

- Delete an existing sequence term numbered **<TERM#>** by typing: **tsq -d <TERM#>**

  You may want to delete sequence terms to remove unneeded qualifications from the sequence specification. When you delete a sequence term, any terms above it are decremented to fill the gap.

Example

Suppose that you have inserted the sequence terms as given for the examples in "To insert a sequence term" above. There are now three sequence terms, and you want to remove terms 1 and 3. Enter the commands:

```
R>tsq -d 1
R>tsq -d 2
```

Notice that you remove term 2 in the second command. This was term 3 until removal of term 1 caused the terms to be rennumbered.

To reset the sequencer

- Reset the trace sequencer by typing: **tsq -r**

  When you reset the sequencer, it is reduced to a one-term sequence that stores all states and triggers on the first occurrence of any state. This is equivalent to the command sequence: **tg any;tsto any;telif never**.

To define a primary branch

- To set the primary branch qualifier for a term given by **<TERM#>**, type: **tif <TERM#> <simple_expr> <count>**

  where **<count>** is an optional parameter that specifies the number of times that **<expr>** must occur to satisfy the branch qualifier.

- To display the primary branch qualifier for a term given by **<TERM#>**, type: **tif <TERM#>**

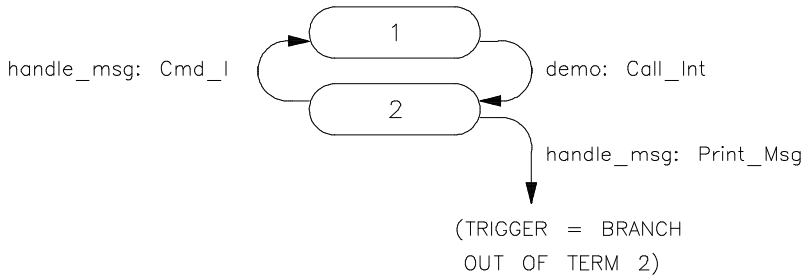
- To display all primary branch qualifiers, type: **tif**

  You use the primary branch qualifiers to set the sequence of conditions that must be satisfied to reach the trigger term and trigger the analyzer. For example, you might want to have the analyzer find a certain address value, then a data read, and then trigger on another address value. This requires three sequence terms.

Example

Suppose that you want to trace the sequence of code from Call_Int through Print_Msg (in the demo program) only if an "A" or "B" command occurs. You don't want the analyzer to store data if Cmd_I is found.

The analyzer sequencer state diagram for this measurement looks like the following:



Set up the measurement:

```
R>init -c
R>demo
R>tsq -i 1
R>tif 1 addr=demo:Call_Int
```

```
R>tif 2 addr=handle_msg:Print_Msg
R>telif addr=handle_msg:Cmd_I
R>tp -b 20
R>t
R>r
U>tl
```

No trace data has been captured, because no program command was entered. Enter the commands:

```
U>m -db demo:Cmd_Input=41
U>tl -d -s -od -20
```

The symbol Call_Int is at the top of the trace list.

To define a global restart term

- To set the sequencer's global restart term, type: **telif <label>=<simple_expr>**
- To display the sequencer's global restart term, type: **telif**

You use the global restart term to restart the trace measurement when a certain condition occurs. This can be useful to filter out bus activity that isn't relevant to the problem. For example, you might have a hashing routine that fails for one key value. You could define the restart term to be "not this data value," which will restart the analyzer for every other key except the one of interest.

See Chapter 11, "Expressions," for more information on simple expressions.

Example

See the example given in the section "To define a primary branch."

To display the current sequencer settings

- Display the current sequencer settings by typing: **tsq**

When you use the **tsq** command without any parameters, the emulator displays all branch and storage qualifiers and the trigger term position.

This command is especially useful for checking your work after you define a complicated trace specification.

Example

The section "To define a primary branch" gives an example of a sequencer setup for a particular measurement problem. To display that sequencer definition, enter the command:

```
U>tsq
```

You will see:

```
tif 1 addr=demo:Call_Int
tif 2 addr=handle_msg:Print_Msg
tsto all
telif addr=handle_msg:Cmd_I
```

Using Complex Configuration

Enter the command: **tcf -c**

Complex configuration allows you to make analyzer measurements that require more powerful trigger logic or need multiple storage specifications.

There are eight sequence terms in complex configuration. Each term has a primary and secondary branch qualifier that allow branching to any other term when that qualifier is matched. Also, each sequencer term has a unique storage qualifier.

The sequence terms are always available in complex configuration. You don't need to insert them as you do with easy configuration.

To build expressions in complex configuration, you first assign combinations of simple expressions to one of eight pattern variables. There is also a range variable to specify address or data ranges. Then you assign combinations of the pattern variables and range to a branch or storage qualifier. The expressions can include various logical operators. For example, you may want to specify a condition involving a specific data value, but you want to exclude that data value if it is found within a particular address range. The complex configuration allows this.

To assign the trigger term

- To assign the trigger term to **<TERM#>**, type: **tsq -t <TERM#>**

  where **<TERM#>** is a sequencer term number.

- To display the current trigger term assignment, type: **tsq -t**

  The trigger term may be any one of terms 2..8 (it cannot be term 1). The analyzer will trigger on entry to the trigger term.

Example

Move the trigger from term 2 to term 6:

```
R>init -c
R>tcf -c
R>tsq -t 6
```

Capture a simple trace and display it with this trigger specification:

```
R>demo
R>tp c
R>t
R>r
U>tl -5..5
```

You will see:

```
 Line   addr,H    68020 Mnemonic                              count,R  seq
-----   --------  ----------------------------------------  ---------  ---
   -5
   -4   00000000   $0000         supr prgm long rd (ds16)         ---   +
   -3   00000002   $1000         supr prgm word rd (ds16)    0.160 uS   +
   -2   00000004   $0000         supr prgm long rd (ds16)    0.120 uS   +
   -1   00000006   $0400         supr prgm word rd (ds16)    0.280 uS   +
    0   00000400   $2E7C         supr prgm long rd (ds16)    0.160 uS   +
    1   00000402   $0000         supr prgm word rd (ds16)    0.200 uS   +
    2   00000404   $1000         supr prgm long rd (ds16)    0.120 uS   +
    3   00000406   $13FC         supr prgm word rd (ds16)    0.200 uS   .
    4   00000408   $0000         supr prgm long rd (ds16)    0.120 uS   .
    5   0000040a   $0000         supr prgm word rd (ds16)    0.200 uS   .
```

Notice that the sequencer changes states 7 times, one for each sequence level plus the trigger.

To reset the sequencer

- Reset the trace sequencer by typing: **tsq -r**

When you reset the sequencer, all primary branch qualifiers are set to jump to the next term on any condition (except for term 8, which is set to never). All secondary branch qualifiers are disabled. The trigger term is set to term 2, and the storage qualifier for all sequence terms is set to all states.

Example

To reset the sequencer, enter:

```
M>tsq -r
```

To view the new trace sequence, enter:

```
M>tsq
```

The sequencer setup looks like:

```
tif 1 any 2
tif 2 any 3
tif 3 any 4
tif 4 any 5
tif 5 any 6
tif 6 any 7
tif 7 any 8
tif 8 never
tsq -t 2
tsto 1 all
tsto 2 all
tsto 3 all
tsto 4 all
tsto 5 all
tsto 6 all
tsto 7 all
tsto 8 all
telif 1 never
telif 2 never
telif 3 never
telif 4 never
telif 5 never
telif 6 never
telif 7 never
telif 8 never
```

To display the current sequencer settings

- Display the current sequencer settings by typing: **tsq**

When you use the **tsq** command without any parameters, the emulator displays all branch and storage qualifiers and the trigger term position.

This command is especially useful for checking your work after you define a complicated trace specification.

To define trace patterns

- To define a trace pattern, type: **tpat <PATTERN#> <simple_expr>**
- To display the expression for a given trace pattern, type: **tpat <PATTERN#>**
- To display the expressions for all trace patterns, type: **tpat**

In complex configuration, the analyzer provides eight pattern variables to which you assign simple expressions. Then you use these patterns to build more complicated expressions for the primary and secondary branch qualifiers.

See Chapter 11, “Expressions,” for more information on expressions.

Example

Suppose that you want to trace and trigger on an access to either Cmd_A, Cmd_B or Cmd_I in the demo program. Enter the commands:

```
R>init -c
R>demo
R>tcf -c
R>tpat p1 addr=handle_msg:Cmd_A
R>tpat p2 addr=Cmd_B
R>tpat p3 addr=Cmd_I
R>tg p1|p2|p3
R>t
```

Now you can run the demo program and enter various program commands by modifying the Cmd_Input location. When you list the trace, the analyzer will have triggered on the first access to any of the locations Cmd_A, Cmd_B or Cmd_I.

To define a range qualifier

- To define the range pattern **r** to be the set of states including two expressions, type: **trng <label>=<expr>..<expr>**

- To define the range pattern **r** to be all states, type: **trng any**

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 set from the lower to upper boundaries of the lookup table.

You can create ranges for either address or data.

Example

Suppose that you want to trigger on reads from the message storage area (Msg_A through End_Msgs) and store only those reads. Enter the following commands:

```
R>init -c
R>demo
R>tcf -c
R>trng addr=handle_msg:Msg_A..End_Msgs
R>tpat p5 stat=read
R>tg r and p5
R>tsto r and p5
R>t
R>r
U>m -db demo:Cmd_Input=41
U>tlb byte0 32..39
U>tlb byte1 40..47
U>tlb byte2 48..55
U>tlb byte3 56..63
U>tf addr,h byte2,a byte3,a
U>tl
```

Note that bytes alternate because each character is picked up from a different half of a 16-bit word.

```
  Line  addr,H    byte2,A  byte3,A
-----  --------  -------  -------
    0  00000502  |        C
    1  00000503  o        .
    2  00000504  |        m
    3  00000505  m        .
    4  00000506  |        a
    5  00000507  n        .
    6  00000508  |        d
    7  00000509  .        .
    8  0000050a  |        A
```

```
      9  0000050b  .        .
```

The first reads from the Msg_A storage buffer are shown in the trace list. The **a** option to the **lowdata** label in the trace format specifies ASCII data display.

To create a complex expression

- Create a complex expression by combining trace patterns **p1..p8** and the range qualifier **r** using intraset and interset operators.

  The rules for combining patterns and ranges are as follows:

  The patterns, range and arm qualifier are divided into two disjoint sets.

  <SET1>={p1,p2,p3,p4,r,!r}
  <SET2>={p5,p6,p7,p8,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.

  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.

  See Chapter 11, “Expressions,” for more information.

Example

Here are some valid complex expressions:

```
p1~p2~r
r and p5
p5 or p1
p2|p1|r
p1|p2 or p5~p6
```

The following expressions are invalid:

```
p1~p2|r
p1 and p2
p1~p2 and p3 and p5 and p7
```

The last expression is invalid because you can't repeat different sets to extend the expression.

If you're having trouble achieving the necessary expression, try using DeMorgan's Theorem. Suppose you want to trace on:

(addr=2000) NAND (data=23)

There is no NAND function in the expression syntax. But, the above is equivalent to:

(addr!=2000) OR (data!=23)

To define a primary branch term

- To define a primary branch qualifier for the term given by **<TERM#>**, type: **tif <TERM#> <complex_expr> [<branch_term> <count>]**

  where **<branch_term>** is an optional term number indicating the term to branch to when the **<complex_expr>** is satisfied. The default is to branch to the next higher-numbered term, except for term 8, which branches to itself.

  <count> is an optional parameter that specifies the number of times that **<complex_expr**> must occur to satisfy the branch qualifier.

- To display the primary branch qualifier for the term given by **<TERM#>**, type: **tif <TERM#>**

- To display all primary branch qualifiers, type: **tif**

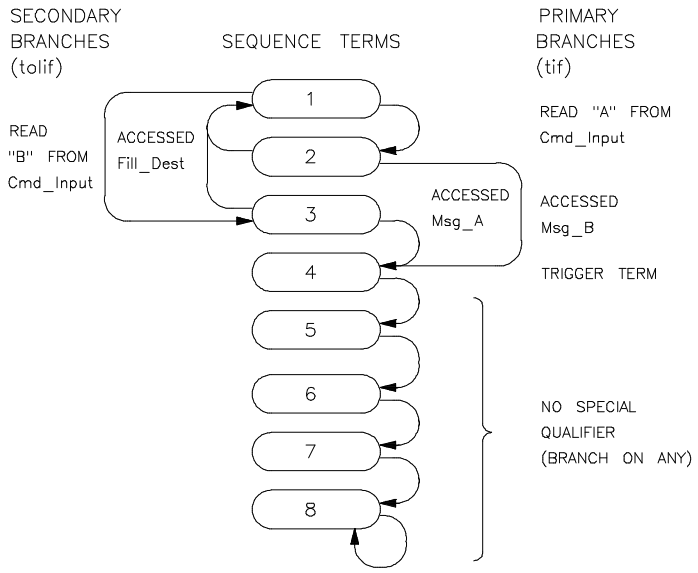
  The primary branch qualifier defines the main path from a given sequencer term to another term (or the same term). If both the primary and secondary branch qualifiers are satisfied simultaneously, the primary branch is taken.

  Usually, you'll use the primary branch qualifiers to define a sequence of states that must be satisfied to reach the trigger condition.

Example

Suppose that there was an intermittent problem with the demo program, where the message for command A was sometimes output when command B was entered, and vice versa. (This isn't a true problem in this program, but is given simply to show the power of the complex analyzer configuration.)

The following state diagram shows how the analyzer will transition from term to term in the sequencer:



Set up the analyzer to match this state diagram:

```
R>init -c
R>demo
R>tcf -c
R>tp e
R>tsq -t 4
R>tpat p1 addr=demo:Cmd_Input and stat=read
R>tlb byte3 56..63
R>tpat p5 byte 3
R>tpat p6 byte3
R>tpat p2 addr=handle_msg:Msg_B
R>tpat p3 addr=handle_msg:Msg_A
R>tpat p4 addr=handle_msg:Fill_Dest
R>tif 1 p1 and p5 2
R>telif 1 p1 and p6 3
R>tif 2 p2 4
R>tif 3 p3 4
R>telif 2 p4 1
R>telif 3 p4 1
```

Patch the code to make it fail:

```
M>m -db 442=05,13
M>m -db 450=05,02
```

The **tpat** commands assign the simple expressions needed. Notice that the address condition was assigned to **p1** and the data conditions to **p5** and **p6**—this is done so the **and** qualifier can be used between the patterns. The **tsq** command sets the trigger term to be the exit from sequencer term 4. The **tif** and **telif** commands define the branch conditions through the sequencer. Finally, the **tp** command sets the trigger position to the end of the trace. This allows you to see the states that lead to an incorrect message being printed. To begin testing for the error condition, you would start a trace with the **t** command, then run the program and begin entering various combinations of command A and command B.

To define a secondary branch term

- To define a secondary branch qualifier for the term given by **<TERM#>**, type: **telif <TERM#> <complex_expr> [<branch_term>]**

  where **<branch_term>** is an optional term number indicating the term to branch to when the **<complex_expr>** is satisfied. The default is to branch to the next higher-numbered term, except for term 8, which branches to itself.

- To display the secondary branch qualifier for the term given by **<TERM#>**, type: **telif <TERM#>**

- To display all secondary branch qualifiers, type: **telif**

  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.

  Since the secondary branch qualifier is unique for each sequence term, it is more flexible than the global restart qualifier in easy configuration. You can use it as a global restart by making all secondary branch qualifiers identical, and having them

restart the trace sequence, or you can use the secondary branch as an alternate path to the trigger if more than one sequence of conditions is acceptable.

To define complex storage qualifiers

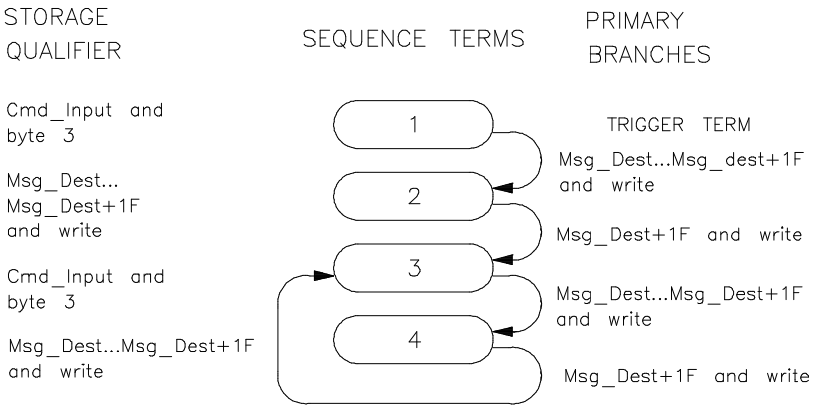
- To define a storage qualifier for the term given by <**TERM#**>, type: **tsto <TERM#> <complex_expr>**
- To define a global storage qualifier (applied to all states), type: **tsto <complex_expr>**
- To display the storage qualifier for the term given by <**TERM#**>, type: **tsto <TERM#>**
- To display the storage qualifier for all terms, type: **tsto**

In complex mode, there are eight storage qualifiers, one for each sequencer 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.

You can use the storage specifications with the primary and secondary branch qualifiers to trace on “windows” of activity, such as certain program subroutines.

Example

Suppose you wanted to see the ASCII data writes to the output area, and the commands that caused them. A state diagram for the analyzer looks like the following:



Now enter the commands:

```
R>init -c
R>demo
R>tcf -c
R>tpat p1 addr=demo:Cmd_Input
R>tpat p2 addr=handle_msg:Msg_Dest+1f
R>tlb byte3 56..63
R>tpat p5 byte3!=0
R>tpat p6 stat=write
R>trng addr=Msg_Dest..Msg_Dest+1f
R>tsto 1 p1 and p5
R>tsto 2 r and p6
R>tsto 3 p1 and p5
R>tsto 4 r and p6
R>tif 1 r and p6 2
R>tif 2 p2 and p6 3
R>tf byte3,a seq
```

The above commands set the sequencer to store the reads of non-zero values from Cmd_Input (when in terms 1 and 3) and the writes of data to the MSG_DEST area (when in terms 2 and 4). The sequencer toggles from term 1 to term 2 (or from 3 to 4) when writes to the Msg_Dest area occur, and from terms 2 to 3 (or 4 to 3) when the last byte is written to the destination area. Also, the trace format is set to show

only the lower byte of the data bus and the sequencer activity (a "+" is shown in the **seq** column when the sequencer changes states).

Enter the commands:

```
R>t
R>r
U>m -db demo:Cmd_Input=41
U>m -db Cmd_Input=42
U>m -db Cmd_Input=43
U>tl *
```

You'll see the commands and the corresponding message written to the Msg_Dest area.

Setting Analyzer Clocks

The HP 64700 Series emulator design allows up to five clock signals for emulation and external analysis. These are J, K, L, M, and N clocks. The HP 64748 and HP 64747 emulators generate the L clock to drive the emulation analyzer. The other clocks are not used.

The Terminal Interface provides the **tck** and **tsck** commands to configure the clock signals. For the emulation analyzer, **tck** and **tsck** are provided primarily for system initialization and control through higher-level interfaces. You can use the **tck** command to change qualification of user/background code execution. You also use this command to specify the maximum data rate that the analyzer will see, which affects the state/time counter.

To trace user/background code execution

- To trace only user code, type: **tck -u**
- To trace only background code, type: **tck -b**
- To trace both user and background code, type: **tck -ub**

The emulation-bus analyzer has built-in qualifiers that allow you to select whether the analyzer captures user code, background code, or both. Usually, you'll want to trace only user code. If you're trying to solve a problem with emulator and target system interaction, you may want to trace both user and background code. The background code only setting is rarely useful for most emulation work.

To configure the analyzer clock

- To set the analyzer for a slow data rate (less than or equal to 16.67 MHz), type: **tck -s S**

- To set the analyzer for a fast data rate (between 16.67 and 20 MHz), type: **tck -s F**

- To set the analyzer for a very fast data rate (between 20 and 25 MHz), type: **tck -s VF**

The emulation-bus analyzer can capture bus cycles at data rates up to 25 MHz. However, the trace state and time counters are limited to lower speeds. The MC68020 analyzer clock is set to **tck -s S** by default, because the data rate is sufficiently low at the maximum clock rate of 33 MHz.

The MC68030/EC030 analyzer clock is set to **tck -s VF** by default. This processor has more complicated requirements due to the burst and synchronous access modes. 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:

$$\text{Analyzer Clock Rate} = \frac{\text{Processor Clock Rate}}{(1 + \text{number of wait states})}$$

To determine the correct setting for the **tck -s** command in the MC68030/EC030 emulator, calculate the maximum data rate by using the above equation. Remember that the emulator always inserts one wait state for all synchronous and burst accesses to emulation memory, and also must insert one wait state for synchronous and burst accesses to target memory when the external clock is greater than or equal to 25 MHz. (See Chapter 7, “Configuring the Emulator,” for more information.) Then choose the data rate option according to the data rate.

The trace state and time count qualifiers are limited by the analyzer clock rate settings as follows:

Analyzer clock rate	tck setting	Valid tcq options
clock ≤ 16.67 MHz	tck -s S	tcq <state> tcq time
clock ≤ 20 MHz	tck -s F	tcq <state>
clock ≤ 25 MHz	tck -s VF	tcq none

Example

Suppose that you are running the MC68030/EC030 processor at 40 MHz. You have set **cf emwait=en** since target memory requires one wait state for synchronous/burst accesses over 25 MHz. The resulting data rate is 20 MHz, so you enter the following commands:

```
R>tcq none
R>tck -s F
```

Note that you set **tcq none**. Since the clock rate is between 16.67 and 20 MHz, you could choose to count states by choosing the appropriate **tcq** options. However, you cannot use **tcq time**.

Using Other Analyzer Features

The analyzer has other features that can be used in all configurations to make trace measurements easier to interpret or capture 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 the caller of a subroutine.
- Count qualifiers allow you to count states or time.
- Trace activity measurements allow you to see whether a particular analyzer signal is high, low or moving.
- Equates save keystrokes by allowing you to assign names to commonly used values. The names can be used in analyzer specifications.

To define a prestore qualifier

- To define a prestore qualifier when you're using easy configuration, type: **tpq <simple_expr>**
- To define a prestore qualifier when you're using complex configuration, type: **tpq <complex_expr>**

You use the prestore qualifier to save states that are related to other routines that you're tracing. For example, you might be tracing a subprogram, and want to see which program called it. You specify the caller as a prestore specification.

Example

Suppose that you want to trigger on a call to Print_Msg and prestore the command that called Print_Msg.

Enter the commands:

```
R>init -c
R>demo
```

```
R>tcf -c
R>trng addr=handle_msg:Mst_Dest..Msg_Dest+1f
R>tpat p1 addr=handle_msg:Print_Msg
R>tpat p5 addr=handle_msg:Cmd_A
R>tpat p6 addr=Cmd_B
R>tpat p7 addr=Cmd_I
R>tpq p5|p6|p7
R>tg p1
R>tsto r
R>t
R>r
U>m -db Cmd_Input=41
U>tl -e
```

To count states or time

- To measure the amount of time for each analyzer storage state, type: **tcq time**
- To measure the number of occurrences of a particular bus state in easy configuration, type: **tcq <simple_expr>**
- To measure the number of occurrences of a particular bus state in complex configuration, type:
  tcq <complex_expr>
- To disable analyzer state/time counting, type: **tcq none**
- To check the current setting of the count qualifier, type: **tcq**

  The trace count qualifier can be used to measure time for each storage state or occurrence counts of a particular bus state. You can display these values either relative to the last stored state (relative mode) or relate to the trigger state (absolute mode). You change this using the **tf** command. See “To Change the Trace Format” earlier in this chapter.

The MC68020 emulator defaults to **tcq time**. The MC68030/EC030 emulator, because of its higher bus cycle rates, defaults to **tcq none**. See "To configure the analyzer clock."

Example

Suppose you want to count the number of writes to the Msg_Dest area in the demo program. Enter the commands:

```
R>init -c
R>demo
R>tcf -c
R>trng addr=handle_msg:Msg_Dest..Msg_Dest+1f
R>tpat p5 stat=write
R>tcq r and p5
R>tsto r and p5
R>tg r and p5
```

The above commands trigger the analyzer when a nonzero command is input and interpreted by the command interpreter routine. The analyzer stores and counts only write cycles to the message output area.

Enter the commands:

```
R>t
R>r
U>m -db demo:Cmd_Input=41
U>tf addr,H,12 mne count,a seq
U>tl -e
```

These commands start a trace, then run the demo program and provide input. Before displaying the trace list, the count is changed to absolute format.

When you display the trace list, the count shows the total number of writes to the Msg_Dest area.

To check trace signal activity

- Display analyzer trace signal activity by typing: **ta**

The **ta** command can help you check target system operation. For each analyzer signal line, the **ta** command will display:

- 0 if the signal is low.
- 1 if the signal is high.
- ? if the signal is moving.

To define equates

- Define an equate with the name **<NAME>** by typing: **equ <NAME>=<expr>**

Equates allow you to type an expression once and recall it for later use. Some values for which you may want to define an equate include occurrence counts, status values, and table offsets.

Because the emulator has symbol-handling capability, you usually won't define equates for address values. It's usually better to download symbols from your host computer or use the **sym** command to define user symbols instead. That way, the symbols will appear in the trace list. Equates can't be shown in the trace list.

Example

Suppose you have a two-dimensional matrix, and you often want to specify a particular row in an analyzer command. If the matrix is 10 bytes square, you can define an equate as follows:

```
M>equ rown=10
```

Suppose that the base address of the matrix is in the symbol mymatrix. Then, you can specify the 22nd location in the matrix as mymatrix+22 or as mymatrix+2*rown+2.

To display equates

- To display the definition of an equate named **<NAME>**, type: **equ <NAME>**
- To display the definition of all equates, type: **equ**

To delete equates

- To delete an equate given by **<NAME>**, type: **equ -d <NAME>**
- To delete all equates, type: **equ -d ***

  Equates use system memory, so you may want to delete equates you are no longer using. This frees memory and makes the equate display easier to read.

  Be sure that you want to delete all equates before using the **equ -d *** command. System-defined equates are deleted if you use this command, but they will be redefined if you initialize the emulator (with the **init** command or by cycling power).

6

Making Coordinated Measurements

Use the Coordinated Measurement Bus to start and stop multiple emulators and analyzers

Basic Elements of Coordinated Measurements

The Coordinated Measurement Bus (CMB) connects multiple emulators and allows you to make synchronous measurements between those emulators.

For example, you might have a target system that contains an MC68020 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 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	This is an active low signal. It can be driven by the analyzer associated with a particular emulator, or used by another device on the CMB to arm the analyzer associated with that emulator.
READY	This is an active high signal. It is used for synchronized start and stop of multiple emulators. When CMB run control interaction is enabled, all emulators must break to the monitor upon receiving a false READY signal, and will not return to the user program until this line is true.
EXECUTE	This is an active low signal. It serves as a global interrupt signal. On receipt of an EXECUTE signal, each emulator must interrupt its current measurement and execute a previously-specified run or trace measurement.

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 control and respond to 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
bnct	Sets drivers and receivers of BNC trigger
cmb	Enables/disables CMB interaction
cmbt	Sets drivers and receivers of CMB trigger
rx	Sets run at CMB EXECUTE address
tarm	Specifies which trigger signals arm analyzer
tgout	Specifies whether analyzer drives trigger signals
tx	Enables/disables trace on CMB EXECUTE
x	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.

Many HP 64700 Series emulators support CMB interaction only when configured to use a background monitor. However, the MC68020 and MC68030/EC030 emulators support the use of the CMB when configured with either a background or foreground monitor.

To connect emulators using the CMB, see the *HP 64700 Card Cage Installation/Service Guide.*

To start a simultaneous program run on two emulators

1 Enable the CMB on each emulator using the **cmb -e** command.

2 Reset each emulator using the **rst** command.

3 Set the run address for the first emulator by typing: **rx <address>**

4 Set the run address for the second emulator by typing: **rx <address>**

5 Start program execution on both emulators by typing: **x**

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 add the **tx -e** command to enable trace on execute for each emulator. When the EXECUTE signal is received, both emulators will begin running as specified by the **rx** command, and will start a trace according to the given trigger specification.

To trigger one emulation-bus analyzer with another

1 Enable the CMB on each emulator using the **cmb -e** command.

2 Reset each emulator using the **rst** 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 **t** command.

6 Start a run on each emulator using the **r** command.

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.*

In the above procedure, you set one emulation 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 that you have two MC68020 emulators, running out-of-circuit. The demo program is loaded in each emulator. The following example will trigger the analyzers in both emulators when Cmd_I is detected in the first emulator.

Set up the first emulator by entering the commands:

```
R>init -c
R>demo
R>cmb -e
R>tg addr=~3&handle_msg:Cmd_I
R>tgout trig1
R>cmbt -r trig1
R>tp c
R>t
```

To trigger one emulation-bus analyzer with another

```
R>b
M>
```

Set up the second emulator:

```
R>init -c
R>demo
R>cmb -e
R>cmbt -d trig1
R>tarm =trig1
R>tg arm
R>tp c
R>t
R>r
W>
```

Start the first emulator:

```
M>r
U>
```

On the second emulator, press <**Enter**>. Note that the second emulator is now running:

```
W>
U>
```

On the first emulator, type:

```
U>m -db Cmd_Input=44
```

Display the trace on the first emulator:

```
U>tl -e -10..10
```

Display the trace on the second emulator:

```
U>tl -e -10..10
```

To break to the monitor on an analyzer trigger signal

1 Set the analyzer to drive a trigger signal by typing: **tgout <signal>**

where **<signal>** is either **trig1** or **trig2**.

2 Set the emulator to break to monitor on receipt of the same trigger signal by typing: **bc -e <signal>**

where **<signal>** is the same one specified in step 1.

3 Specify the trigger conditions for the trace. (See Chapter 5, “Using the Analyzer,” for more information).

4 Start the trace by typing: **t**

5 Start a run by typing: **r**

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, and 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. See the **bnct** and **cmbt** commands in Chapter 10, “Emulator Commands.”

7

Configuring the Emulator

How to adapt the emulator to your system

Each MC68020/MC68030/MC68EC030 system differs in the way it configures the processor, uses memory, and memory mapped I/O devices. During system development, your needs for emulator resources may change as your system design matures. You can allocate emulator resources using Terminal Interface commands. This resource allocation is called the emulator configuration.

Memory

The emulator must know how your system memory resources are allocated. You can use emulation memory for some memory ranges. This is useful in the early stages of system debugging.

In the MC68020 emulator, if your target system runs at more than 25 MHz, emulation memory requires one wait state (except for the 4 Kbytes of dual-port memory, which will run at 33 MHz without wait states).

In the MC68030/MC68EC030 emulator, emulation memory always requires one wait state for synchronous and burst modes. If your target system runs at more than 25 MHz, all synchronous and burst modes, and memory accesses will require one wait state.

The emulator also allows you to specify the function code used when loading memory.

You can choose whether to interlock your system $\overline{\text{DSACK}}$ signals (and $\overline{\text{STERM}}$ for the MC68030/MC68EC030) with internal memory terminations for emulation memory cycles and monitor bus cycles (foreground only on the MC68030/MC68EC030). The interlock can be enabled for only the blocks that require it.

Emulation Monitor

The emulation monitor is used to implement some emulator features. For example, display or modification of your target system memory or emulation memory that isn't dual-port is done by the monitor. 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* for more information on foreground and background monitors.)

If you're using the MC68020 emulator with the background monitor, the emulator makes the background cycles visible to your target system. These cycles appear in a 4 Kbyte range that begins with the base address you set for the monitor. The

MC68030/MC68EC030 emulator doesn't make background cycles visible to your system. You can set a "keep-alive address" from which the background monitor will periodically read a byte visible to your system during monitor operation.

If you select a foreground monitor, you can choose the default monitor that is resident in the emulator, or you can design a custom foreground monitor that supports special target system needs. You can specify the interrupt priority mask to use during foreground monitor execution.

A foreground monitor must be used when the MMU of the MC68030 is enabled. If the background monitor is selected when you attempt to enable the MMU, the foreground monitor will be selected, by default.

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.

Break Conditions

Software and hardware breakpoints allow you to terminate your program and enter the monitor.

Software breakpoints use one of the BKPT instructions (BKPT 1..7). You can choose which instruction is used.

Other Configuration Items

The emulator allows you to restrict commands to those that won't temporarily interrupt user execution to perform monitor functions. This is important for some systems that require non-stop, real-time code execution.

You can disable the processor cache memory. 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 cache, all instructions and data are fetched from memory, and therefore will appear in the trace list.

You can block your target system interrupts from the processor. This can help you troubleshoot problems with spurious interrupts or allow you to delay testing of interrupt service routines.

Mapping and Configuring Memory

Every system allocates memory and I/O differently, 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 support external memory, but a change in application definition may need more program code, requiring external memory. While the design is being changed, you can develop the program using the emulator's emulation memory to simulate your target system memory.

The emulator has flexible memory resources that allow you to configure the emulator to support your needs.

To assign memory map terms

- Assign memory to a specific address range by typing:
  map <range> <memory_type> <attribute>

<range> is an address range aligned on 256-byte boundaries (resolution is 256 bytes).

<memory_type> is as follows:

Type value	Memory Assigned
eram	Emulation RAM
erom	Emulation ROM
tram	Target System RAM
trom	Target System ROM
grd	Guarded memory

<**attribute**> can be:

dp to indicate that this block is to reside in the special 4-Kbyte block of dual-ported emulation memory. (Dual-ported memory can be accessed by the host controller without the emulation monitor program, which means that your program executes uninterrupted during the access.)

dsi indicates that your system and emulation $\overline{\text{DSACK}}$s should be interlocked. (The MC68030/MC68EC030 emulator also interlocks the $\overline{\text{STERM}}$ signals when you choose **dsi**.)

ci asserts the $\overline{\text{CIIN}}$ line to the MC68030/MC68EC030 for all addresses in this memory block, indicating that accesses to this block should not be cached. This attribute is available only on the MC68030/MC68EC030 emulator.

(Combine multiple attributes by separating them with commas, for example: **dp,dsi**.)

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 your target system memory.
- Reserve emulation memory blocks.
- Set the type of memory blocks so that configuration items such as write to ROM break will operate correctly.

The MC68020 and MC68030/MC68EC030 emulators have seven map terms. Your address specifications must begin and end on 256-byte boundaries. 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 of memory needed by the absolute address range to be stored.

There is one 4 Kbyte block of dual-ported emulation memory. (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 a foreground monitor, the emulator reserves this block for the monitor code.

If you specify an address range less than 4K with the **dp** attribute, all 4K is allocated because that is the minimum block size for that memory. If you specify a

block size less than 4K and the dual-port memory is unmapped, the emulator uses that memory to more closely match the requested address range to the block size.

In the MC68020 emulator, the dual-port memory does not require wait states, even when you use the emulator at 33 MHz. Also, the dual-port memory is 16 bits wide.

In the MC68030/MC68EC030 emulator, the dual-port memory runs at maximum emulator speed. The dual-port memory in this emulator is 32 bits wide.

There are also two memory sockets on the probe. This memory is not dual-ported; the monitor is used to read and write the locations when you display or modify memory. The bus width for this memory is 32 bits. You can install 256 Kbyte or 1 Mbyte emulator memory modules in these sockets for the following configurations.

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 generally not recommended; it does not allocate blocks as well as Installation 3.
** If you look down at the component side of the probe with the cables leading towards you, memory slot 0 is to your left and memory slot 1 is to your right.

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 memory usage and install memory in the way that will maximize block usage. (See the examples on the next page.)

If you specify the **dsi** attribute, the emulator waits for both the emulation memory data to become valid and your system DSACK 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 DSACK in the address range mapped to emulation memory, don't use the **dsi** attribute, because the system will hang while waiting for your DSACK. (See "To Interlock Emulator and Target DSACKs for Monitor Cycles" for more information.) For the MC68030/MC68EC030 emulator, your system STERM signal is also used for cycle termination if you specify the **dsi** attribute.

If you don't specify the **dsi** attribute when you map a memory block, your target system DSACK and BERR signals (and STERM for the MC68030/MC68EC030) are ignored on accesses to that block.

If you specify the **ci** attribute (MC68030/MC68EC030 emulator only), the $\overline{\text{CIIN}}$ (cache inhibit input) 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 **cf ce** 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.

Some commands reset the memory mapper. These commands are: **map -d ***, **cf mon**, **cf monaddr**. You should configure the emulator before you map memory. Otherwise, you will need to reenter the map commands.

Example

Suppose 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 system from 0 through ffff hex. Also, you want to use the dual-port emulation memory at 20000 hex:

```
R>map 1c000..1c0ff tram

R>map 0..0ffff trom

R>map 20000..20fff eram dp
```

Remember that you must use the background monitor if you want to use the dual-port emulation memory to store your program.

The relationship between memory ranges and the block sizes of memory is easier to understand by looking at an example. Suppose you have Installation 1 from the table above. Then you enter the following map commands:

```
R>map 0..7fff eram
R>map 20000..3f0ff eram
R>map 40000..4ffff eram
R>map 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 you defined (the range from 50000..500ff). The entire 4 Kbyte block is reserved even though you specified only a 256-byte range. Two 64K blocks and one 128K block are used from the other emulation memory, leaving two 64K blocks and one 128K block. One of the 64K blocks is used for the first map term, but 32K of that block is unused and unavailable. The third term uses the other 64K block. The second term uses part of the 128K block, leaving the rest unavailable.

The mapper's resolution is independent of the block allocation. In the above example, if you had **map other grd** and your program accessed 8000h, the emulator would do a break on access to guarded memory.

Combinations of regular emulation memory and dual-port emulation memory may be confusing when you look at analysis displays. Assume that you have installation 3 from the table. Suppose you reset the map, and then mapped a range covering 260 Kbytes:

```
R>map 0..40fff eram
```

The emulator will allocate one 256K block from the emulation memory modules and will use the 4-Kbyte, dual-port memory for the rest of the range. Only one mapper term is created (without the **dp** attribute). In the MC68020, because the emulation memory modules are 32 bits wide and the dual-port memory is 16 bits wide, you will see a change from 32-bit to 16-bit fetches as the processor crosses the boundary between the two memory types.

You can use function codes when mapping memory. See “To Set the Function Code for Memory Loads” for examples.

To assign the memory map default

- To map all remaining memory to your target system RAM, type: **map other tram**
- To map all remaining memory to your target system ROM, type: **map other trom**
- To assign all remaining memory as guarded memory space, type: **map other grd**

The other map term specifies all address ranges not otherwise covered by existing memory map terms. This can save you time in memory mapping.

Often you will want to be notified when the processor accesses a nonexistent memory location during a program run. Use the **grd** (guarded) term to do this. The emulator will break to monitor and display a message when a guarded memory access occurs.

To check the memory map

- To check the current memory map, type: **map**

To delete memory map terms

- To delete a particular memory map term, type: **map -d <term#>**

  Where **<term#>** is in the range 1-8.

- To remove all memory map terms and reset the map, type: **map -d ***

To enable one wait state for emulation memory (MC68020)

- If your system clock frequency is greater than 25 MHz, type: **cf emwait=en**

  Otherwise, type: **cf emwait=dis**

- To check the memory wait state setting, type:
  cf emwait

  In the MC68020 emulator, emulation memory doesn't require any wait states for clock speeds under 25 MHz. For clock speeds above 25 MHz, you must enable this configuration item, which adds one wait state to all emulation memory accesses (except for those to the dual-port emulation memory, which will run at 33 MHz without wait states). This ensures that emulation memory has enough time to respond to the memory access. Otherwise, emulator operation will be erratic.

To enable one wait state for synchronous/burst accesses (MC68030/MC68EC030)

- If your target system clock frequency is greater than 25 MHz, type: **cf emwait=en**

  Otherwise, type: **cf emwait=dis**

- To check the memory wait state setting, type:
  cf emwait

  The MC68030/MC68EC030 emulator always requires one wait state for synchronous and burst memory accesses to emulation memory. When the clock speed is above 25 MHz, the emulator *must* add a wait state for synchronous and burst mode accesses to your target system memory. You do this by selecting **cf emwait=en**. Otherwise, emulator operation will be erratic. No wait states are required for synchronous and burst mode accesses when the clock rate is below 25 MHz.

To set the function code for memory loads

- To set the function code for memory loads that use the load command, type: **cf lfc=<FC>**

  where **<FC>** may be one of:

  u—User
  s—Supervisor
  d—Data
  p—Program
  ud—User Data
  up—User Program
  sd—Supervisor Data
  sp—Supervisor Program
  x—Don’t Care

- To check the function code setting for memory loads, type: **cf lfc**

  If you map memory ranges using function codes, you must use this configuration item to set the function code type before you load programs using the load command. This ensures that the program is loaded into the correct function code space.

Example

Suppose that you have two programs, one supervisor and the other user. You might map the following memory ranges:

```
R>map 1000..1fff@s eram

R>map 1000..1fff@u eram
```

Now, if the programs are named super.X and user.X (HP64000 absolute format), you would load them as follows:

```
R>cf lfc=s

R>load -hbs "transfer -tb super.X"

R>cf lfc=u

R>load -hbs "transfer -tb user.X"
```

(See “To Load a Program” in Chapter 1 for information on the load command.)

To enable the processor memory management unit (MC68030 only)

- To turn on the MMU in the MC68030 emulation processor, enter: **cf mmu=en**

Once enabled, the MMU of the MC68030 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 of the MC68030 will manage memory. The MMU of the MC68030 must be enabled by this configuration question before the operating system can establish those control values.

Your target system will enable the MMU during program execution by using the /MMUDIS signal. If you disable the MMU with this configuration question, the /MMUDIS signal from your target system will be ignored.

A foreground monitor must be used when the MMU of the MC68030 is enabled. If the background monitor is selected when you type in the **cf mmu=en** command, the foreground monitor will be selected automatically.

Note

Make sure the foreground monitor is mapped to memory space that has a 1:1 translation. Refer to the end of this chapter for instructions on how to map the foreground monitor to 1:1 address space when using the MC68030 MMU.

Examples

To enable the MC68030 MMU so that the operating system can set it up to manage memory, enter the command:

M>**cf mmu=en**

To disable the MC68030 MMU, enter the command:

M>**cf mmu=dis**

To see the present state of the MMU, enter the command:

M>**cf mmu**

To obtain additional information about the MMU, enter the command:

M>**help cf mmu**

To select the emulation monitor

- To select the background monitor, type: **cf mon=bg**
- To select the foreground monitor, type: **cf mon=fg**
- To see which monitor is currently selected, type: **cf mon**

The emulation monitor is used to perform emulation functions such as display and modification of your target system memory, emulation memory that is not dual-port, and processor resources such as registers.

The background monitor overlays processor address space and doesn't use any processor memory resources. However, interrupts are disabled (including level 7) when the emulator is running in background. This condition may not be tolerable to some target system designs.

If you select the foreground monitor, interrupts can be enabled during monitor execution, which may make the emulator more transparent in some applications. See "To Set Foreground Monitor Interrupt Priority" in this chapter.

A foreground monitor must be used when the MMU of the MC68030 is enabled. If the background monitor is selected when you enable the MMU, the foreground monitor will be selected, by default, and a STATUS message will appear to tell you that the monitor type has changed.

When you select the foreground monitor, the emulator maps the 4 Kbyte block of dual-port memory for the monitor. You can't use any portion of the range allocated to the foreground monitor for any other purpose.

You may have two types of foreground monitors: one default, and the other custom. The default foreground monitor is resident in the emulator, and is automatically loaded whenever the processor leaves emulation reset. If this monitor doesn't meet your needs, you can modify the monitor source code (supplied with the emulator) to create a custom monitor, and load it using the **-f** option to the **load** command.

When you select a different monitor, the memory map (and the emulation processor) is reset. First select the monitor type. Then map memory.

If you have trouble with emulation monitor functions, you can reload the monitor. Both the foreground and background monitors are loaded when the processor transitions out of emulator reset.

The MC68020 and MC68030/MC68EC030 emulators differ slightly in the implementation of the monitor configuration. The following table summarizes the differences.

Emulator	Background monitor (cf mon=bg)	Foreground monitor (cf mon=fg)
MC68020	**monaddr**—sets address block for memory overlay; bus cycles to this overlay are driven to target **mondsi**—interlocks emulation and monitor $\overline{\text{DSACK}}$s for monitor cycles **monintr**—ignored for background monitor	**monaddr**—sets address block for monitor in emulation memory **mondsi**—interlocks emulation and monitor $\overline{\text{DSACK}}$s for monitor cycles **monintr**—allows lowering interrupt mask during foreground monitor execution
MC68030/MC68EC030	**monaddr, mondsi, monintr**—not available **monkaa**—set address from which to read a byte periodically during background operation	**monaddr**—sets address block for monitor in emulation memory **mondsi**—interlocks emulation and monitor $\overline{\text{DSACK}}$s for monitor cycles **monintr**—allows lowering interrupt mask during foreground monitor execution **monkaa**—not available

More information on emulation monitors is given in the book *Concepts of Emulation and Analysis.*

To set the monitor base address

- To set the base address for the monitor, type:
  cf monaddr=<ADDRESS>

  where **<ADDRESS>** is a hexadecimal address on a 4 Kbyte boundary (XXXXX000h).

- To check the monitor base address, type: **cf monaddr**

Background monitor

When you select the background monitor, the emulator uses overlay memory to load the monitor. This overlay memory doesn't use any processor memory space.

In the MC68020 emulator, the address, data and control strobes are driven to your target system during background monitor operation. Background write cycles appear as reads to your system. You can relocate the background monitor (using the **monaddr** configuration item) so that these read cycles won't interfere with I/O or other target system hardware.

In the MC68030/MC68EC030 emulator, the **monaddr** configuration item isn't available. Background monitor bus cycles aren't driven to your target system. The **monkaa** (monitor keep-alive address) configuration item is provided to support systems that require periodic bus activity. See "To set the background monitor keep-alive address."

Foreground monitor

For both the MC68020 and MC68030/MC68EC030 emulators, this configuration item sets the base address where the monitor is loaded. When you select the foreground monitor, the emulator uses the 4-Kbyte block of dual-port emulation memory to load the monitor. It resets the memory map, then creates a map term at the address you specify for **monaddr**. You can't delete or alter this map term by using the **map** command. Instead, you must change the monitor configuration by using the **mon**, **monaddr**, and **mondsi** configuration items.

If the memory management feature of the MC68030 emulator is enabled, be sure the foreground monitor is mapped 1:1. Refer to the end of this chapter for instructions on how to map the foreground monitor to 1:1 address space.

To interlock monitor cycles with your cycle termination signals

- To interlock the emulator and your target system cycle termination signals for monitor accesses, type: **cf mondsi=en**
- To terminate monitor accesses with only the emulator-generated cycle termination signals, type: **cf mondsi=dis**
- To check the interlock setting for monitor accesses, type: **cf mondsi**

When you enable interlocking, emulation monitor cycles aren't terminated until your target system DSACK (or STERM for MC68030/MC68EC030) is received. Otherwise, the emulator-generated signals will terminate the cycle. Also, the emulator will respond to BERR signals from your system when interlocking is enabled.

This configuration item applies to both the background and foreground monitors on the MC68020 emulator. In the MC68030/MC68EC030 emulator, this configuration item is available only with the foreground monitor (**cf mon=fg**).

This configuration item affects the map term defined for the monitor.

If you enable the interlock (**cf mondsi=en**), and the monitor is in an address range where yourtarget system does not return DSACK (or STERM), the emulator will stop functioning. You will see a w> prompt indicating that the CPU is in a wait state. Use the **rst** command to reset the processor, and then disable the interlock.

If you set **cf mondsi=dis**, your DSACK and BERR signals (and STERM for the MC68030/MC68EC030) are ignored during monitor accesses.

If you disable the interlock, remember that the emulator bus cycles will be visible to your system during monitor operation (except when you use the

MC68030/MC68EC030 background monitor). This may cause a problem if your target system wasn't expecting these cycles. Erratic system operation may result.

To set foreground monitor interrupt priority

- To select the interrupt priority level for general foreground monitor execution, type: **cf monintr=<level>**

  where **<level>** is in the range 0..7.

- To check the interrupt priority level, type: **cf monintr**

  During background monitor execution, interrupts are always disabled. This may cause problems for some systems, especially those for real-time control where interrupt servicing must be done immediately.

  To solve this problem, you can select the foreground monitor, and then set the interrupt priority level to one that allows your system to function correctly, yet avoids excessive interrupt processing.

  On the MC68020 emulator, this configuration item is ignored if you choose the background monitor. On the MC68030/MC68EC030 emulator, this configuration item isn't visible if you choose the background monitor. Both emulators block your target system interrupts if you set **cf ti=dis**. See "To Disable Target System Interrupts." The emulator is reset when you change the setting of this configuration item.

  At monitor entry, if the processor's interrupt priority level was greater than the value set by **monintr**, the monitor uses the previous priority level (not **monintr**). Otherwise, the priority is lowered to the **monintr** level.

  The foreground monitor only lowers the interrupt priority to the level you specify when it is not executing critical code (for example, on monitor entry and exit). Otherwise, all interrupts are disabled.

Examples

Suppose your system has a disk device driver that uses interrupt level 5, and the service routine must be run to prevent system damage. To allow interrupts of higher priority than level 4 to be serviced during foreground monitor execution, enter:

```
R>cf monintr=4
```

To set the background monitor keep-alive address (MC68030/MC68EC030)

- To enable the background monitor keep-alive function, type:
  cf monkaa=<ADDRESS>

  where **<ADDRESS>** is a hexadecimal address with an optional function code (0XXXXXXXXh@fc).

- To disable the background monitor keep-alive function, type: **cf monkaa=none**

- To check the setting of the background monitor keep-alive function, type:
  cf monkaa

  The MC68030/MC68EC030 emulator implements the background monitor in a different manner than the MC68020 emulator. The MC68020 emulator drives all background cycles to your target system. The **cf monaddr** configuration item sets the address block for those bus cycles.

  The MC68030/MC68EC030 emulator does not drive background monitor cycles to your target system. Some target systems cannot tolerate this. For example, your system may have a watchdog timer that will time out if the background keep-alive address isn't read periodically, or a block of dynamic RAM in your system may need to be refreshed.

  In these situations, you set the **cf monkaa** 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.

  This configuration item does not appear in the MC68020 emulator.

Example

To select the background monitor and have it periodically read a byte from address ffff hex in user space, enter the commands:

```
R>cf mon=bg
R>cf monkaa=0000ffff@u
```

To preset the interrupt stack pointer and PC

- To set the initial interrupt stack pointer and the initial program counter when the emulator enters the monitor from reset, type: **cf rv=<RESETISP>,<RESETPC>**

  where **<RESETISP>** and **<RESETPC>** are both 32-bit address values in hexadecimal. Both values must be even. These values usually should correspond to the values loaded into offsets 0 and 4 of your vector table.

- To check the reset interrupt stack pointer and pc settings, type: **cf rv**

  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 either entering the command **r rst** or by entering the command **r** at the R> prompt.)

  However, if you reset the emulator, break to the monitor, and then run the emulator, the stack pointer and program counter values have not been initialized. Therefore, the run will fail.

  The **cf rv** configuration item is 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 without errors. (You can accomplish the same thing by using the **reg** command to set the PC and ISP values while in the monitor.)

  You should set the interrupt stack pointer to the value normally contained at offset 0 of your vector table. The program counter should be set 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. Set the interrupt stack pointer to 7ff0 and the initial PC to 400h:

```
R>cf rv=7ff0,400
```

If you now use the **b** command to break to the monitor, the isp is set to 7ff0 and the pc is set to 400.

Defining Break Conditions

The emulator supports software and hardware breakpoints. These breakpoints allow you to break the processor to the monitor when a specified event occurs, such as reaching a particular address or completing a trace measurement. The configuration items and commands shown in this section allow you to control the conditions that will cause an emulator break. Also, the "restrict to real time runs" configuration item allows you to disable breaks for most conditions, ensuring continuous target program executions.

To define the software breakpoint vector

- To specify the BKPT vector to use for software breakpoints, type: **cf sw=<number>**
- To check the current BKPT vector, type: **cf sw**

<number> can be any value in the range **1..7**. Breakpoint value 0 is not supported by the hardware. The default setting is **7**. You should choose a vector that isn't used by your target system for other purposes.

The emulator uses the BKPT instruction to implement software breakpoints. When you define a breakpoint using the **bp <address>** or **bp -e <address>** commands, the emulator saves the instruction at the specified address. Then it writes the BKPT instruction at that address.

When the BKPT instruction is encountered during user program execution, the processor executes a breakpoint acknowledge cycle; then puts the BKPT vector on the address bus. The emulator sees this and jams a monitor entry vector onto the data bus during the vector fetch. The processor thus enters the emulation monitor.

The monitor replaces the BKPT instruction with the instruction saved earlier, and clears the breakpoint status.

If you change this configuration item, any active breakpoints in the emulator's breakpoint table are disabled.

To enable or disable break conditions

- To enable a break condition, type: **bc -e <cond>**
- To disable a break condition, type: **bc -d <cond>**
- To check the settings of the break conditions, type: **bc**

<cond> can be any of the following: **rom**, **bp**, **bnct**, **cmbt**, **trig1**, **trig2**. All break conditions are disabled when the emulator is initialized.

You can choose to enable or disable hardware and software breakpoints. For example, you might want to disable the write to ROM break temporarily. (Perhaps you need to see the next few bus cycles after the write in the trace list.) Or you might want to enable a break when the analyzer finds its trigger condition.

Examples

To enable software breakpoints, enter:

```
R>bc -e bp
```

To disable software breakpoints and write to ROM break:

```
R>bc -d bp rom
```

To generate an analyzer trigger signal when the analyzer finds its trigger, and break to the monitor when the trigger is generated:

```
R>tgout trig1
R>bc -e trig1
```

Setting Other Configuration Items

The emulator has a few miscellaneous configuration items:

- Restrict the emulator to real-time runs.
- Disable the processor's cache memory.
- Disable target system interrupts.

To restrict to real-time runs

- To restrict the emulator to real-time runs, type: **cf rrt=en**
- To enable all emulator functions, type: **cf rrt=dis**
- To check the current setting of the real-time runs configuration item, type: **cf rrt**

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 system program. This may cause problems in systems that need real-time program execution.

If you set the **cf rrt** configuration item to **en**, the emulator will stop running user code only with the **rst** (reset), **b** (break), **r** (run), and **s** (step) commands. Commands such as **reg** (registers) that require a break to monitor are rejected. Also, the **m** (memory) command will be rejected if the address argument specifies standard emulation memory (not dual-ported) or your target system memory.

This configuration item doesn't 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 set this configuration item to **dis**, all commands are accepted.

To disable the processor cache memory

- To disable the processor cache(s), type: **cf ce=dis**
- To enable the processor cache(s), type: **cf ce=en**
- To check the cache enable/disable setting, type: **cf ce**

The MC68020 processor has a cache that stores the most recently used instructions. The MC68030/EC030 processor adds a cache for recently used data.

The cache memory increases processor performance, but the emulation-bus analyzer can't trace processor accesses to the internal cache. This may cause confusing trace displays or failure to trigger, especially if the code is a small loop where all the instructions and operands fit into cache and registers.

When you disable the cache(s), the processor will always access external memory. Then the analyzer will see all bus cycles, which will improve the trace list, but processor performance will be reduced.

When you're more concerned about measuring processor performance, you should enable the cache(s). If you are making analyzer measurements at the same time, you may need to experiment to find suitable trigger combinations.

This configuration item disables the on-chip cache(s) by controlling the $\overline{\text{CDIS}}$ signal. If disabled, the emulator will assert the $\overline{\text{CDIS}}$ signal to prevent your target system from enabling the cache(s). If **cf ce** is enabled, the $\overline{\text{CDIS}}$ signal from your target system and the cache control register (CACR) enable bit determine whether the cache is enabled.

If you need to disable caching only for accesses to a specific memory block, use the **ci** memory map attribute (available only on the MC68030/MC68EC030 emulator). 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 disable your system interrupts

- Disable your system interrupts by typing: **cf ti=dis**
- Enable your system interrupts by typing: **cf ti=en**
- Check the interrupt enable setting by typing: **cf ti**

You may want to disable system interrupts if your 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 routines.

Your system interrupts are always disabled during background monitor execution. The foreground monitor blocks all interrupts during certain critical routines such as monitor entry.

The foreground monitor can be configured to lower the interrupt priority mask after monitor entry. See "To Set the Foreground Monitor Interrupt Priority."

Mapping The Foreground Monitor For Use With The MC68030 MMU

To use the memory management feature of the MC68030 emulator, you have to use a foreground monitor that is mapped 1:1 (logical address = physical address). The reason that 1:1 monitor mapping is important is that the MC68030 MMU may be enabled or disabled at any time by your target system during execution of your program; whether or not the MMU is enabled, the emulator must be able to enter the foreground monitor to provide emulation features. There are two ways to map the address range 1:1 where the foreground monitor is located:

- Modify the mapping tables in the MMU to maintain a 1:1 mapping of the memory address space where the foreground monitor is located. Make sure the monitor mapping is not write protected.

- Use one of the two transparent translation registers (TT0 or TT1) to control the block where the foreground monitor is located. You must remember to set the Read/Write Mask bit (RWM) to 1. Transparent translation registers can be set to translate only read accesses or only write accesses. To use a transparent translation register to control the address space of the foreground monitor, both read and write accesses must be enabled (by ignoring the R/W bit).

To modify the MMU mappings to translate the monitor address space 1:1

- In the operating system that sets up the MMU for your target program, set aside a 4-Kbyte address space to contain the foreground monitor.

  or

- After the operating system for your target program has set up the MMU, but before you enable the MMU, modify the MMU translation tables to ensure that the foreground monitor resides in logical address space that will be translated 1:1 to physical address space.

  When you modify the content of any MMU mapping table, remember that the tables are located in physical address space. You must enter your modification commands using physical addresses.

  If you are mapping page sizes smaller than 4 Kbytes through the MMU mapping tables, you will need to ensure 1:1 translations for all of the pages that contain portions of the emulation foreground monitor.

  Select an address space to contain your foreground monitor that is higher than the address space used for your target program and I/O. This will optimize deMMUer resources by using them first to reverse-translate your address space.

Examples

To map the foreground monitor to address space that is translated 1:1 by the MMU, beginning at 24000H:

R>mmu 24000..24fff

The above command causes the emulator to show the present logical-to-physical mapping of the address range occupied by the monitor. Note that the monitor's base address (24000 in this example) must be on a 4 Kbyte boundary (hexadecimal number ending in 000H).

R>mmu -t 24000

The above command shows the details of the path address 24000 takes through the MMU mapping tables to reach its corresponding physical address. Place

appropriate values in the path through the mapping tables to ensure that the page containing logical address 24000 is mapped to physical space beginning at address 24000. If using a page size of 4 Kbytes or greater, you only need to map a single page for the foreground monitor. If using a smaller page size, you will have to map multiple pages.

To modify a transparent translation register to map the monitor address space 1:1

- Modify the value of a transparent translation register to the base address you specified for the foreground monitor, or the first address within the range to be occupied by the foreground monitor:
  reg <transparent translation register>=<address>

  Where **<transparent translation register>** may be TT0 or TT1, and **<address>** must begin on a 4 Kbyte boundary (hexadecimal number ending in 000H)

Examples

To map the foreground monitor to 1:1 address space beginning at 0800 0000H by using TT0, enter the commands:

R>cf monaddr=08000000

R>reg tt0=08008777

This will map 8000000..8ffffff transparently (1:1). This is a 16M block, the smallest that can be specified in a transparent translation register.

8

Solving Problems

What to do when the emulator doesn’t respond 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 problems.

If you see unexplained states in the trace list

- ☐ Check to see that the sequence, storage and trigger specifications are set up to exclude the states that you don't need.
- ☐ Try using the **tl <instruction_state> <operand_state>** command to inform the dequeuer which operand state belongs with the named instruction state.
- ☐ Try using the **-ol** option to the **tl** 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 **ci** (cache inhibit) attribute to those ranges in the memory map. (See Chapter 7, "Configuring the Emulator.")

If the analyzer won't trigger

- ☐ Instruction fetches from cache memory aren't visible to the analyzer. You can disable the cache while using the analyzer by entering the **cf ce=dis** command. Reenable the cache to improve performance when you are finished using the analyzer.
- ☐ When the MC68030/MC68EC030 fetches instructions from program memory, it addresses 32-bit longwords. These addresses are always multiples of 4 (ending in 0h, 4h, 8h, and Ch), except for PC-relative fetches. The instruction you are trying to trigger on may be in the high word or the low word of the long word. If you specify trigger on a symbolic address without knowing whether that symbol is in

the high word or low word, the address may not appear on the address bus. If you think this may be the problem, try specifying your trigger symbol as "<symbol>-2H". This long-word correction is not necessary when you are trying to trigger on data fetches; data is always fetched from the absolute address of the data location.

If the emulator won't work in your system

☐ The emulator must be plugged into a powered-on target system with a clock signal to function correctly. (Apply power to the emulator before applying power to your target system.)

If you suspect that the emulator is broken

1 Shut off power to your target system, and then the emulator.

2 Disconnect the emulator from your target system.

3 Connect the emulator to the demo board. Also connect the power cable from the emulator to the demo board (See Chapter 15, "Installation and Service").

4 Set all the demo board configuration switches to the TEST position (See Chapter 15).

5 Apply power to the emulator.

6 Type **pv 1** to run performance verification.

If either the emulator or analyzer fail the performance verification, check the installation of those modules. See Chapter 15 for details. 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 emulation memory module installations and the use of the dual-port memory, it's possible that a 256 byte map request may use 512 Kbytes. (The map term will be only 256 bytes.) Most systems won't have such differences between memory block size requirements and available memory. However, certain emulation memory module installations will aggravate the problem.

- Also, use of the dual-port memory is controlled first by monitor selection and next by explicit selection of a dual-port term in the map. If you choose a foreground monitor, the dual-port memory block is reserved for 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 another 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 since 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.)

- On the MC68020 emulator, if dual-port memory is used with the emulation memory module to satisfy a map request, you may see unusual behavior in the trace list. The processor executes correctly, but you will notice a status change from 32 bit accesses (ds32) to 16-bit accesses (ds16) as the execution crosses the boundary from the emulation memory module to the dual-port memory. (You won't see this problem on the MC68030/MC68EC030 emulator because the dual-port memory is 32 bits wide.) To fix this situation, map the dual-port memory explicitly to another unused range (use the **dp** attribute). Then the request for a 260-Kbyte block will use memory from the emulation memory module only.

  See the section "Mapping and Configuring Memory" in Chapter 7 for more information on memory allocation.

If the demo program won't work

- ☐ Check to be sure that you have the emulator plugged into the demo board, with power connected to the demo board from the emulator. (The demo program will not work with other target systems.)
- ☐ Make sure that the switches on the demo board set to the OCE position (Out-of-Circuit Emulation, away from TEST).
- ☐ Make sure you initialized the emulator, then executed the demo command to load the program and configure the emulator.

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

- ☐ Check to make sure that your system uses an open-collector driver to drive the processor $\overline{\text{RESET}}$ line (this also applies to $\overline{\text{HALT}}$ on the MC68020 processor). This line is bidirectional; the processor must be able to pull the line low.
- ☐ Check to make sure that all critical components in your system are reset by the $\overline{\text{RESET}}$ signal to the processor. System startup problems can arise when this rule is violated. Suppose that your system reset circuit drives several critical system components directly, but drives the processor through an open-collector buffer. Suppose also that the critical components are memory-mapping circuits that locate ROM containing the vector table at address zero for startup, then move it to a high address range after system initialization. An emulator reset system can't drive this

separate reset line. Therefore, a run after emulation reset will fail, because the vector table is not located in the correct place.

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 **dmmu -lv** 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.

Read "Using the deMMUer" in Chapter 9 for ways to make more efficient use of deMMUer resources.

If verbose mode shows less than eight mappings but the deMMUer is "out of resources"

☐ Check if you are using both root pointers in your memory mapping scheme? The deMMUer may have run out of resources for only one of the root pointers.

☐ Check if you are using function-code mapping. The deMMUer may have run out of resources for only one of the function-code memory spaces.

- ☐ Read "Using the deMMUer" in Chapter 9 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: **dmmu -e**.
- ☐ Check to see if you loaded the deMMUer with the information needed to reverse translations made by the MMU with the command: **dmmu -lv**.
- ☐ Read "Using the deMMUer" in Chapter 9 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 and cpu space.

- ☐ Check to see which physical memory spaces are being reverse translated by the deMMUer. Enter the **dmmu -lv** 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 **dmmu -l** 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 **dmmu -l** 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 **dmmu -l** command. Issue the **dmmu -l** command again.

  Read "Using the deMMUer" in Chapter 9 to understand how the deMMUer selects the physical addresses it will translate.

If you can’t break into the monitor after you enable the MC68030 MMU

☐ Enter the command: **rst -m**. If your MC68030 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.

☐ If you are mapping a page size that is smaller than 4 Kbytes, make sure that you have provided 1:1 address mappings for all of the pages that contain monitor code.

☐ Refer to the end of the chapter titled, "Using 68030 Memory Management" for a detailed example that discusses how to solve a "can’t break into monitor" problem.

Part 3

Reference Information

Commands and Expressions

The Terminal Interface command set is a complete operating environment for the emulator. The command interpreter includes a rich expression-handler that allows you to specify measurement values in terms that make sense in the domain of the problem.

In This Part

Chapter 9, "Using 68030 Memory Management," explains how the emulator supports development of a virtual memory system. This chapter describes considerations you need to understand when developing a system that uses the MMU of the 68030.

Chapter 10, “Emulator Commands,” lists all the Terminal Interface commands. This chapter describes the syntax and operation of each command and includes examples of command usage.

Chapter 11, “Expressions,” describes the different types of expressions used in Terminal Interface commands.

Chapter 12, “Messages,” lists the error and status messages you may see while operating the Terminal Interface. Each message describes the reason why you got that message and how to recover from the error.

Chapter 13, “Data Formats,” lists the file format for the binary trace list and the symbol files.

Chapter 14, “Specifications,” gives the physical, electrical, environmental, and timing specifications for both the MC68020 and MC68030/EC030 emulators.

If you’re looking for a general introduction to the emulator, see part 1. Part 2 describes how to use the emulator to make measurements.

9

Using 68030 Memory Management

Understanding logical and physical emulation and analysis

Understanding Emulation And Analysis Of The MC68030 Memory Management Unit

You only need to read this chapter if you are using the on-chip MMU (Memory Management Unit) of the MC68030 microprocessor. If you are using an MC68020 or MC68EC030, or if you are using an MC68030 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 MC68030 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 MC68030 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 68030 lets you divide the memory space into one of several page sizes. You can choose to divide the memory into pages as small as 256 bytes or as large as 32 Kbytes). Assume you set up the MMU to divide the memory into 1-Kbyte pages. Your program will occupy 10,000 pages of code, and 2,000 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 logical memory space 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 1 Kbyte of physical memory, and program execution continues.

As pages are swapped back and forth between the program (logical space) 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 (**tl -d**).

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 Function Codes

The MMU lets you use function codes as the first level within the translation tables that map memory. It also allows separate tables to be set up for supervisor and user access. For example, you can create one set of tables to translate addresses in supervisor space and another set of mapping tables to translate addresses in user space. The supervisor space can use the SRP (supervisor root pointer) or the CRP (CPU root pointer), as you choose. The user space must use the CRP. The supervisor memory can begin at supervisor address 0 and the user memory 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 function code is supervisor at the time the access is attempted. Take care to ensure that function codes are specified with addresses if the MMU will be making the distinction (example: <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 $\overline{\text{MMUDIS}}$ signal. When $\overline{\text{MMUDIS}}$ is asserted, the MMU is disabled. When $\overline{\text{MMUDIS}}$ is negated, the MMU is enabled to translate addresses. The emulator controls the $\overline{\text{MMUDIS}}$ line according to the way you set the "mmu" configuration parameter.

If you enter **cf mmu=dis**, the $\overline{\text{MMUDIS}}$ line is held asserted. If you enter **cf mmu=en**, the $\overline{\text{MMUDIS}}$ line is directly controlled by your 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 a value into the translation control register (TC). If the enable bit of the TC register value is "e=1" and the value in the TC register is valid, the MMU will be enabled. If the enable bit in the TC register is "e=0", or if the value loaded into the TC register is invalid, the MMU will be disabled.

Restrictions when using the MC68030 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).

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. 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.

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 PC always contain logical address values, even when the MMU is turned on.

If you enter a "run from address" command (**r <address>**), 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 are always logical addresses. When you set a breakpoint at an address, that address is translated by the MMU and the BKPT #7 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 100h**) will show you the memory content at logical address 100H (which might be a different physical address). If you want to see the content at physical memory address 100H, you will have to enter the command **m 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 access will fail. To avoid this, modify the MMU tables to remove write protection from the memory you want to modify.

Seeing details of the MMU Translations

The following paragraphs discuss emulator displays that help you understand translations made by your MMU. There are three displays, each giving a different level of detail of the MMU translations.

- The present address mappings in your MMU tables.
- The translation table entries for a single logical address.
- The contents of a single level of the translation tables pointed to by a selected logical address.

How the emulator helps you see the details of the MMU mappings

To see all of the logical-to-physical translations presently mapped, enter the command **mmu**. 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:

```
U>mmu
  FUNCTION      LOGICAL ADDRESS       PHYSICAL ADDRESS
    CODE        Lower      Upper      Lower      Upper
  user data         0       7FFF    00000000   00007FFF
                 8000       FFFF    00008000   0000FFFF
                10000      17FFF    00010000   00017FFF
                18000      1FFFF    00018000   0001FFFF

  user prog         0       7FFF    00000000   00007FFF
                 8000       FFFF    00008000   0000FFFF
                10000      17FFF    00010000   00017FFF
                18000      1FFFF    00018000   0001FFFF

  supr data         0       7FFF    00000000   00007FFF
                 8000       FFFF    00008000   0000FFFF
                10000      17FFF    00010000   00017FFF
                18000      1FFFF    00018000   0001FFFF

  supr prog         0       7FFF    00000000   00007FFF
                 8000       FFFF    00008000   0000FFFF
                10000      17FFF    00010000   00017FFF
                18000      1FFFF    00018000   0001FFFF
U>
```

The above listing shows function codes were included in the mapping scheme. If function codes had not been included, the function code headings would not be present, and only a single list of logical-to-physical address mappings would be shown.

Note that the emulator enters the monitor to obtain the information it shows in the MMU displays. Your user 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 a software breakpoint in some safe area of your program code and run until the breakpoint is hit. Then you can safely view the MMU mappings.

The display you get with the **mmu** command can show as little as one line per page of mapped logical address space. Contiguous entries are shown on one line to make the display more readable. Early terminations (which result in contiguous translation of multiple pages) will also be shown on a single display line.

The display of MMU mappings will only show pages for which the system has valid mappings. No information is given in the default **mmu** display for paths designated invalid, or for paths containing illegal entries.

If your system uses a small page size and has a large physical memory, the **mmu** command may cause your display to scroll through a long list of mappings. To avoid a list of mappings that scrolls for a long time, include an address or address range in your **mmu** command. The command **mmu 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 include a function code in your command (if function-code mapping is in use). The command **mmu 0..0ffff@u** will show all of the mappings for addresses from 0 through 0ffff in user address space.

The display does not take into account overrides caused by the transparent translation registers. A status message will indicate that a transparent translation overlapped a mapping. Use the display register (**reg**) command to determine the transparent translation.

Supervisor/user address mappings

If you are using separate supervisor and user mappings, the emulator will support this choice and show appropriate information. The MMU has two ways of doing this:

(1). Using SRP and CRP root pointers. When using the SRP and CPU root pointers to provide separate user and supervisor translation mappings, no distinction is made between program and data space.

- To see only the mappings under the SRP, use the command: **mmu <address>[..[<address>]]@s**. The "@s" tells the emulator to show the supervisor mapping for the associated logical address or address range.
- To see only the mappings under the CRP, use the command: **mmu <address>[..[<address>]]@u**.
- If you do not specify a root pointer in your command in the above example, the mappings will be shown for the SRP root pointer.
- If you specify no address, then mappings will be shown for both root pointers.

(2). Use function code lookup table level. This allows up to seven separate sets of translation tables, each under a different function code value (although only the following four function codes are normally used: user data, user program, supervisor data, and supervisor program).

- To see only the mappings under a selected function code, use a command like: **mmu <address>[..[<address>]]@ud**
  where: **@ud** displays mappings under user data.
  @up displays mappings under user program.
  @sd displays mappings under supervisor data.
  @sp displays mappings under supervisor program.
- The default suffix is **@sp** (supervisor program) when an address is specified in an **mmu** command.

Translation details for a single logical address

To see translation details for a logical address, enter a command such as: **mmu -t <address>. The -t** 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.

```
U>mmu -t 40f8H
  Logical Address (hex)   supr prog     0    0    0    0    4    0    F    8
  Logical Address (bin)      110     0000 0000 0000 0000 0100 0000 1111 1000
  Table Level               FCODE    IIII IIII IIII IIIA APPP PPPP PPPP PPPP

  LEVEL INDEX LOCATION     CONTENTS      TBL/PAGE L/U LIMIT S CI M U WP DT
  CRP                  80000002 00001020 00001020                      SHORT
  FC    0006  00001038     0000104a      00001040                 1 0  SHORT
  A     0000  00001040     00000019      00000000             0  1 1 0  PAGE

  Physical Address (hex) = 000040f8
U>
```

Address mapping details

The example display shows:

- The translation mapping for logical address 40f8H in supervisor program space. Both the hexadecimal and binary values are shown for the logical address.
- The Table Level line shows how each address bit is mapped by the translation control register. In the example display, the 15 high-order bits are ignored by the initial-shift. The next two bits are used as an offset into Table A. The lowest-order 15 bits contain the offset into the physical page indicated by Table A.
- The CRP (CPU root pointer) is used by this address mapping. Its contents point to the function code table whose base address is 1020.
- The index used in the function code table is 6 (supervisor program space) which points to physical address 1038. The content of this address is 104a, indicating a short A level table located at base address 1040. The status also indicates that this table has been used "U".
- The function code table points to Table A whose base address is 1040. The index into Table A is 0. The content of Table A at the indexed location is 19.

This indicates that the base address of this physical page is 0. There are also a number of status flags indicated.

- The physical address is finally calculated by adding the physical page offset to the base address of the physical page (40f8 + 0 = 40f8).

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 A has been modified (by a write or read-modify-write). This applies only to addresses in this page. A "1" might have been shown under the "S" status bit in the FC line. It would indicate that only supervisor accesses are allowed for this function code. This restriction would apply to all addresses of this function code, even though S=1 only appeared at the upper level of the table.

The L/U and LIMIT status entries work together. For example, if Table A had eight bits assigned to it (shown in the Table Level line), then there could be as many as 256 entries in Table A, pointing to as many as 256 separate Table B's, if mapping through Table B had been included. You might have specified a limit of ten Table A entries even though many more could have been specified. This limit would be shown under LIMIT in the display. In the L/U column, L would be shown to indicate that ten is the lower limit of valid spaces in Table A. A message saying "Limit violation accessing Table A" would appear if you tried to offset into Table A at some location other than the ten locations that were allowed by the limit.

The DT (descriptor type) column may show:

- SHORT indicates this is a short format table descriptor.
- LONG indicates this is a long format table descriptor.
- PAGE indicates this is the base address of a page in physical memory.
- INVALID indicates an invalid path through the tables.
- EARLY indicates this is the base address of a page in physical memory before the table levels in use have been exhausted.

Note that the address shown in the example display was mapped to supervisor program space by the function code table. This is the default mapping when function codes are used in the mapping scheme. If you wanted to see the mapping through the tables under the user function code, you would use a command like

mmu -t 30f8@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 function code (e.g. **u**, **s**, **ud**, **sd**, **up**, **sp**).

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 such as: **mmu -ta 40F8**. The emulator would interpret this as a command to show the details of Table A where it is used to map logical address 40F8. There might be a great many Table A's, but this command will only show the Table A that is used to map the logical address you specified in your command.

In the example display of table details:

- Table A has only four entries; only two bits in the logical address are used to index into Table A.
- The LOCATION column shows the physical address of each indexed location in Table A.
- The TBL/PAGE column shows the base addresses of physical pages indicated by each location in Table A. In other mapping schemes, the contents might have pointed to four base addresses of B tables.
- The first indexed location in Table A shows that its associated physical page has been accessed and modified ("U" bit and "M" bit both equal "1").

```
U>mmu -ta 40f8H
  Logical Address (hex)   supr prog     0    0    0    0    4    0    F    8
  Logical Address (bin)      110     0000 0000 0000 0000 0100 0000 1111 1000
  Table Level               FCODE    IIII IIII IIII IIIA APPP PPPP PPPP PPPP

  LEVEL INDEX LOCATION     CONTENTS      TBL/PAGE L/U LIMIT S CI M U WP DT
  A     0000  00001040     00000019      00000000             0  1 1 0  PAGE
  A     0001  00001044     00008001      00008000             0  0 0 0  PAGE
  A     0002  00001048     00010001      00010000             0  0 0 0  PAGE
  A     0003  0000104c     00018001      00018000             0  0 0 0  PAGE
U>
```

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

- **-d** disables the deMMUer. 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.
- **-e** enables the deMMUer. Your analyzer receives logical addresses translated by the deMMUer according to the tables in place when you last loaded the deMMUer.
- **-l** loads the deMMUer. The emulator will read the MMU registers and interpret the translation tables to load the deMMUer.
- **-v** sets verbose mode for the deMMUer load function. A list is displayed of the physical address ranges that will be reverse-translated by the deMMUer.

Restrictions when using the deMMUer

Keep the deMMUer up to date

When you load the deMMUer (**dmmu -l**), the emulator reads the present value of the TC, SRP, and CRP 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 for changes in the translation setup is to load the deMMUer (**dmmu -l**) 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 software breakpoint in a region of your target program that is outside of the time-critical regions and perform the load of the deMMUer after the software breakpoint is hit.

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 the highest logical address. 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.

Resource limitations

If you enter the command **dmmu -l** and your emulator performs its task and returns a prompt to the screen, you won't need to know about the deMMUer resource limitations. When the deMMUer is loaded without any problems, the prompt simply shows on screen and you can proceed with your measurement. The following information will help you deal with problems when you try to load the deMMUer and receive a message such as "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. Normally, entries in this table are allocated automatically, without intervention.

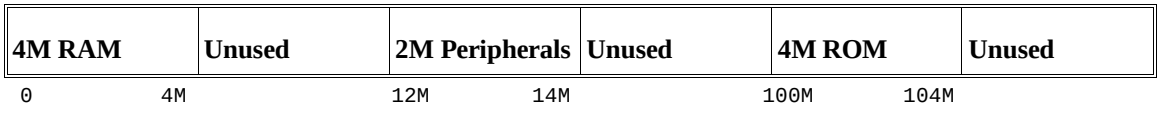
address..address
address..address
address..address
address..address
address..address
address..address
address..address
address..address

Small-page/large-page modes

The size of each address range in the table depends on the page size you selected in the TC register of the MMU. If you are using the small-page mode (smaller than 4 Kbytes per page), each address range in the deMMUer table will be 2 Mbytes; up to 16 Mbytes of physical addresses can be reverse translated. If you are using the large-page mode (4 Kbytes or more per page), each address range in the deMMUer table will be 32 Mbytes; up to 256 Mbytes of physical addresses can be reverse translated.

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 12 to 14 Mbytes, and one from 100 to 104 Mbytes. The rest of the physical address space is unused.

If your MMU mapping tables are set up to only access memory in these ranges, your deMMUer will load properly and you can proceed with your measurements. If you have failed to restrict your mappings to these physical ranges, the deMMUer may not load completely.

When the emulator tries to load the deMMUer and finds there is more physical memory identified in the MMU mapping tables than it can translate in its deMMUer table, it will try to get help by reading the emulation memory map. If the emulation memory map is arranged as follows, the deMMUer will load in a way that ensures the physical ranges of interest will be in the deMMUer.

0..4m eram
12m..14m tram
100m..104m trom
other grd

When the emulator reads the memory map for help in loading the deMMUer, it sorts the entries by size first and address range second. If using the small-page mode, the smallest address range (12m..14m) will occupy term 1 in the deMMUer translation table. Address range (0..4m) will occupy terms 2 and 3, and address

range (100m..104m) will occupy terms 4 and 5. Terms 6, 7, and 8 will be assigned to any other physical ranges encountered in the tables. You may see an "out of resources" message because the deMMUer table might run out of space, but the program spaces you care about will all be reverse translated. You can use the verbose option of the deMMUer load command to make sure the program spaces you care about will be reverse translated.

If you had been using the large page mode, your entire program would have fit within two table entries.

How to avoid the "out of resources" message

With the above example, you could have avoided the "out of resources" message. If you had placed invalid descriptors in your MMU tables in the paths that lead to unused physical address ranges, or if you had used limit fields in your table descriptors, the deMMUer would have had more than enough spaces in its eight-entry table to reverse translate the valid address ranges.

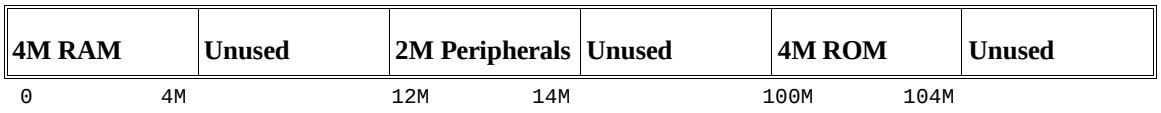
Other ways to conserve space in the deMMUer table

Minimize address ranges in the memory map

You can compose a 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 spaces in its eight-entry table to reverse translate those physical address ranges.

Careful use of the emulator memory map

What if you tried to accommodate the example program (repeated here for your convenience) with the following memory map?




```
0..4m eram
100m..104m trom
map other tram
```

Your memory map will work fine for the emulator. The 2M of peripheral address space will be in the space mapped to tram and the emulator will handle it correctly. The map will not give much help to the deMMUer. The emulator will read the memory map and assign deMMUer terms to reverse translate, as follows:

- Terms 1 and 2 to reverse translate address range 0..4m.
- Terms 3 and 4 to reverse translate address range 100m..104m.
- Terms 5, 6, 7, and 8 may be used to reverse translate address ranges 4m..6m, 6m..8m, 8m..10m, and 10m..12m.
- The "out of resources" message will be displayed and the peripherals address space will not be reverse translated.

You can get the reverse translations you need in the above example by adding one more term to your emulation memory map: 12m..14m trom. This small map entry will cause the emulator to reserve a term in the deMMUer table to reverse translate your peripherals address space. Without this entry, the deMMUer will run out of room for table entries before it gets to your peripherals address space.

Use the verbose option to the deMMUer load command to make sure all of the memory space you need to reverse translate is loaded into the deMMUer table (**dmmu -lv**).

What the emulator does when it loads the deMMUer

When the emulator loads the deMMUer, it does the following:

- 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.

Dividing the deMMUer table between user and supervisor memory space

Using two root pointers

If you enable use of the supervisor root pointer (SRP), you can have two sets of MMU translation tables, one under each root pointer. The emulator divides the deMMUer table into two equal address spaces. The first four spaces provide reverse translations for user physical address ranges, and the last four spaces provide reverse translations for supervisor physical address ranges.

address..address@u
address..address@u
address..address@u
address..address@u
address..address@s
address..address@s
address..address@s
address..address@s

If you have more physical supervisor address space than can be reverse translated by four table entries, you will receive the "out of range" message.

Using function codes

If you enable use of function code addressing (FCL=1 in the TC register), the emulator assumes you have four sets of MMU translation tables, one under each of the following: user data, user program, supervisor data, and supervisor program. In this case, the emulator divides the deMMUer table into four blocks, with two entries for each of the supported function codes.

address..address@ud
address..address@ud
address..address@up
address..address@up
address..address@sd
address..address@sd
address..address@sp
address..address@sp

If you have more physical address space than can be reverse translated by two entries under any one of the four function codes (for example three deMMUer table entries needed for user program space), you will see the "out of resources" message.

The deMMUer will not provide any reverse translations for MMU tables under any of the undefined function codes.

The emulator does not rigidly enforce the splitting of the resources into equal sizes for each function code. If the emulation memory map contains entries referring to specific function codes, they will take precedence, and may cause the deMMUer to contain only entries of that type.

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 "mmu" 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 **mmu** command can show you the details of how your logical addresses are mapped to the wrong physical addresses.

You can also use the **mmu** command to test your mappings before you enable the MMU. Simply enter the command **mmu**.

The **mmu** command by itself reads all present translations in your MMU tables. No invalid or illegal paths are shown in the listing. You can read through the display on screen to see if all of your address ranges are represented, and if they are mapped to appropriate space in physical memory.

When you enter the **mmu** command, the emulator reads the MMU registers (TC, CRP, and SRP) and MMU tables, even if the enable bit in the TC register is in the "disable" state. If you do not have correct values in the TC, CRP, and SRP registers, the emulator will let you specify correct values to be used when composing the display of translations by using the **tc, crp**, and/or **srp** options to the **mmu** command.

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 "mmu -t <address>" command 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 (type in **reg**) to see the value of the logical address in the program counter. Then use the "mmu -t <address>" command to see the path through the tables that the MMU took when it translated that logical address to a guarded address in physical memory. Note that the value of the program counter may have changed after the guarded access occurred. In this case, the present address in the program counter may map to proper physical memory.

If the present program counter address does not translate to an address in guarded physical memory, the access to guarded memory may have been caused when your program read or wrote to data memory before the present program counter address appeared. Set up the analyzer to make a trace (with the deMMUer turned on) and trigger at the logical program counter address (**tg addr=<pc address>**). Select a center trigger so you can see activity preceding and following the trigger point (**tp c**). In order to capture every transaction on the emulation bus, qualify all states for capture (**tsto any**).

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 (**mmu -t <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, you can look at the details of Table B for that address by using a command, such as **mmu -tb <address>**.

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 (**tg never**) 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 **mmu -t <address>** commands to see how the MMU maps each one to physical memory.

Software breakpoint problems

You get the "undefined software breakpoint" message after you turn on the MMU. How can this happen? You set a breakpoint. The emulator saved the instruction at the breakpoint address and wrote a BKPT #7 instruction in its place. Then the MMU changed its mappings. Now the logical address where the breakpoint is to occur is translated to a different physical address. The breakpoint is not there. No emulation break occurs when the logical address is translated to the new physical address. Some different logical address will eventually be translated through the MMU to reach the physical address where the BKPT #7 instruction is located. When the emulator finds the BKPT #7 instruction at the address, it will have no record of placing a breakpoint there, and no record of what the original instruction was for that address. All the emulator can do is display the message, Undefined software breakpoint.

You set a software breakpoint, but when the breakpoint address is hit, no software break occurs. How can this happen? You can write-protect addresses in the MMU mapping tables. If you have write protected the address where you set your

software break point in the MMU mapping tables, then the BKPT #7 instruction never gets written into that address when you set the software breakpoint. Because the old instruction remains at the breakpoint address, the emulator is not able to recognize a breakpoint at that address. Make sure that you remove any write protection in the MMU mapping tables for addresses where you want to set software breakpoints before you try to set them.

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 MC68030 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: **rst -m**.

The **rst** 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: **mmu**.

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 **mmu** command might show the logical address range of your foreground monitor mapped to physical addresses beginning at C000H, as follows:

```
LOGICAL ADDRESS       PHYSICAL ADDRESS
    Lower      Upper      Lower      Upper
      4000       4FFF    0000C000   0000CFFF
```

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: **mmu -t 4000**. In this example, you would see the following display of mappings:

```
Mmmu -t 4000
  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 AAAA BBBB BBBB CCCC PPPP PPPP PPPP

  LEVEL INDEX LOCATION     CONTENTS      TBL/PAGE L/U LIMIT S CI M U WP DT
  CRP                 00000003 00001020 00001020  U  0000              LONG
  A     0000  00001020 0000000b 00001030 00001030  U  0000  0       1 0  LONG
  B     0000  00001030 0007000a 00001040 00001040  U  0007  0       1 0  SHORT
  C     0004  00001050     0000c009      0000c000              0  0 1 0  PAGE

  Physical Address (hex) = 0000c000
```

In the example display, the foreground monitor whose logical address is 4000 was placed in physical address C000. Table C points to the page containing the foreground monitor. The base address of Table C is 00001040, and the content used by logical address 4000 is at index 0004 whose physical address is 00001050. 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>m -dl 00001050=00004009**

After this modification, you can get a new display of the mapping tables for logical address 4000 to see if your modified MMU tables now map your foreground monitor correctly. Enter the command: **M>mmu -t 4000**

```
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 AAAA BBBB BBBB CCCC PPPP PPPP PPPP

LEVEL INDEX LOCATION     CONTENTS      TBL/PAGE L/U LIMIT S CI M U WP DT
CRP                  00000003 00001020 00001020  U  0000               LONG
A     0000  00001020 0000000b 00001030 00001030  U  0000  0      1 0  LONG
B     0000  00001030 0007000a 00001040 00001040  U  0007  0      1 0  SHORT
C     0004  00001050     00004009      00004000             0  0 1 0  PAGE

Physical Address (hex) = 00004000
```

Note that in the example shown here, the MMU page size was 4 Kbytes. Therefore, the monitor was contained on one page. If the page size had been smaller, the 4 Kbytes of the foreground monitor would have occupied two or more pages. You would have to modify the logical base address of each page that contained monitor code.

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

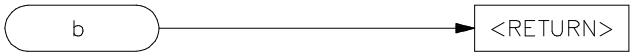
Emulator Commands

Syntax and options for Terminal Interface commands

The Command Set

This chapter describes all the commands in the HP 64747/HP 64748 Terminal Interface. Each command description includes syntax and parameter information, along with a description of command operation and a list of other commands that are often used with a particular command.

b



The **b** command issues a break to the emulator, causing it to stop executing the user program and begin execution of the monitor program.

There are no parameters to this command.

Examples

Break the emulation microprocessor into the monitor by typing:

```
U> b
```

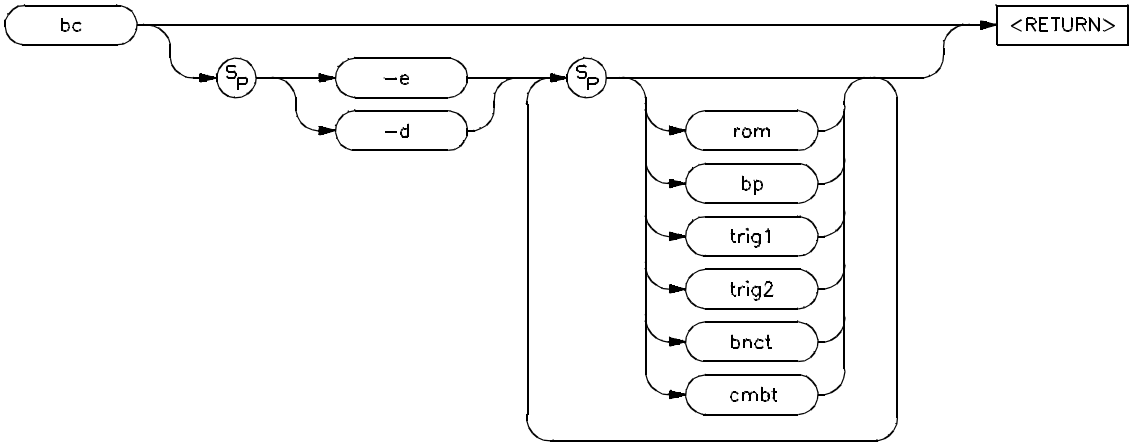
If the emulator is in the reset state when a break occurs, it will be released from reset and will begin execution within the emulation monitor.

See Also

r (runs the user program from the current pc or a specified address)

s (steps the user program a number of instructions from the current pc or a specified address)

bc



The **bc** command allows you to set break conditions for the emulation system.

The parameters are as follows:

-e — Enables the indicated break conditions (which must be specified immediately following the **-e** on the command line).

-d — Disables the indicated break conditions (which must be specified immediately following the **-d** on the command line).

The options **-e** and **-d** cannot both be specified within the same **bc** command.

rom — Enable/disable emulator breaks to monitor on occurrence of a write to ROM by the user program.

A microprocessor like the MC68020 with a pipeline architecture begins execution of the next instruction before it completes execution of the current instruction. Because a write to ROM cannot be detected until the bus cycle that causes it has completed, situations will arise where instructions after the instruction that caused the write to ROM to occur will execute.

bp — Enable/disable recognition of software breakpoints inserted with the **bp** command.

The "breakpoints" break condition should not be disabled (**bc -d bp**) while the emulator is running user code. If this command is entered while the emulator is running user code, and the emulator is executing code in the area where the breakpoints are being modified, program execution may be unreliable. (Breakpoints are modified as a result of the **bc -d bp** command because enabled breakpoints are replaced by the original opcodes when the "breakpoints" break condition is disabled.)

bnct — Enable/disable breaks generated by assertion of the **bnct** (rear panel BNC) signal. Note that this signal may also drive either the **trig1** or **trig2** signals; or, it may drive both.

cmbt — Enable/disable breaks upon assertion of the CMB (Coordinated Measurement Bus) trigger signal. Note that the CMB trigger signal may also drive either the **trig1** or **trig2** signals; or, it may drive neither or both.

trig1 — Enable/disable breaks generated by assertion of the **trig1** (trace trigger one) signal. Refer to the **tgout**, **bnct**, and **cmbt** commands for information on specifying drivers and receivers of the **trig1** signal.

trig2 — Enable/disable breaks generated by assertion of the **trig2** (trace trigger two) signal. Refer to the **tgout**, **bnct**, and **cmbt** commands for information on specifying drivers and receivers of the **trig2** signal.

If no parameters are specified, the enable/disable status of all six break conditions is displayed.

Examples

Display the status of all six break conditions:

```
M> bc
```

Enable breaks on write to ROM and upon assertion of the **trig1** signal, and disable software breakpoints and breaks generated by the **trig2** signal:

```
M> bc -e rom trig1
M> bc -d bp trig2
```

You can independently enable or disable six different break conditions: write to ROM, software breakpoints, breaks due to assertion of the BNC or CMB trigger signals, and breaks due to the assertion of the internal **trig1** and **trig2** signals. This allows you to have the emulator break to the monitor upon error conditions (such as

write to ROM or finding a software breakpoint in a piece of code it never should have reached), or break to the monitor when an analyzer measurement has completed.

When you use the **bc** command, the emulator may break into the monitor while each enable/disable is being executed. If the emulator was executing your program when the **bc** command was received, it will return to your program when finished executing the command. If you request only a display of the current break conditions, the emulator does not break to the monitor.

A hardware reset that occurs during processing of the **bc** command may result in the particular break condition being left in an unknown state. If this occurs, a display of the break conditions will show a question mark "?" instead of **-e** or **-d** next to the break condition.

See Also

bnct (specify drivers and receivers of the rear panel BNC signal)

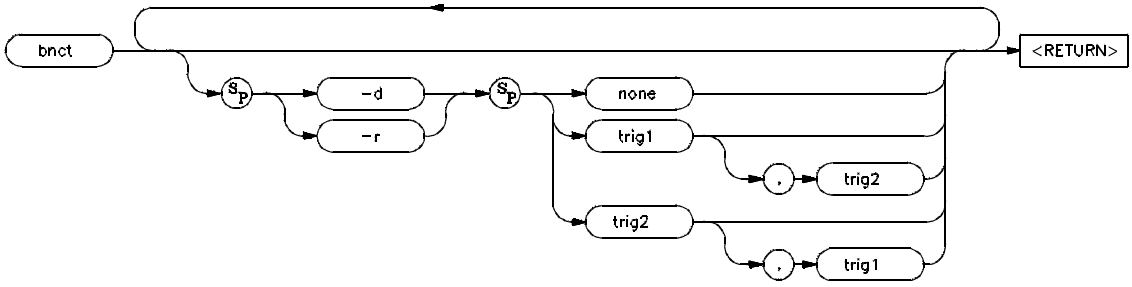
cmbt (specify drivers and receivers of the CMB trigger signal)

bp (set/delete software breakpoints)

map (specify whether memory locations are mapped as RAM or ROM)

tgout (specify whether the **trig1** and/or **trig2** signals are to be driven when the analyzer finds the trigger condition)

bnct



The **bnct** command allows you to specify which of the internal **trig1/trig2** trigger signals will drive and/or receive the rear panel BNC trigger. You can specify the signals individually, as an ORed condition for drive, or as an ANDed condition for receive; or, you can specify that the signals are not to be driven or received.

The parameters are as follows:

-d — The **-d** parameter indicates that the BNC port will drive the triggers, trig1 and trig2, to the emulator's internal analyzer.

-r — The **-r** parameter causes the BNC port to receive the triggers, trig1 and trig2, from the analyzer, and send them out the BNC port.

none — If you specify **none** with the **-d** option, then the rear panel BNC signal will not drive either of the analyzer triggers. If you specify **none** with the **-r** option, the rear panel BNC will not receive trig1 or trig2 from the internal analyzer.

trig1 — If **trig1** is specified, then the internal "trig1" signal will drive or receive the BNC signal, depending on whether you specified the **-d** or **-r** option.

trig2 — If you specify **trig2**, then the internal "trig2" signal will drive or receive the BNC signal, depending on whether you specified the **-d** or **-r** option.

You can also specify that both the **trig1** and **trig2** signals are to drive or receive the BNC signal. To do this, place a comma between the two signals on the command line.

Defaults

If no options are specified, the current setting of **bnct** is displayed. Upon powerup, **bnct** is set to **bnct -d none -r none**.

If you specify one of the **-d** or **-r** options without the other, the other option is left in the same state it was in before the command was entered.

Examples

To view the current **bnct** setting, type:

```
M> bnct
```

To trigger an instrument hooked to the BNC when the HP 64700 analyzer finds its trigger, you might do the following:

```
M> tcf -e
M> tg addr=2000
M> tgout trig1
M> bnct -d none -r trig1
```

By specifying this command sequence, the external instrument will be triggered when the emulation processor reaches the trigger pattern of address=2000.

The reverse situation is where you want to trigger the HP 64700 analyzer when an external instrument finds its trigger. Type:

```
M> bnct -d trig1 -r none
M> tarm =trig1
M> tg arm
```

Normally, you would use this command to cross-trigger instruments. For example, you may wish to trigger a digitizing oscilloscope connected to various timing signals when the emulation-bus analyzer finds a certain state, or you may wish to do the converse and trigger the HP 64700's analyzer when an oscilloscope finds its trigger.

You should not set up an analyzer in an emulator to both drive and receive the same trigger signal. For example, if you issued the commands **tg arm**, **tarm =trig1**, **tgout trig1**, and **bnct -d trig1 -r trig1**, then the analyzer **trig1** signal will become latched in a feedback loop and will remain latched until the loop is broken. To break the loop, you must first disable the source of the signal, and then

momentarily disable either the drive or receive function. In this case, the commands **tgout none** and **bnct -d none** will break the loop.

See Also

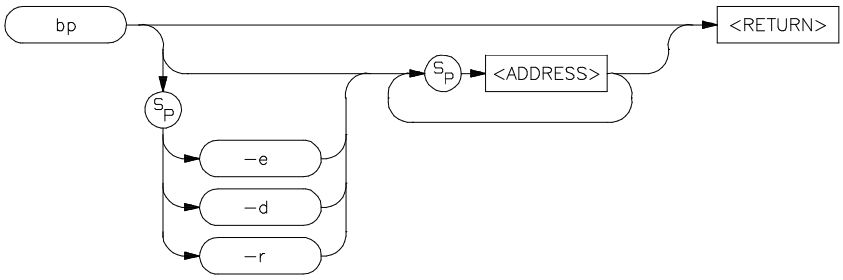
bc (break conditions; can be used to specify that the emulator will break into the emulation monitor upon receipt of one of the **trig1**/**trig2** signals)

cmbt (coordinated measurement bus trigger; used to specify which internal signals will be driven or received by the HP 64700 coordinated measurement bus)

tarm (analyzer trace arm; used to specify arming (begin to search for trigger) conditions for the analyzer -- **trig1/trig2** can be used to arm the analyzer)

tgout (specifies which of the **trig1/trig2** signals are to be driven when the analyzer trigger is found)

bp



The **bp** command is used to insert, delete, display, or modify the status of software breakpoints.

The parameters are as follows:

<ADDRESS> The **<ADDRESS>** parameter allows you to specify the address location where the software breakpoint is to be inserted. If you specify options **-d**, **-a**, or **-h**, then the address specifies the location of the breakpoint to be deleted, activated, or inactivated. For these options, you may specify the character ***** as the address specifier, indicating that the operation is to be performed on all of the addresses present in the software breakpoint table.

The default for the **<ADDRESS>** parameter is a hexadecimal expression, however, other numeric bases may be specified. See the **<ADDRESS>** syntax pages in Chapter 11 for information on specifying address information.

The memory access mode for writing breakpoints is set by the **mo** (mode) command; if the mode is set to byte access and an odd address location is specified, an invalid instruction may be inserted for processors that expect alignment of opcodes on even byte boundaries.

-r Deletes the software breakpoint(s) at the addresses specified. If the address specified does not contain a breakpoint instruction, an error will be returned. When the breakpoint is deleted, the original memory contents are restored, and then the address is removed from the breakpoint table.

-e Enables (activates) the breakpoint(s) at the address(es) specified. This installs the necessary breakpoint instruction in memory. If the breakpoint is already enabled, no action is taken.

-d

Disables (deactivates or "hits") the breakpoint(s) at the address(es) specified. The breakpoints remain in the breakpoint definition table and can be reset by using the **bp -e <ADDRESS>** command. If the breakpoint is already disabled, no action is taken.

If no parameters are specified, the current status of all breakpoints is displayed. Upon powerup or **init** initialization, the breakpoint table is cleared and the breakpoint feature is disabled.

Examples

The following examples use the demo program.

Assume that you need to verify that the processor is reaching the Cmd_A, Print_Msg, and Fill_Dest routines. You can insert software breakpoints at these addresses and run the program to each successive breakpoint. First, enable the software breakpoint feature.

```
M> bc -e bp
```

Now define the breakpoints at the start of each routine by typing:

```
M> bp handle_msg:Cmd_A Print_Msg Fill_Dest
```

View the current breakpoint settings:

```
M> bp
```

When you run the processor and enter commands for the demo program, all the breakpoints will be "hit" (executed by the processor) and will be disabled. You can reenable the existing breakpoints:

```
M> bp -e handle_msg:Cmd_A Print_Msg Fill_Dest
```

You could also type **bp -e *** to reenable all breakpoints in the table. Disable the breakpoint at Fill_Dest:

```
M> bp -d handle_msg:Fill_Dest
```

Remove the breakpoint at Print_Msg from the breakpoint table:

```
M> bp -r handle_msg:Fill_Dest
```

Disable all breakpoints in the table:

```
M> bp -d *
```

Remove all breakpoints in the table:

```
M> bp -r *
```

Disable the breakpoint feature,

```
M>bc -d bp
```

The MC68020 and MC68030/EC030 emulators use the BKPT instruction to implement software breakpoints. The **cf sw** command sets the particular version of the BKPT instruction that is used. There are four different operations to maintain the software breakpoint table.

Inserting Breakpoints

Specifying only an address inserts the breakpoint instruction in memory and makes a breakpoint table entry corresponding to that address. If a software break instruction already exists at the address specified, an error message is generated and the current **bp** command is aborted.

Enabling Breakpoints

Enabling a breakpoint at a specified address causes the system to search the breakpoint table for that address; if it exists in the table, the breakpoint instruction is written to memory at the corresponding address.

Disabling Breakpoints

Disabling the breakpoint for a specified address again causes a search for a breakpoint table entry; if found, the original contents of the address (before the breakpoint was defined) are written to the corresponding memory location. The contents of the breakpoint table are unchanged, except to indicate that the particular breakpoint is now inactivated.

When the breakpoint table is displayed with the **bp** command, the enable/disable status of each breakpoint is tested by reading the memory locations in question. If a

software break instruction is found, the breakpoint is displayed as **enabled**; if not, the breakpoint is displayed as **disabled**.

Software breakpoints should not be set, enabled, disabled, or removed while the emulator is running user code. Also, you should not disable the "breakpoints" break condition (**bc -d bp**) while the emulator is running user code.

If any of these commands are entered while the emulator is running user code, and the emulator is executing code in the area where the breakpoint is being modified, program execution may be unreliable.

The problem occurs when the software breakpoint instruction (or the original opcode) is partially written in the emulation memory location while the emulation processor is fetching from that location; an illegal opcode may result. This problem does not occur when breakpoints are in target RAM because the emulation processor breaks into the monitor to enable or disable software breakpoints.

Removing Breakpoints

Removing a breakpoint causes a search for a corresponding breakpoint table entry; if found, the original memory contents are written to the specified address, and the entry is removed from the breakpoint table.

When a software breakpoint instruction inserted by **bp** is executed by your program, it is removed from memory and marked **disabled** in the breakpoint table.

A status message indicates that a software breakpoint was found.

If the emulator executes a software break instruction that was placed by you (either through your compiler or via memory modification) and not by the **bp** command, an "undefined breakpoint" error message is generated.

If the emulator is executing in the user program when you define or modify breakpoints, it may break into the monitor for each breakpoint defined or modified. Whether or not it will do this depends on the location of the breakpoint in memory (breaks to the monitor are required if the location is in target RAM), and whether your particular emulator must break to the monitor for accesses to that memory type (breaks into the monitor are not necessary if the location is in dual-port emulation memory). If a break to the monitor is required, the emulator will return to user program execution after breakpoint definition or modification.

In general, you should only define software breakpoints at memory locations that contain user program instructions. If you set breakpoints at other locations, it is unlikely that they will ever be executed. The only exception to this might be in a

case where you suspect that your program is jumping into a data block and attempting to execute code; setting a software breakpoint in this area will allow you to verify the problem (and stop a runaway program).

Remember that any operation which modifies memory or the memory map will alter the existing breakpoints. For example, if you load a new program in the same address range where breakpoints reside, the breakpoints will be destroyed. Changing the memory map will prevent the emulator from placing new breakpoints or enabling existing breakpoints.

You cannot define breakpoints until you have enabled them with the **bc -e bp** command. If you disable the software breakpoint feature with the **bc -d bp** command, the breakpoints currently defined will remain in the breakpoint table, but will be disabled and will remain in that state until the breakpoint feature is reenabled and the specified breakpoints are reenabled (**bc -e bp** and **bp -e <ADDRESS>**).

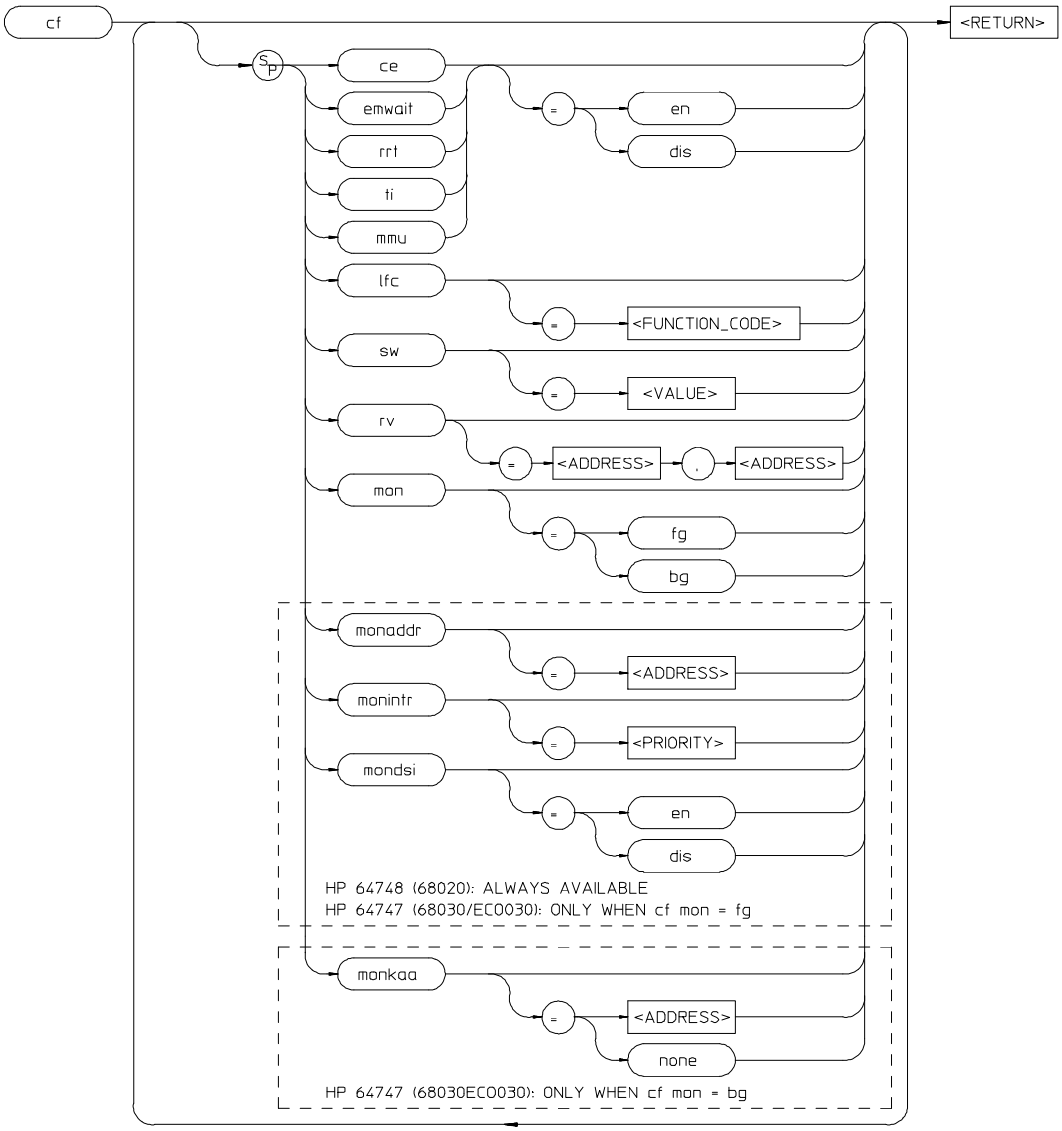
See Also

bc (enable/disable breakpoint conditions (including **bp**))

cf (set instruction type used for software breakpoint (only available on some processors))

mo (defines memory access and display modes; the **bp** command uses the currently defined modes when writing software breakpoints into memory)

cf



The **cf** command allows you to modify various emulator specific configuration parameters. The MC68020 and MC68030/EC030 configuration items allow you to

set up the emulator in a way that best suits your system needs. Many configuration items allow you to configure the emulator to work properly with your target system.

The parameters are as follows. If you enter the configuration item name without a setting, the current setting is displayed.

ce	Enable/disable the instruction cache for the MC68020, or the instruction and data caches for the MC68030/EC030.
emwait	Enable this if you are using emulation memory at clock speeds greater than 25 MHz. Otherwise disable this configuration item.
mondsi	Enables $\overline{\text{DSACK}}$ interlocking between the emulator and target system for both foreground and background emulation monitor bus cycles on the MC68020. Enables $\overline{\text{DSACK}}$ and $\overline{\text{STERM}}$ interlocking between the emulator and target system for foreground emulation monitor bus cycles on the MC68030/EC030.
rrt	Restricts the emulator to real time runs.
ti	Enable/disable target system interrupts.
lfc	Sets the function code for the next load command.
mmu	Enables or disables the MC68030 MMU. If enabled, the /MMUDIS line and TC register determine whether or not logical-to-physical address translations are performed. If disabled, the emulator asserts the /MMUDIS line to prevent address translations. This is the hardware enable for the 68030 MMU.
mon	Chooses a foreground or background monitor. If you enable the MC68030 MMU, the foreground monitor will be selected automatically.
monaddr	Sets the base address for the foreground monitor (and maps a corresponding memory block).Also sets the address range for which bus cycles will appear for the background monitor in the MC68020 emulator. Not available in background on the MC68030/EC030 emulator.
rv	Sets the initial interrupt stack pointer and PC values when the emulator enters the monitor from reset. Allows the emulator to run correctly if you break to monitor after reset, and then start a run.
monintr	Lowers the interrupt priority mask during foreground monitor execution so that target system interrupts above that level will be serviced. Not available in the MC68030/EC030 emulator when the background monitor is selected.
sw	Sets the form of the BKPT instruction used to implement software breakpoints.

monkaa

Defines an address from which the background monitor of the MC68030/EC030 periodically reads a byte. Used to "keep-alive" circuits that depend on constant bus activity, such as watchdog timers or dynamic RAM.

Examples

To set the software breakpoint instruction to BKPT 6, enter:

```
M> cf sw=6
```

To block interrupt requests from the target system, enter:

```
M> cf ti=dis
```

To select a foreground monitor and put it at address 4000 hex, enter:

```
M>cf mon=fg

R>cf monaddr=4000
```

The default configuration of the emulator after initialization is as follows:

- Configuration items:
  - HP 64748 (MC68020):

```
R>cf

 cf ce=en
  cf emwait=en
  cf lfc=x
  cf mon=bg
  cf monaddr=1000
  cf mondsi=en
  cf monintr=6
  cf rrt=dis
  cf rv=1,0ffffffff
  cf sw=7
  cf ti=en
```

  - HP 64747 (MC68030/EC030):

```
R>cf
```

```
cf ce=en
cf emwait=en
cf lfc=x
cf mmu=dis
cf mon=bg
cf monkaa=none
cf rrt=dis
cf rv=1,0ffffffff
cf sw=7
cf ti=en
```

- No memory map terms assigned, all other memory mapped to target RAM (**tram**).
- All break conditions (**bc**) are disabled.

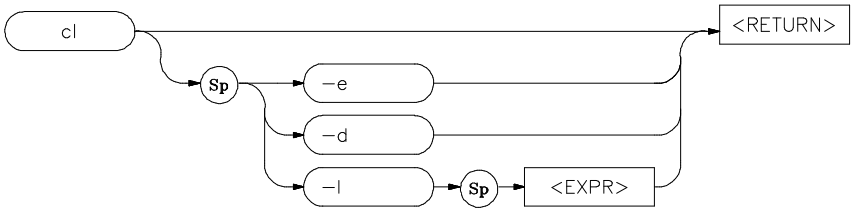
When you connect the emulator to a target system, you may want to modify all configuration items. The modifications you make depend on your target system requirements.

See Also

help (you can get an on line display of the configuration items for a particular emulator by typing **help cf**. To obtain more information regarding a particular configuration item, type **help cf <config_item>**).

Also see Chapter 7 of this manual.

cl



You can enable command line editing to include the ability to manipulate command text lines.

The parameters are as follows:

-d This option disables command line editing.

-e This option enables command line editing.

-l This option allows you to set the column length for the command line. This value can be from 40 to 132 columns.

Command line editing is disabled by default.

Examples

Set the number of columns in the command line to 80:

```
cl -l 80
```

Enable command line editing:

```
cl -e
```

Add text to the previously executed command:

```
<ESC> k A <additional text>
```

Command line editing has two typing modes. The normal command entry is input mode. The input mode functions like normal (canonical) command entry. The control mode allows command modification.

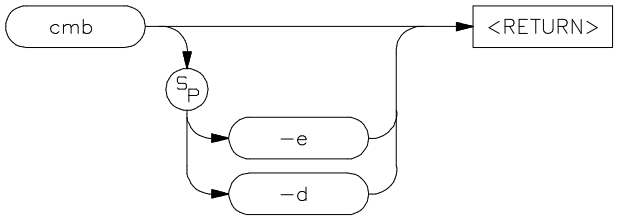
The commands in control mode are as follows:

Command	Action
i	Insert before current character.
A	Append to end of line.
dd	Delete command line.
$	Move cursor to end of line.
^	Move cursor to start of line.
l	Move right one character.
j	Fetch next command.
a	Insert after current character.
x	Delete current character.
D	Delete to end of line.
0	Move cursor to start of line.
h	Move left one character.
k	Fetch previous command.
r	Replace current character.
/<string>	Find previous command matching <string>.
n	Fetch previous command matching <string>.
N	Fetch next command matching <string>.

See Also

See Chapter 3 of this manual for more information on command entry.

cmb



The **cmb** command allows you to enable or disable interaction on the CMB (Coordinated Measurement Bus). The CMB allows you to make complex measurements involving cross-triggering of multiple HP 64700 analyzers and other HP 64000 system instruments, and synchronous emulator runs and breaks .

The parameters are as follows:

-e The **-e** option enables interaction between the emulator and the Coordinated Measurement Bus.

-d The **-d** option disables interaction between the emulator and the Coordinated Measurement Bus.

If no options are supplied, the current state of CMB enable/disable is displayed.

Examples

View the current state of CMB interaction:

```
M> cmb
```

Enable CMB interaction:

```
M> cmb -e
```

Disable CMB interaction:

```
M> cmb -d
```

The **cmb** command only affects the ability for multiple emulators to run or break in a synchronized fashion; the analyzer trigger capability is unaffected by the **cmb** command.

Interaction Enabled

When interaction is enabled via the **cmb -e** command, the emulator will run code beginning at the address specified via the **rx** command when the CMB /EXECUTE (/ means active low) pulse is received.

The CMB READY line is driven false while the emulator is running in the monitor. The line goes to the true state whenever execution switches to the user program.

Notice that if the **rx** command is given, CMB interaction is enabled just as if a **cmb -e** command was issued. Refer to the syntax pages for the **rx** command for further information.

Interaction Disabled

When interaction is disabled via the **cmb -d** command, the emulator ignores the actions of the /EXECUTE and READY lines. In addition, the emulator does not drive the READY line.

See Also

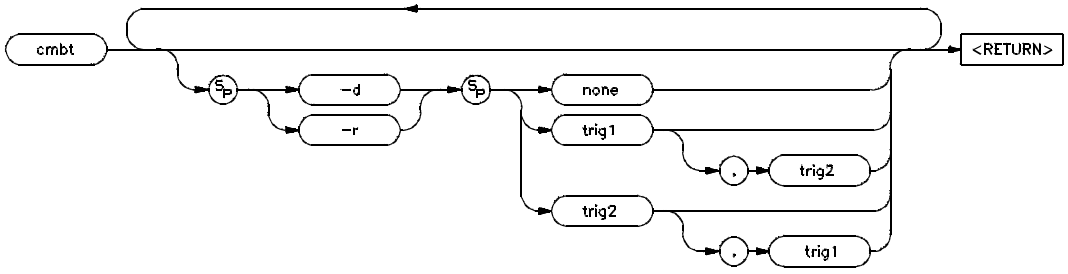
rx (allows you to specify the starting address for user program execution when the CMB /EXECUTE line is asserted)

tx (controls whether or not the emulation analyzer is started when the /EXECUTE line is asserted)

x (pulses the /EXECUTE line, initiating a synchronous execution among emulators connected to the CMB and enabled)

Also, see Chapter 6, “Coordinated Measurements,” for more information on CMB operation.

cmbt



The **cmbt** command allows you to specify which of the internal **trig1**/**trig2** trigger signals will drive and/or receive the rear panel CMB (Coordinated Measurement Bus) trigger. You can specify the signals individually, as an ORed condition for drive, or as an ANDed condition for receive; or, you can specify that the signals are not to be driven and/or received.

The parameters are as follows:

-d — The **-d** parameter causes the CMB to drive the trigger signals, trig1 and trig2, to the emulator's internal analyzer.

-r — The **-r** parameter causes the CMB to receive the trigger signals, trig1 and trig2, from the analyzer.

none — If you specify **none** with the **-d** option, then the CMB trigger signal will not drive either of the analyzer triggers. If you specify **none** with the **-r** option, the rear panel CMB will not receive trig1 or trig2 from the emulation-bus analyzer.

trig1 — If **trig1** is specified, then the internal "trig1" signal will drive or receive the CMB trigger signal, depending on whether you specified the **-d** or **-r** option.

trig2 — If you specify **trig2**, then the internal "trig2" signal will drive or receive the CMB trigger signal, depending on whether you specified the **-d** or **-r** option.

You can also specify that both the **trig1** and **trig2** signals are to be driven and/or received. To do this, place a comma between the two signals on the command line.

If no options are specified, the current setting of **cmbt** is displayed. Upon powerup, **cmbt** is set to **cmbt -d none -r none**.

Examples

To view the current **cmbt** setting, type:

```
M> cmbt
```

Trigger the analyzer in another 68020 emulator connected to the CMB:

```
M> tcf -e
M> tg addr=demo:Loop
M> tgout trig1
M> cmbt -d none -r trig1
```

Set the other HP 64700 analyzer to break to monitor upon receiving the CMB trigger:

```
M> cmbt -r trig1
M> bc -e cmbt
```

You might want to have an external instrument arm the analyzer in one emulator which then arms a second analyzer attached through the CMB. The second emulator then breaks to monitor when it finds its trigger condition. Use the following command sequence in the first emulator:

```
M> bnct -d trig1 -r none
M> tarm =trig1
M> tsq -i 3
M> tif 1 arm
M> tif 2 addr=demo:Loop
M> tgout trig2
M> cmbt -d trig2 -r none
```

On the second emulator, type:

```
M> cmbt -d trig1 -r none
M> tarm =trig1
M> tsq -i 3
M> tif 1 arm
M> tif 2 addr=handle_msg:Cmd_A
M> tgout trig2
M> bc -e trig2
```

You use this command to trigger other HP 64700 analyzers and possibly HP 64000 system instruments. For example, you may wish to start a trace on another HP 64700 analyzer when the analyzer in this emulator finds its trigger; or, you may wish to do the converse and trigger the analyzer in this emulator when another emulation analyzer finds its trigger.

You should not set up an analyzer in an emulator to both drive and receive the same trigger signal. For example, if you issued the commands **tg arm**, **tarm =trig1**, **tgout trig1**, and **cmbt -d trig1 -r trig1**, then the analyzer **trig1** signal will become latched in a feedback loop and will remain latched until the loop is broken. To break the loop, you must first disable the signal's source, and then momentarily disable either the drive or receive function. In this case, the commands **tgout none** and **cmbt -d none** will break the loop.

See Also

bc (break conditions; can be used to specify that the emulator will break into the emulation monitor upon receipt of one of the **trig1**/**trig2** signals)

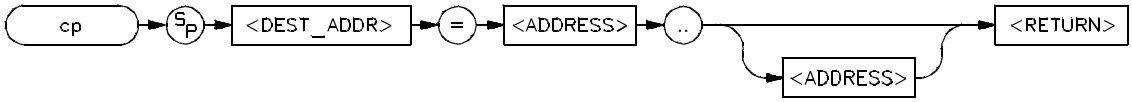
bnct (BNC trigger; used to specify which internal signals will be driven or received by the rear panel BNC connector)

cmb (Used to enable or disable interaction on the CMB. This does not affect whether measurement instruments can exchange triggers over the CMB; it only controls run/break interaction between multiple emulators)

tarm (analyzer trace arm; used to specify arming (begin to search for trigger) conditions for the analyzer -- **trig1/trig2** can be used to arm the analyzer)

tgout (specifies which of the **trig1/trig2** signals are to be driven when the analyzer trigger is found)

cp



The **cp** command allows you to copy a block of data from one region of memory to another. For example, you might want copy a data table in your program to a buffer space so you can try some of your algorithms for processing data in that buffer.

The parameters are as follows:

<DEST_ADDR> Specifies the lower boundary of the destination range. The processor specific conventions for <ADDRESS> can be used for complete address specification including function codes or segmentation. Refer to the *Emulator User's Guide* for your particular emulator for details.

<ADDRESS> Specifies the lower, and possibly upper, memory address boundaries of the source range to be copied. The default is a hexadecimal number; other bases may be specified. Certain emulators allow additional processor specific addressing information for <ADDRESS>; refer to the *Emulator User's Guide* for your particular emulator for further information.

.. The separator between the lower and upper address boundaries is two periods (..). Notice that no additional spaces are inserted. You can use "<ADDRESS>.." to specify a range from the address through the next 127 bytes."

Exactly one address range must be specified.

Examples

Copy the data area of the demo program to a base address of 700 hex:

```
M> cp 700=handle_msg:Msg_A..End_Msgs
```

When **cp** is executed, the data from the specified range is copied to the destination address, with the lower boundary data going to the destination address, lower boundary + 1 to destination + 1, and so on until the upper boundary of the source range is copied. If the source or destination addresses

reside within the target system, the emulator will break to the background monitor and will return to foreground after the copy is completed.

If memory mapped as guarded is encountered in the source or destination range during the copy, the command is aborted; however, all locations modified prior to accessing guarded memory are left in the modified state.

See Also

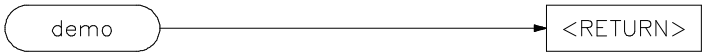
cim (copies a memory image from the target system to emulation memory)

m (allows you to display or modify memory locations or ranges)

map (used to define the type and location of memory used by the emulator)

ser (used to search memory ranges for a specific set of data values)

demo



The MC68020 and MC68030/EC030 emulators contain a simple demo program that allows you to learn about the Terminal Interface without the bother of writing and loading a program.

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

The program emulates a simple command interpreter. It has a one-byte input buffer for commands, and recognizes the ASCII characters "A" and "B." All other values are considered invalid.

When you input a command to the buffer, the program calls a subroutine that interprets the command and writes a corresponding message to an output buffer.

There are two modules in the program. One is the main module, called demo. The second module, which has the subroutines for printing the messages, is called handle_msg. These files are listed here.

Example

To load the assembly language version of the demo program, enter:

```
demo
```

```
*
* HP 64748 Demo Program
* This program is a simple 68000 assembler program that can be used
* to demonstrate run and trace features for the HP 64748 emulator.
* It emulates a simple command interpreter.
*
* The program scans the location Cmd_Input looking for a non-null
* value. When it finds one, it calls a routine to determine whether
* the command is "A," "B," or an invalid command. This routine sets
* up certain parms for the output message handler, then calls the
* message handler to write a message based on the command input.
*
* Module name: demo.s
* Written: February 27, 1991
* Last Revised: February 28, 1991
*

* Define the chip. Call it 68000 here since we don't use any 68020-specific
* features.

          chip 68000

* Int_Cmd is in the module handle_msg.

          xref     Int_Cmd

* Set up the stack pointer and initial program counter for run-from-reset.

          sect     Table,,r
          org $0

          dc.l     Top_of_Stack
          dc.l     Main

* Set up the trace vector
* so single-stepping works
          org $24
          dc.l 0

* The stack is declared as 16 long words, which should be more than
* sufficient since there shouldn't be more than 3 PC's on the stack (plus
* whatever the emulation monitor pushes).

          sect    Stack,,d
Stack     ds.l    16
Top_of_Stack

* The only data local to this module is the command input buffer, which
* is a single byte.

              sect    Data,,d

Cmd_Input     ds.b     1

* Main program starts here.

          sect    Prog,,c

* Load the user stack pointer, then clear the command input byte.
```

demo.s assembly source file

```
Main      move.l  #Top_of_Stack,a7
          move.b  #0,Cmd_Input

* Now loop, looking for a nonzero value. If found, call the message interpreter.

Loop      move.b  Cmd_Input,d0
          bne     Call_Int
          bra     EndLoop

* Call to the message interpreter is a simple subroutine branch. The parameter
* is passed in d0 (the command we read). When finished, clear the command
* input buffer.

Call_Int  bsr     Int_Cmd
          move.b  #0,Cmd_Input

* Branch forever.

EndLoop   bra     Loop
          end     Main

* End of demo.s
```

demo.s assembly source file (Cont'd)

```
* This module contains the message intepreter and message printing routines
* for the 64748 demo program. The main program module is demo.s.
*
* Module: handle_msg.s
* Written: February 27, 1991
* Last Revised: February 28, 1991


* Int_Cmd must be made global so demo.s can see it.

          xdef   Int_Cmd

* Data local to this module are the message definitions and the message
* output buffer.

          sect   Data,,d

* Define the messages printed for commands A, B and invalid respectively.

Msg_A          dc.b     'Command A entered'
Msg_B          dc.b     'Entered B command'
Msg_I          dc.b     'Invalid command'
End_Msgs

* Message output buffer.

Msg_Dest     ds.b     32

* Start program code for this module.
```

handle_msg.s assembly source file

```
          sect     Prog,,c

* Int_Cmd is the command interpreter routine. It is called by the main
* program loop in demo.s whenever a command is found.

Int_Cmd   cmp.b   #'A',d0
          beq     Cmd_A
          cmp.b   #'B',d0
          beq     Cmd_B
          bra     Cmd_I

* If "A", then load a pointer to the beginning of the "A" message into a0,
* and load the message's length into d1. Then call the routine to print
* the message. When done, return to the caller (the main program loop in
* demo.s).

Cmd_A     lea     Msg_A,a0
          move.l  #Msg_B-Msg_A-1,d1
          bsr     Print_Msg
          rts

* If "B", then load a pointer to the beginning of the "B" message into a0,
* and load the message's length into d1. Then call the routine to print
* the message. When done, return to the caller (the main program loop in
* demo.s).

Cmd_B     lea     Msg_B,a0
          move.l  #Msg_I-Msg_B-1,d1
          bsr     Print_Msg
          rts

* The command isn't recognized, so load a pointer to the beginning of the
* invalid message into a0,
* and load the message's length into d1. Then call the routine to print
* the message. When done, return to the caller (the main program loop in
* demo.s).

Cmd_I     lea     Msg_I,a0
          move.l  #End_Msgs-Msg_I-1,d1
          bsr     Print_Msg
          rts

* End of Int_Cmd



* Print_Msg handles the writing of the appropriate message to the Msg_Dest
* buffer. After the message is written, it writes nulls to the remaining
* locations to clear them from previous commands.

* To print the message, we load a pointer to the output buffer into
* a1, then do a block xfer with autoincrement addressing. When the number
* of characters specified in d1 has been moved, fall out of the Again loop.

Print_Msg      lea     Msg_Dest,a1
Again          move.b  (a0)+,(a1)+
               dbeq    d1,Again
```

handle_msg.s assembly source file

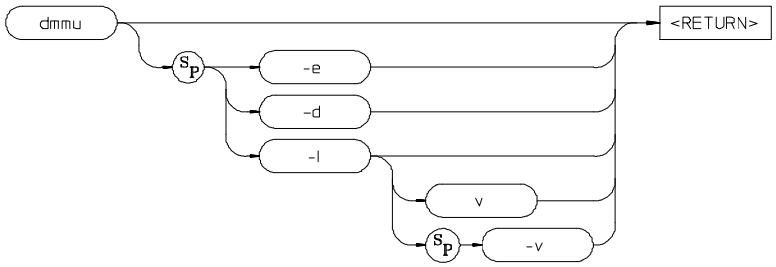
```
* Now move a null to the next location pointed to by a1 (which is now after
* the last character of the message. Compare the address in a1 to the end
* address of the message buffer, and keep repeating until all remaining
* destination buffer locations are zeroed. Then return to the caller
* (Int_Cmd in this case).

Fill_Dest      move.b  #0,(a1)+
               cmpa    #Msg_Dest+32,a1
               bne     Fill_Dest
               rts

* End of Print_Msg


* End of handle_msg.s
```

handle_msg.s assembly source file

```
CHIP 68000
LIST c,d,p,s,t,x
SECT Prog=$400
SECT Data=$500
SECT Stack=$f00

LOAD demo
LOAD handle_msg
```

demo.k linker command file

dmmu



The **dmmu** command is used to enable or disable the deMMUer so it can translate physical addresses it receives from the emulation bus and deliver corresponding logical addresses to the analyzer. This command can also be used to load the deMMUer with appropriate information to reverse the MMU translations; this is done by reading the present MMU register values and the present MMU translation tables in memory. Finally, you can see the present state of the deMMUer by simply typing **dmmu** and pressing RETURN.

If the MMU page size is at least 4 Kbytes, the deMMUer can translate up to 256 Mbytes of physical address space. If the MMU page size is smaller than 4 Kbytes, the deMMUer can translate 16 Mbytes of physical address space.

The parameters are as follows:

d — Turns off the deMMUer. Addresses on the emulation bus will be supplied directly to the analyzer without translation.

e — Turns on the deMMUer. Addresses on the emulation bus will be translated (physical to logical) before being supplied to the analyzer. Reverse translations will be made according to the setup that was present in the MMU at the time you entered your last **dmmu -l** command.

l — Reads the MMU registers and MMU tables, and loads the deMMUer.

v — Sets the verbose mode for the deMMUer load function. The verbose mode shows a list of the physical addresses that can be translated by the deMMUer after loading the deMMUer. If these address translations include function codes, the function codes are shown beside the addresses (example: 000000000..003ffffff@sp).

Examples

To enable the deMMUer to translate addresses for the analyzer:

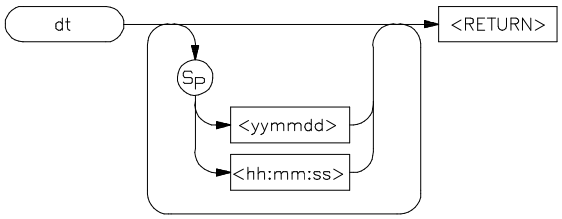
M> **dmmu -e**

To load the deMMUer to reverse translate addresses using the current translation tables:

M> **dmmu -l**

See Also **mmu** (display MMU translations or table information).

dt



The **dt** command allows you to set or display the current date and time stored by the HP 64700 series emulators.

The parameters are as follows.

<yymmdd> This variable sets the date. **yy** are the last two digits of the current year; **mm** specify the current month, and **dd** specify the day of the month.

If **yy** is greater than 50, the year is assumed to be in the 20th century (**19yy**). If **yy** is less than 50, the year is assumed to be in the 21st century (**20yy**).

<hh:mm:ss> This variable sets the time in 24 hour format. **hh** specify the hour, **mm** specify the minutes, and **ss** specify the seconds. Notice that the only difference between the date and time variables is the presence of colons; therefore, if you forget the colons while trying to reset the time, you will change the date setting.

If no parameters are specified, the current date and time settings are displayed.

Examples

Display the current date and time settings:

```
M> dt
```

Set the date to August 18, 1991:

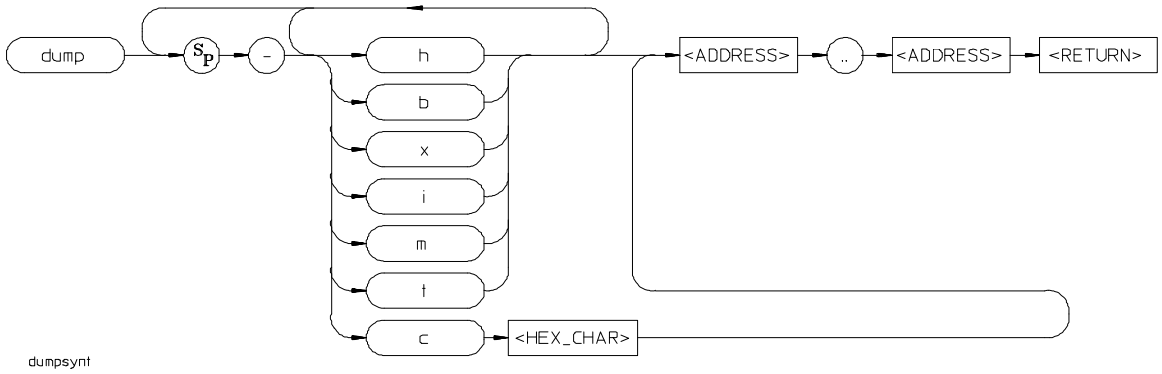
```
M> dt 910818
```

To set the date to August 18, 1991 and the time to 11:05:00, type:

```
M> dt 910818 11:05:00
```

The emulator system date & time clock is reset when power is cycled.

dump



The **dump** command allows you to dump the contents of emulation and/or target system memory to a host file. The contents can be dumped in HP, Tektronix hex, Intel hex, and Motorola S-record formats by specifying various options on the command line.

The parameters are as follows.

-h — The **-h** option indicates that the memory contents will be dumped in HP absolute file format.

-b — Specifying the **-b** option indicates that the records will be sent in binary; this is only valid with **-h** (HP file format).

-x — If you specify **-x**, the records will be sent in hexadecimal; this is only valid with the **-h** option (HP file format).

-i — Specify the **-i** option if you need to have the file transferred in Intel hex record format. Note that the various options for HP file format transfer (such as **-x**, **-b**, and **-e**) are invalid with this format.

-m — Specify the **-m** option if you need to have the file transferred in Motorola S-record format.

-t — Specify the **-t** option if you need to have the file transferred in Tektronix hex format.

-c — Specifying -c along with an ASCII hexadecimal character indicates that the character specified should be sent to the host at the end of the file upload.

<HEX_CHAR> — <HEX_CHAR> is an ASCII character to be sent to the host at the end of the upload process. The character is used to close the host file which is receiving the uploaded data.

<ADDRESS> — Specifies the lower, then upper, address boundaries of the memory range to be dumped. The default is a hexadecimal number; other bases and expressions may be supplied. Refer to the <EXPR> syntax pages for details. In addition, many microprocessors allow special address information such as segmentation or function codes to be specified; see the <ADDRESS> syntax pages in Chapter 11 for details.

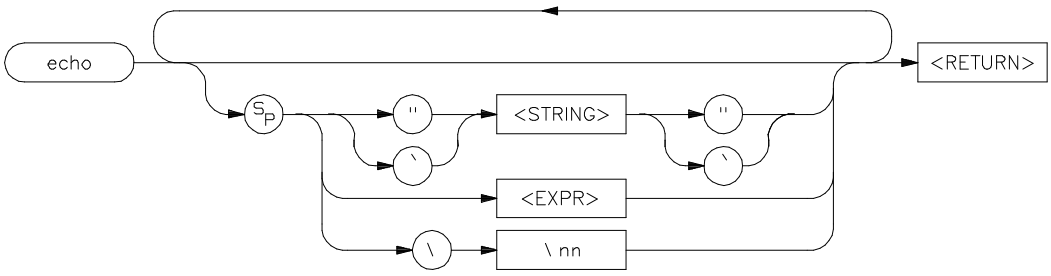
There are no defaults; a file format and address range must be specified.

If you are uploading the file in HP file format using the HP 64000 **transfer** software, record checking is performed automatically by the **transfer** protocol.

The HP 64000 format ".X" file created with a "dump -hx" command has records that contain 136 fewer bytes of data than the file format standard allows. Because of this, HP 64000 format ".X" files which are created with the **dump** command may take longer to be processed by consumers of the ".X" file (depending on how the consumer processes sequential records).

See Also — **load** (used to load emulation memory from a host computer file)

echo



The **echo** command allows you to display ASCII strings or the results of evaluated expressions on the standard output device.

The parameters are as follows.

<STRING> Any set of ASCII characters enclosed between single open quote marks (‘), or double quotes ("). Because the command buffer is limited to 256 characters, the maximum number of characters in a string is 248.

Many keyboards (and printers) actually represent the single open quote mark (ASCII 60 hexadecimal) as an accent grave mark. The correct character in any case is the one encoded as ASCII 60 hexadecimal. The correct double quotation mark is ASCII 22 hexadecimal.

A character which is used as a delimiter cannot be used within the string. For example, the string **"Type "C""** is incorrect and will return an error. The string **‘Type "C"‘** is correct.

<EXPR> A valid expression (refer to the expression syntax pages for descriptions of valid expressions). The expression will be evaluated and the result will be echoed. Note that no delimiters are used to define the start and end of the expression.

<nn> “nn” is the hex code for any valid ASCII character. More than one character can be echoed with a single command; each “nn” must be preceded by a backslash. A total of 62 ASCII characters can be represented within a single **echo** command.

This capability is particularly useful for sending non-displaying control characters to a terminal; see the examples below.

The default is to echo nothing.

Examples

To echo the string “Set S1 to OFF” to the standard output, type the following:

```
M> echo "Set S1 to OFF"
```

Alternatively, you could use the ASCII character evaluation capability to do the same thing by typing the following:

```
M> echo \53 \65 \74 \20 \53 \31 \20 \74 \6f \20 \4f \46 \46
```

A more useful application of the backslash option is to send terminal control characters:

```
M> echo \1b "H" \1b "J" \1b "&dBSet S1 to OFF"
```

The above command sends “<ESC>H<ESC>J<ESC>&dB Set S1 to OFF” to the terminal. On an HP 2392A this homes the cursor, clears the screen, sets the video mode to inverse video, and writes the message “Set S1 to OFF.” Therefore, the user would see the message “Set S1 to OFF” in inverse video at the upper left hand corner of an otherwise blank screen. You might combine this with a macro command as part of a procedure. For example:

```
M> mac PROMPT={echo "Set S1 to OFF";w}
M> PROMPT
```

Calculate the value of the expression (1f + 1e):

```
M> echo 1f+1e
```

You must enclose strings in single open quote marks (‘) (ASCII 60 hex) or double quotation marks (") (ASCII 22 hex). A string not enclosed in delimiters will be evaluated as an expression and the result will be echoed. In addition, you may supply a backslash with a two digit hex constant; the corresponding ASCII character(s) will be echoed.

Echoing strings or ASCII characters is particularly useful within macros, command files, and repeats where you wish to prompt the user to perform some action during a “wait for any keystroke” command (see syntax for **w**). The expression capability is useful as a quick calculator.

Note that all options may combined within the same echo command as long as they are separated by spaces.

When using **echo** to calculate results of expressions, remember that all operations are carried out on 32-bit two's complement signed integers. Results greater than 32 bits are truncated.

See Also

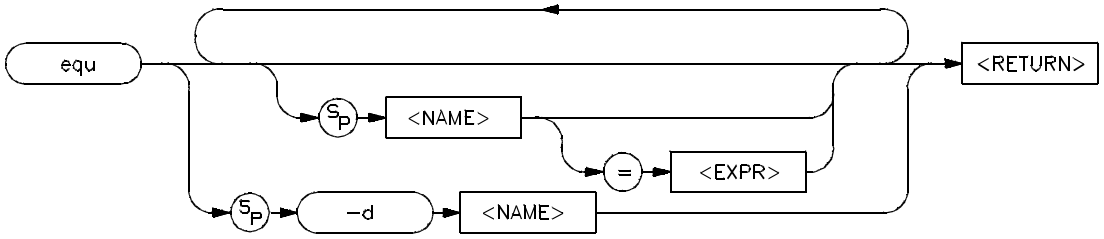
expr (details on what constitutes valid expressions)

mac (grouping a set of commands under a label for later execution)

rep (grouping a set of commands for immediate repetition)

w (wait command, allows user specified delays)

equ



The **equ** command allows you to equate arithmetic values with names that you can easily remember; these names can then be used in other commands to reference the value. This is useful in defining trigger patterns for the analyzer and in other applications.

The parameters are as follows.

<NAME> You use <NAME> to assign a character string to the expression. <NAME> must be an alphanumeric designator no greater than 31 characters in length, beginning with an alpha character or underscore and including only alphanumeric characters or underscores thereafter. If <NAME> is specified without an expression, then the existing definition for that name is displayed. If <NAME> is specified as *****, and the **-d** option is not given, then the definitions for all equates is displayed. However, if **-d** is supplied, then the equate table is cleared.

<EXPR> An arithmetic expression to be assigned to <NAME>. The default is a hexadecimal number. See the <EXPR> syntax pages in this manual for further details.

-d The **-d** option allows you to delete an existing equate. If you specify **-d** and <NAME>, then the named equate is deleted. If <NAME> is given as *****, then all equates are deleted.

68020/68EC020 Equates

Name	Value	Description
berr	0xxxxx0xxxxxxxxxxy	Bus error cycle.
byte	0xxxxxxxxx01xxxxxy	Byte transfer request (SIZ0/SIZ1).
cpu	0xxxxxxxxxxxx111xy	Function code cpu space.
data	0xxxxxxxxxxxxx01xy	Function code data space.
fgd	0xxxxxxxxxxxxxxx1y	Foreground memory cycle.
long	0xxxxxxxxx00xxxxxy	Longword transfer request (SIZ0/SIZ1).
mon	0xxxxxxxxxxxxxxx0y	Emulation monitor cycle.
prog	0xxxxxxxxxxxxx10xy	Function code program space.
read	0xxxxxxxxxxx1xxxxy	Read cycle.
rerun	0xxxx00xxxxxxxxxxy	Retrying a previous bus cycle.
sup	0xxxxxxxxxxxx1xxxy	Function code supervisor space.
supdata	0xxxxxxxxxxxx101xy	Function code supervisor data space.
supprog	0xxxxxxxxxxxx110xy	Function code supervisor program space.
tbyte	0xxxxxx10xxxxxxxxy	Cycle terminated as byte.
three_byte	0xxxxxxxxx11xxxxxy	Three byte transfer request (SIZ0/SIZ1).
tlong	0xxxxxx00xxxxxxxxy	Cycle terminated as long word.
tword	0xxxxxx01xxxxxxxxy	Cycle terminated as word.
user	0xxxxxxxxxxxx0xxxy	Function code user space.
userdata	0xxxxxxxxxxxx001xy	Function code user data space.
userprog	0xxxxxxxxxxxx010xy	Function code user program space.
word	0xxxxxxxxx10xxxxxy	Word transfer request (SIZ0/SIZ1).
write	0xxxxxxxxxxx0xxxxy	Write cycle.

68030/68EC030 Equates

Name	Value	Description
async16	0xxxxxx011xxxxxxxy	Asynchronous word transfer.
async32	0xxxxxx001xxxxxxxy	Asynchronous long word transfer.
async8	0xxxxxx101xxxxxxxy	Asynchronous byte transfer.
berr	0xxxx10xxxxxxxxxxy	Bus error cycle.
burst0	0xxxxxx000xxxxxxxy	First long word of burst cycle.
burst1	0xxxxxx010xxxxxxxy	Second long word of burst cycle.
burst2	0xxxxxx100xxxxxxxy	Third long word of burst cycle.
burst3	0xxxxxx110xxxxxxxy	Fourth long word of burst cycle.
byte	0xxxxxxxxx01xxxxxy	Byte transfer request (SIZ0/SIZ1).
cpu	0xxxxxxxxxxxx111xy	Function code cpu space.
data	0xxxxxxxxxxxxx01xy	Function code data space.
fgd	0xxxxxxxxxxxxxx1y	Foreground memory cycle.
logical	0xx0xxxxxxxxxxxxy	Logical memory address.
long	0xxxxxxxxx00xxxxxy	Longword transfer request (SIZ0/SIZ1).
mon	0xxxxxxxxxxxxxx0y	Emulation monitor cycle.
physical	0xx1xxxxxxxxxxxxy	Physical memory address.
prog	0xxxxxxxxxxxx10xy	Function code program space.
read	0xxxxxxxxxx1xxxxy	Read cycle.
retry	0xxxx00xxxxxxxxxxy	Retrying a previous bus cycle.
sup	0xxxxxxxxxxx1xxxy	Function code supervisor space.
supdata	0xxxxxxxxxxx101xy	Function code supervisor data space.
supprog	0xxxxxxxxxxx110xy	Function code supervisor program space.
sync	0xxxxxxxx0xxxxxxxy	Synchronous long-word transfer.
tablewalk	0xxxxxxxxxxxxxxxxy	Searching through translation tables.
three_byte	0xxxxxxxxx11xxxxxy	Three byte transfer request (SIZ0/SIZ1).
user	0xxxxxxxxxxx0xxxy	Function code user space.
userdata	0xxxxxxxxxxx001xy	Function code user data space.
userprog	0xxxxxxxxxxx010xy	Function code user program space.

word	0xxxxxxxxx10xxxxxy	Word transfer request (SIZ0/SIZ1).
write	0xxxxxxxxxxx0xxxxy	Write cycle.

If no parameters are specified, then the current table of all equates is displayed. If <NAME> is specified, then only the equate for that particular name is displayed.

Examples

You can predefine some equates to make it easier to set up the analyzer and run specifications. For example, suppose you want to take five traces of the demo program, with the trigger at address demo:Loop. You would like to have each trace numbered.

Enter the following commands:

```
M> tg addr=demo:Loop
M> equ c=0
M> mac numtrclist={t;w -m;equ c=c+1;echo "trace # "
c;tl}
M> r
M> rep 5 numtrclist
```

You will see five trace lists, each sequentially numbered, displayed on screen. You could use this feature in combination with a host logging program or redirection of your terminal display to printer to continuously monitor operation of a system. (To further aid your troubleshooting, you could also display the date and time of each trace sample using the **dt** command.)

You can remove equates from the table either individually or all at once:

```
M> equ -d c
M> equ
```

Notice that the equate for the name **start** has been removed. Now type:

```
M> equ -d *
```

This removes all equates, including the system-defined equates.

Multiple equates may be defined on the same command line, separated by a space.

Each equate is translated to its actual value at the time of command entry. For example, if you specify an equate **count=21h**; and an expression **start=2000h**, then the command **tg addr=start count** will be entered into the system as **tg addr=start 33**. At this point, redefining the value of **addr** or **count** would not change the address expression or the occurrence counter for the trigger.

The HP 64747/HP 64748 emulators predefine some equates that equate names to certain processor status bit patterns. You should be careful not to delete these equates because they are useful in specifying analyzer trace qualifiers.

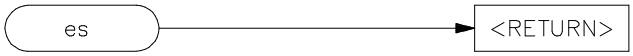
The combination of a single **equ** command with all names and expressions cannot exceed 255 characters. The number of equates and symbols that may be defined is limited only by available system memory; thus, it is dependent on the number of macros defined and on any emulator control code loaded by a high level software interface for the emulator (such as the HP 64700 PC Interface).

See Also

tg, tpat, tif, telif, and others. (**equ** provides an easy way to name expressions to use in setting up trigger or branch conditions)

r, m, bp (equates may be used to specify run addresses, memory addresses, or breakpoint addresses)

es



The **es** command displays the current status of emulation activity. It has no parameters.

Examples

View the emulator status:

```
M> es
```

The following types of information may be displayed:

- processor status—running/in monitor/reset
- slow bus cycle
- slow clock
- emulation halted due to halt input from target system or output from processor
- emulation in "wait" state due to input signal (ready, sync, DTACK) from target system
- emulation in monitor due to bus grant to the target system

The exact messages and information displayed varies slightly, depending on the emulator in use.

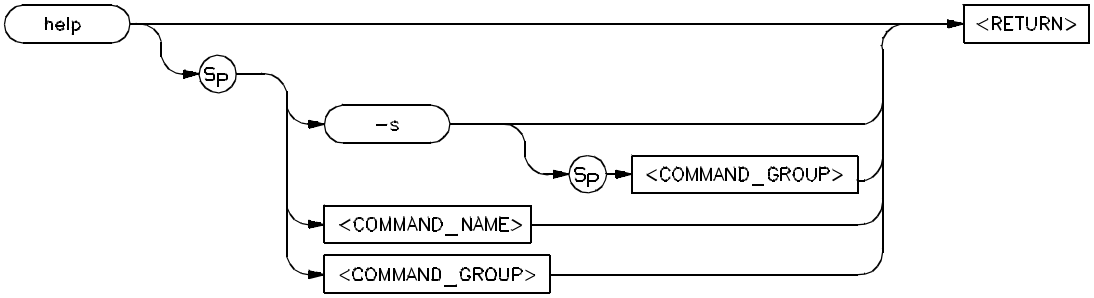
The emulator will not break to the monitor to obtain information. Therefore, any information that can only be obtained while in the monitor will not be displayed if the emulator is not in the monitor.

See Also

ta (allows you to display activity on emulation-bus analyzer lines)

ts (allows you to display the current trace status of the emulation-bus analyzer)

help,?



The **help (?)** command lets you display syntax, description and examples for any HP 64700 emulator Terminal Interface command. You may display a brief description for anything from a single command to command groups or the entire command set. Detailed information is available for single commands.

You may enter a question mark **?** instead of typing help; it performs the same function.

The parameters are as follows.

-s — This option switches in the abbreviated help mode; only the expanded name of each command is displayed next to the command.

<COMMAND_NAME> — If the name of an individual command is specified, only the detailed help information is displayed for that command.

<COMMAND_GROUP> — Specifying the name of a command group lists the commands available within that group.

If you specify "*" for <COMMAND_NAME> or <COMMAND_GROUP>, information for all commands will be displayed.

The **help** command without any parameters provides a list of command groups.

Examples

Display general help information listing the command groups and information regarding the use of the **help** command:

```
M> help
```

help,?

Display the short version of the help listing:

```
M> ? -s
```

Display the same listing of commands for only one of the command groups:

```
M> help -s emul
```

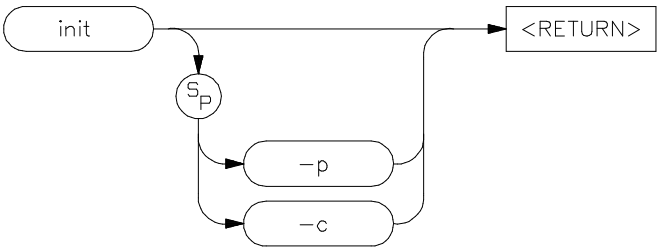
Display more information about each of the available memory commands by leaving out the **-s** flag:

```
M> help emul
```

Display specific information for the **m** command:

```
M> help m
```

init



The **init** command allows you to reinitialize the emulator. Powerup, complete, and limited initializations are available through various options.

The parameters are as follows.

-p The **-p** option causes a powerup initialization sequence. This initializes the operating system, data communications, emulation and analyzer boards, and runs extensive performance verification.

-c The **-c** option causes a complete initialization sequence. Everything is initialized as defined by the powerup sequence with the exception of the performance verification.

Examples

Perform a powerup initialization sequence:

```
m> init -p
```

You will see:

Perform a complete initialization sequence, which resets the entire emulator without executing performance verification:

```
m> init -c
```

Perform a limited initialization sequence, resetting only the emulator and analyzer:

```
m> init
```

You should only use the **init** command if the emulator is not responsive to other commands. If you wish to change other configuration parameters without initializing the emulator, there are commands available for that purpose. (See below.)

If no options are specified, a limited initialization sequence is performed. The operating system and data communications are not affected but all of the emulation and analysis boards are reset. For example, a limited initialization would not change macro definitions, system date and time, or the data communications parameters, but the emulation memory map and breakpoint list would be reset to their default states.

The **init -c** and **init -p** commands cause a loss of system memory. If these commands are used in macros, commands that follow them will not be executed.

See Also

cf (change emulation configuration)

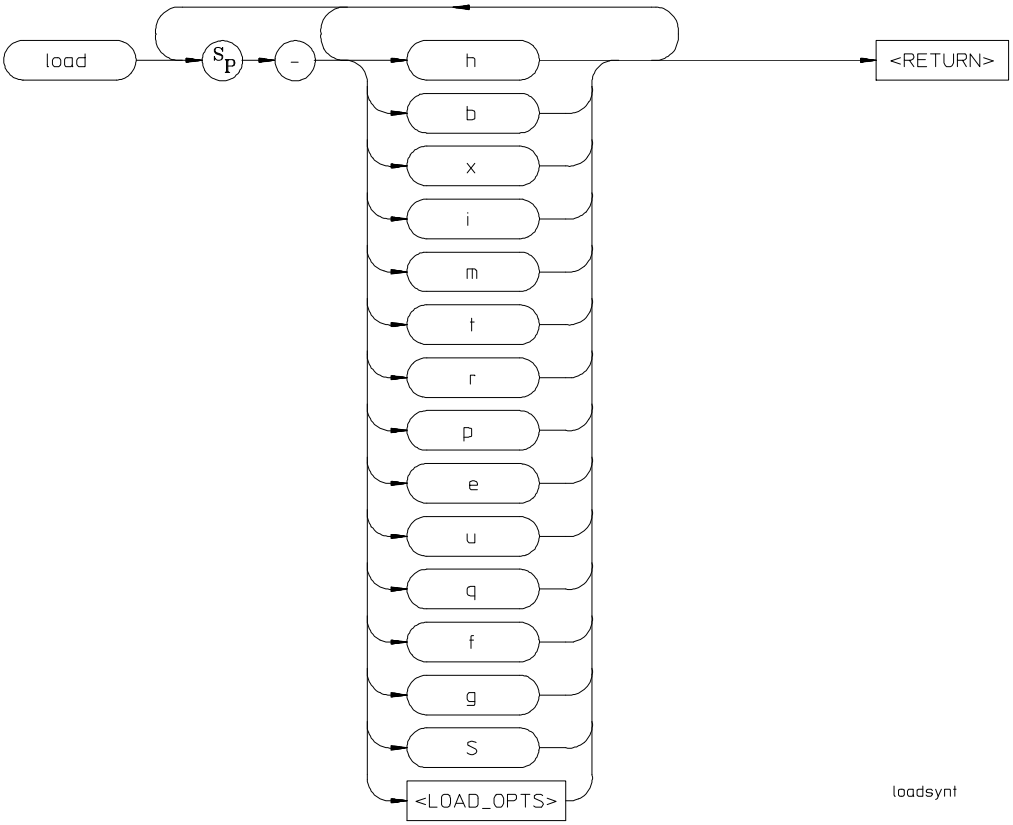
dt (set system date and time)

map (define the emulation memory map)

stty (set data communications parameters)

tinit (reset the analyzer to powerup defaults)

load



The **load** command lets you load program code into emulation or target memory. Various file formats are supported via options to the load command.

The parameters are as follows. At least one dash (-) must be included before any parameters are specified. It is optional to include or omit dashes for succeeding parameters.

-i Specifies that the program code will be in Intel hex file format.

-m Specifies that the program code will be in Motorola S-record file format.

-t Specifies that the program code will be in Tektronix hex file format.

-h	Specifies that the program code will be in HP file format. In this case, the file is expected to be transferred using the HP 64000 Hosted Development System **transfer** protocol.
-e	Load only those portions of program code which would reside in memory mapped to emulation memory space. (Refer to the **map** command.)
-u	Load only those portions of program code which would reside in memory mapped to target memory space. (Refer to the **map** command.)
-q	The program code will be transferred in quiet mode. If **-q** is not specified, the emulator controller will write a "#" to the standard output for each record successfully received and processed.
-r	The foreground monitor will be reloaded into dual-port memory.
S	This allows you to download a symbol file from the host computer into the emulator. This option is valid for HP 64700 emulators that support the use of symbols.
<LOAD_OPTS>	This represents all options to the **load** command that are specific to a particular HP 64700-Series Emulator. The MC68020 and MC68030/EC030 emulators do not support any custom load options.
-b	When using the HP file format, the program is expected to be in binary.
-x	When using the HP file format, the program is expected to be in hex.
-p	When using Intel, Motorola or Tektronix file formats, this option sets up a protocol checking scheme using ASCII **ACK**/**NAK** characters. If using this option, the host should send one record at a time and wait for the emulator to return an ASCII **ACK** character between records. If the emulator returns an ASCII **NAK** instead, there has been an error in data transmission. When the emulator receives the EOF character, it will return only the normal emulator prompt because data transmission is complete. If, during the transfer, the host receives a **NAK** for a record, it should retransmit the record until an **ACK** is received or until a timeout value is reached, whichever occurs first.
-f	You specify the **-f** option if you are loading a custom foreground monitor into the emulator. In the default, at least one file format option must be specified.

The destination of the program code is determined by the information contained in the program file. Additional options allow you to load only target memory or emulation memory as desired.

If a load error occurs, the current load procedure is aborted. However, records which were successfully loaded will remain in memory.

For the MC68020 and MC68030/EC030 emulators, the function code information in the program file must conform to the specifications of the emulation memory mapper. For information on specifying emulator function codes, see the <ADDRESS> syntax pages in Chapter 11. You should also refer to the manuals supplied with your assembler or high-level language to determine how those tools specify function codes for your processor.

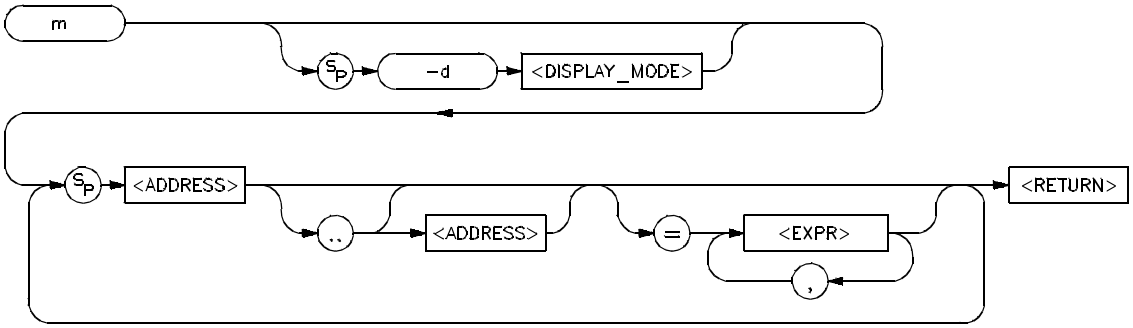
When you load an absolute file, the incoming data is examined for valid records (in the specified format). If the data being sent does not contain any valid records, the emulator will wait forever looking for valid records. The process must be terminated be entering a <CTRL>**c**.

See Also

dump (allows you to transfer emulation memory contents to a host)

See Chapter 4 for instructions on loading programs using different communications configurations.

m



The **m** command allows you to display and modify emulation and target system memory. Options allow you to specify the display mode, specific address or addresses for display or modification, and the data values to be inserted.

The parameters are as follows.

-d — The **-d** option allows you to set the display mode for memory accesses.

<DISPLAY_ MODE> — A one-character mnemonic specifying the display mode to use in creating memory displays. The allowable display modes are specific to the microprocessor in use; some typical modes are **b** (byte), **w** (word) and **m** (mnemonic). See the **mode** syntax pages to determine the correct display modes. If no display mode is specified, the global display mode set via the **mo** command is used as a default.

<ADDRESS> — Specifies the address to be displayed or modified. As noted in the syntax, an address followed by two periods and another address specifies a range of addresses to display or modify. Address notation is specific to each microprocessor. The MC68020 and MC68030/EC030 emulators allow the use of function codes in specifying address information. However, for all processors the address default representation is a hexadecimal number. See the **<ADDRESS>** syntax pages in Chapter 11 for examples of correct address specifications.

If you specify only the first address of a range followed by two periods and omit the second address of the range, 128 bytes of the range starting at the first address specified are selected for display or modification.

<EXPR> Data value to which a particular location is to be modified. If a range of locations is to be modified to a sequence of data values, the values must be separated by commas. Refer to the examples for details.

At least one address must be specified. If no display mode is specified the display mode set by the **mo** command is used. Data items specified in memory modification are repeated as a group to fill the address range specified (see the examples below for clarification). The memory <DISPLAY_MODE> defaults to the last value specified, or the default format for the emulator in use upon powerup initialization (varies dependent on the microprocessor being emulated).

Examples

Display the memory range f00 hex through f1f hex in byte format:

```
M> m -db 0f00..0f1f
```

Display the same address range in word format:

```
M> m -dw 0f00..0f1f
```

Display the range in long word format (32 bits):

```
M> m -dl 0f00..0f1f
```

Display memory contents as assembler mnemonics:

```
M> m -dm demo:Main..EndLoop
```

You can display several rows of memory at a time. Type:

```
M> m -db 700..7ff
```

Modify the contents of location 700 hex to the byte value 21 hex by typing:

```
M> m 700=21
```

Notice that the results of the memory modification are not automatically displayed. To view the results of a modification, you enter another **m** command.

Clear the contents of a memory range:

```
M> m 700..71f=00
```

Modify the contents of a range to some other hex value:

```
M> m 700..71f=21
```

Provide a sequence of data items for modification:

```
M> m 700..71f=41,42,43
```

If the selected address range for display or modification includes target system memory, the emulation processor will be broken to background upon execution of the command. After the command is complete, the processor will be returned to foreground execution if no errors occurred.

The method of specifying address information varies among different types of microprocessors. See the <ADDRESS> syntax pages in Chapter 11 for specific address information for the MC68020 and MC68030/EC030. Remember that specifying an address a particular way in one command will affect the way you need to specify it for all commands. For example, if you use function codes in specifying a memory map, you will also need to use function codes within the address information for the **m** command to display or modify those ranges of memory.

The way the data items are handled (for modification) depends on the <DISPLAY_MODE> in effect. For example, if the display mode is byte, and the data items 1a, 3f, and 66 are entered as 1a3f66, the location specified will be modified to 66 hex. If the display mode is word, the location will be modified to 3f66 hex. And if the display mode is long word, the location will be modified to 001a3f66. Note that data may be specified in decimal, octal, or binary in addition to the hexadecimal default. (See the **<EXPR>** syntax pages for information on specifying numeric bases.) Conversely, if you specify the value 33 hex for modification in byte mode, the value 33 is entered; in word mode, the value 0033 is entered; in long word mode, the value 00000033 is entered. In other words, if the value supplied is shorter than the mode in effect, it is padded with leading zeros.

In mnemonic mode, the instruction disassembler assumes that the first address location disassembled contains the first byte of an opcode; therefore, if you specify an address location that does not contain an opcode, the memory display will be incorrect.

The <DISPLAY_MODE> parameters depend on what modes are supported by the emulator. See the <MODE> syntax pages for details on supported display modes.

Display modes default to the last one specified. Therefore, if you would like to examine data areas after using the mnemonic display mode, you should change the mode.

When a sequence of data items is provided for memory modification, the sequence is repeated until the entire range has been modified.

If symbols have been defined, either by loading a symbol file or by using the **sym** command, these symbols can be used in the **m** command and will appear in the mnemonic mode (**-dm**) memory display. The command processor retains the name of the last module referenced. If a symbol does not contain a module name, the list of global symbols is searched. If the symbol is not found, the list of user symbols is searched. If the symbol is still not found, the system searches the last module referenced. If it doesn't find it there, the rest of the modules are searched.

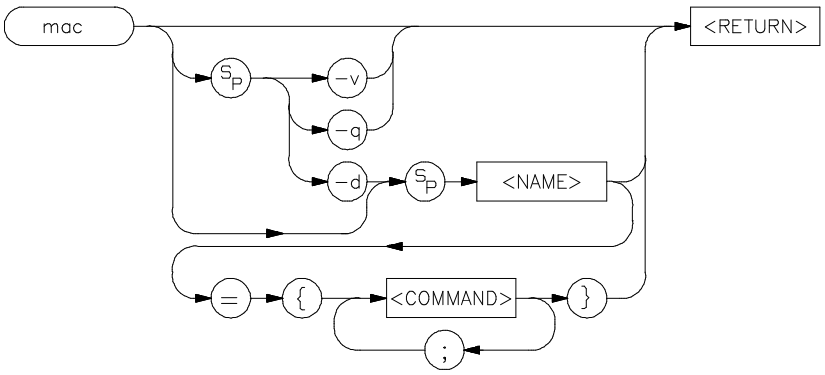
See Also

map (specify mapping of memory to emulation or user memory and to RAM or ROM)

mo (specify global access and display modes)

io (display modify I/O locations (for processors which support dedicated I/O))

mac



The **mac** command allows you to save a group of commands under a name of your choice. This allows you to instantly recall that command group by typing in the assigned name; the emulator will then preprocess the macro to expand the commands stored therein to a normal command line; the command line is then executed as usual.

The parameters are as follows:

-d — The **-d** parameter, in conjunction with the macro <NAME>, deletes the macro defined by <NAME>. If <NAME> is given as the character "*" then all macros are deleted.

<NAME> — This represents the name you assign to the macro definition. Names can be any combination of alphanumeric characters; however, you cannot define a macro that has a name identical to that of another HP 64700 Terminal Interface command.

If you specify a name which is the same as a currently defined macro, that macro will be overwritten by the new macro you define.

Certain HP 64700-Series emulators may predefine macros to aid you in setting up configurations for certain emulation tasks, such as in-circuit emulation.

<COMMAND> — This represents one or more emulator commands, including names which are used to define other macros. <NAME> and <COMMAND> must be separated by an equal sign (=), and the command string must be enclosed with braces "{ }." Each <COMMAND> must be separated from other commands by a semicolon (;).

When using command substitution, you can include pseudo-parameters in the form of "&token&" in the macro definition. Do not include any white space between the two "&" symbols. When you execute the macro, include the string to be substituted for &token& as a parameter on the command line. The macro will execute using the command expanded with the string you substituted. See the Examples section for more information.

`-q` This option sets the macro expansion echo to quiet mode. In this mode, any macro that you run will be executed without displaying the expanded command string.

`-v` This option sets the macro expansion echo to verbose mode. In this mode, any macro that you run will first display the expanded command string as a comment, and then will execute the macro.

If no parameters are supplied, the current set of macro definitions is displayed. If only <NAME> is supplied without a command string, the macro defined by <NAME> is displayed.

Examples

Define a macro that resets the emulator, then defines the memory map, resets the processor and breaks into the monitor, and then sets up the stack pointer:

```
M> mac setup={init;map 0..7fff eram;rst -m;reg usp=7000}
```

To execute the command, type:

```
M> setup
```

You could define another macro called "echonwait" as follows:

```
M> mac echonwait={echo "Set S1 to OFF";w}
```

Delete the macro named **setup**:

```
M> mac -d setup
```

Delete all macros:

```
M> mac -d *
```

Define a macro that fills an arbitrary 100-byte block range with a user-defined value:

```
M> mac fill={equ start=&address&;m -db
start..start+100t=&value&}
```

Invoke the macro:

```
M> fill  50  88
```

In this example, 50 will be substituted for &address&, and 88 will be substituted for &value&. So, addresses 50 through 150 decimal will contain the value 88.

Nested macro calls are permitted and limited only by constraints of system memory.

The commands within the macro definition are not checked for correct syntax until the macro is executed; therefore, it is advisable to test the command string before defining the macro.

The number of macros that can be created is limited to 100, but may be less, depending on the complexity of the macros defined.

The length of the macro name combined with the macro definition is limited only by the maximum HP 64700 command length of 255 characters; thus, the macro name and definition can be a maximum of 251 characters.

A command within a macro definition cannot contain the pound sign character (#) unless the command is enclosed in a quoted string. (Otherwise, text following the # is interpreted as a comment.) This means there can be no matching brace at the end of the command. Use the **echo** command to place comments in a macro definition.

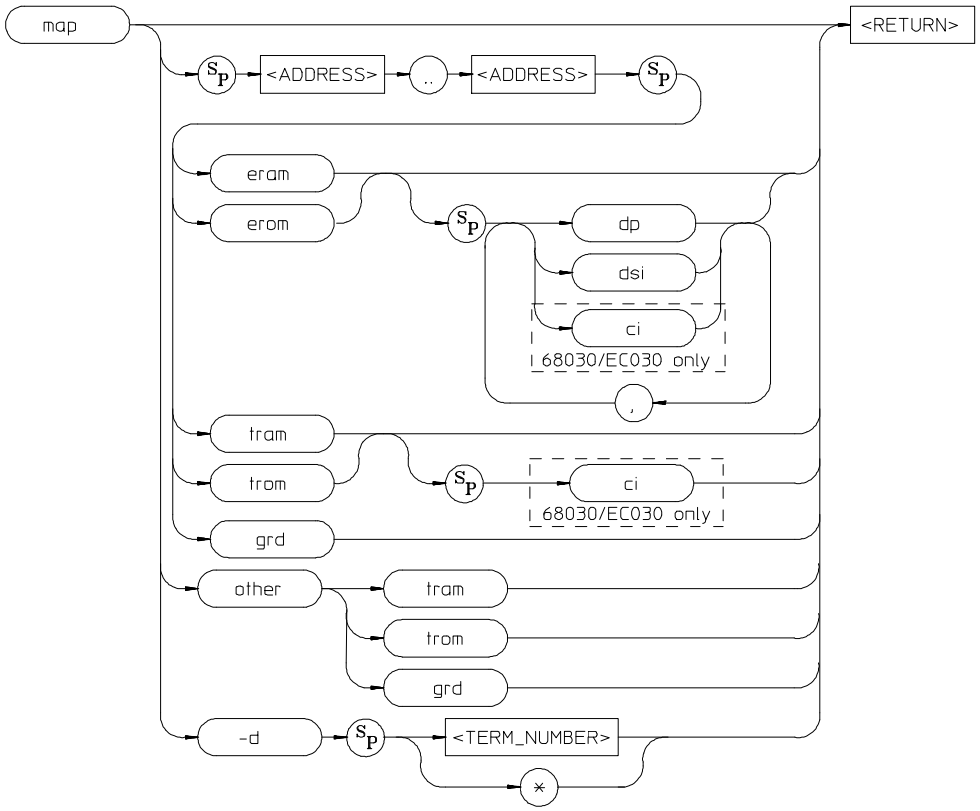
Command line substitution is possible when invoking a macro. During the macro definition, you may include pseudo-parameters which allow you to substitute parameters, such as file names, when invoking the macro.

Pseudo-parameters are replaced on a position-dependent scheme, where the first pseudo-parameter encountered in the macro string is replaced with the first parameter passed into the macro. The second pseudo-parameter is replaced with the second parameter passed into the macro, and so on.

You can define multiple pseudo-parameters in a macro using the same name for both (or all) of them. Because pseudo-parameters are position-dependent, the first pseudo-parameter will always be substituted with the first parameter you pass into the macro, the second pseudo-parameter with the second parameter you pass into the macro, and so on.

See Also **rep** (repeat; allows you to repeat any command, including macros)

map



The **map** command allows you to map address ranges to one of five different classes of memory. For example, you may want to specify that addresses 1000 through 2fff hex are in emulation RAM, and addresses 3000 through 3fff hex (where your program code will reside) are in emulation ROM. Later, when your target system hardware is prototyped, you will be able to easily modify these specifications to indicate that the address ranges actually reside in target system RAM or ROM.

The parameters are as follows.

<ADDRESS> The address values specify the address range to be assigned to a particular memory type. Whenever the emulation processor accesses the range specified, it will be directed to the memory type specified in the map. Specification of address

information defaults to a hexadecimal value. The MC68020 and MC68030/EC030 emulators also allow specification of function codes. See the **<ADDRESS>** syntax pages in Chapter 11 for details of address specification.

other — The address range **other** specifies all address ranges not otherwise specified by mapper terms. The MC68020 and MC68030/EC030 emulators restrict type definition of the "other" range to **trom**, **tram**, or **grd**.

eram — Specifying **eram** indicates that the given address range is to reside in emulation address space and act as RAM (read/write).

erom — Specifying **erom** indicates that the given address range resides in emulation address space; it is to act as ROM (read only). The **bc** command allows you to specify that emulation processor writes to this space or to space designated as target ROM (**trom**) will cause an emulation system break.

The emulator protects emulation memory from being modified when a write to emulation ROM occurs. (This feature may not be supported in future HP 64700-Series emulators.)

tram — Specifying **tram** indicates that the given address range lies within target system RAM space. When the emulation processor accesses an address within this range, the target system data buffers will be enabled by a mapper signal to complete the transaction.

trom — Specifying **trom** indicates that the given address range lies within target system ROM space. As with the **erom** parameter above, the **bc** command may be used to set up the emulation system to break upon a write to these address ranges. In any case, if target ROM memory is actually implemented as RAM, and the necessary write strobes are connected to this memory, the emulator will allow the processor to overwrite the memory locations.

grd — The **grd** parameter indicates the given address range is to be “guarded;” therefore, the emulation system software should not know that it exists. An emulation system break will always be generated upon accesses to guarded memory.

dp — Use the 4 Kbyte block of dual-port emulation memory. Valid only for erom and eram.

dsi — Interlock target system and emulation DSACKs (only valid for erom and eram blocks.

ci — Inhibit caching for this memory block (only valid for the MC68030/EC030 emulator.)

If the command **map** is entered with no parameters, the current memory map is displayed.

Examples

View the memory map:

```
M> map
```

Suppose that you need to map the following ranges:

- 1000 through 1fff to supervisor program space, using dual-port emulation RAM and interlocking DSACKs with the target system.
- 2000 through 2fff to user space using emulation RAM and interlocking DSACKs with the target system.
- 5000 through 7fff to user data space using target RAM and inhibiting caching.
- 8000 through 8fff to user program space using target ROM.
- All other memory is mapped as guarded.

Implement this map by entering

```
R> map 1000..1fff@sp eram dp,dsi
R> map 2000..2fff@u eram dsi
R> map 5000..7fff@ud tram ci
R> map 8000..8fff@up trom
```

Delete all map terms (reset the map):

```
R> map -d *
```

The emulation system assigns a term number to each address range specified by you in the map command. Term numbers are assigned in ascending order of address range. Therefore, if you map the addresses 0 through 100 (TERM_NUMBER_1) and 1000 through 1fff (TERM_NUMBER_2), then specify another range of 300 through 3ff, TERM_NUMBER_2 will be renumbered as TERM_NUMBER_3 and the range 300 through 3ff will become TERM_NUMBER_2. Remember to use the assigned term number when specifying mapper terms to be deleted by the **map -d <TERM_NUMBER>** command.

The memory mapper reassigns blocks of emulation memory after the insertion or deletion of mapper terms. For example, if you modified the contents of 300 through 3ff above, deleted TERM_NUMBER_1, and displayed locations 300 through 3ff,

you would notice the contents of those locations are not the same as they were before deleting the mapper term.

The mapper address block resolution for the MC68020 and MC68030/EC030 emulators is 256 bytes. However, the block sizes for target memory and emulation memory on a particular emulator are identical. If an address range smaller than a multiple of the block size is entered as a map specification, the range is rounded upwards to the nearest block size multiple.

When any map term is added or deleted the emulation processor will be reset and held in the reset state until a break or run command is issued. The processor remains reset in recognition of the fact that returning to execution directly after mapper modification is most likely invalid.

Be sure to disable all breakpoints (**bc -d bp**) before changing the map. Breakpoints are not cleared when the memory map is changed. (Breakpoints are also not cleared when a file is loaded, or when memory is manually modified.) After the new map and the program are set up, you can re-enable the breakpoints by re-enabling the breakpoints break condition (**bc -e bp**) and entering the **bp -e *** command. When the list of breakpoints is displayed (**bp**), the memory is checked to verify whether the breakpoint is still in memory.

If all mapper terms are deleted with the command **map -d ***, the "other" range is unaffected.

See Also

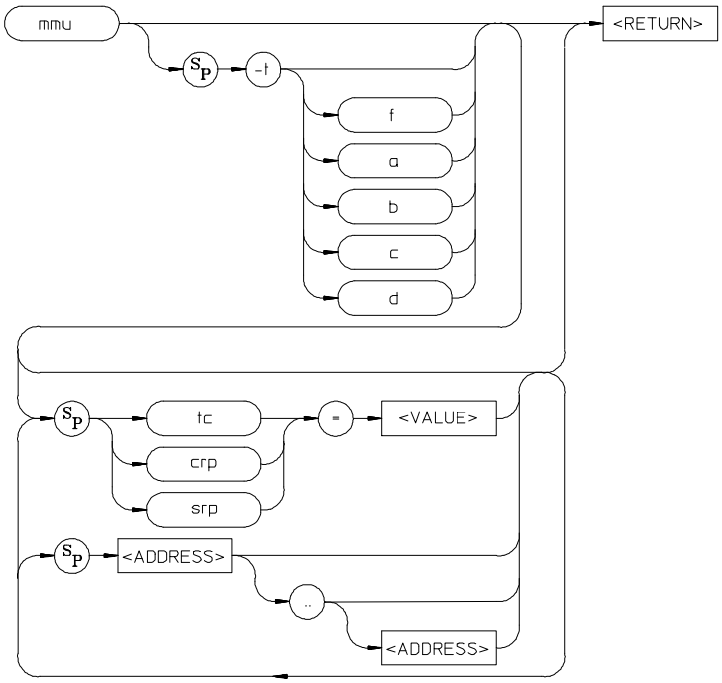
bc (break conditions; determines whether emulator breaks to monitor upon write to space mapped as ROM)

m (memory display/modify)

bp (set/delete software breakpoints)

Chapter 7, "Configuring the Emulator," has a complete description of the block allocation strategy used for the MC68020 and MC68030/EC030 emulation memory resources.

mmu



The **mmu** command is used to display valid logical-to-physical address translations. You can display all of the present translations for all logical addresses, or for only a limited range of logical addresses. Further, you can display the details of how a single logical address is mapped through the tables to its corresponding physical address. Finally, you can display the details of a single translation table used by a selected logical address.

You can use the **mmu** command to view the present set of valid translations, even when the TC register and the root pointer registers are invalid. Parameters in the **mmu** command let you specify values to use when none exist in these MMU registers.

The parameters are as follows:

a	Shows the content of Table A for the logical address you included in your command.
<ADDRESS>	The address or address range specifies a logical address reference for the MMU information to be displayed.
b	Shows the content of Table B for the logical address you included in your command.
c	Shows the content of Table C for the logical address you included in your command.
crp	Lets you specify a value to be used in place of the present content of the CRP (CPU root pointer) when reading the tables and showing the address mappings.
d	Shows the content of Table D for the logical address you included in your command.
f	Shows the content of the function-code table for the logical address you included in your command.
srp	Lets you specify a value to be used in place of the present content of the SRP (supervisor root pointer) when reading the tables and showing the address mappings.
t	Shows the details of the translation through the tables for the logical address you included in your command.
tc	Lets you specify a value to be used in place of the present content of the TC (translation control) register when reading the tables and showing the address mappings.
<VALUE>	A number to be used in place of the present value within the referenced MMU register (TC, CRP, or SRP). This number does not overwrite the present content of the register.

When **mmu** is used by itself, it shows a list of the valid MMU mappings. One entry in the list is allocated for each page in the system.

Examples

Show all of the valid logical-to-physical mappings in the MMU:

M> **mmu**

Show all of the logical-to-physical mappings for logical addresses in the range of 7FF0 through 800F:

M> mmu 7ff0..800f

Show the table details used to translate logical address 400:

M> mmu -t 400

Show the details of Table A used to translate logical address 40FC.

M> mmu -ta 40fc

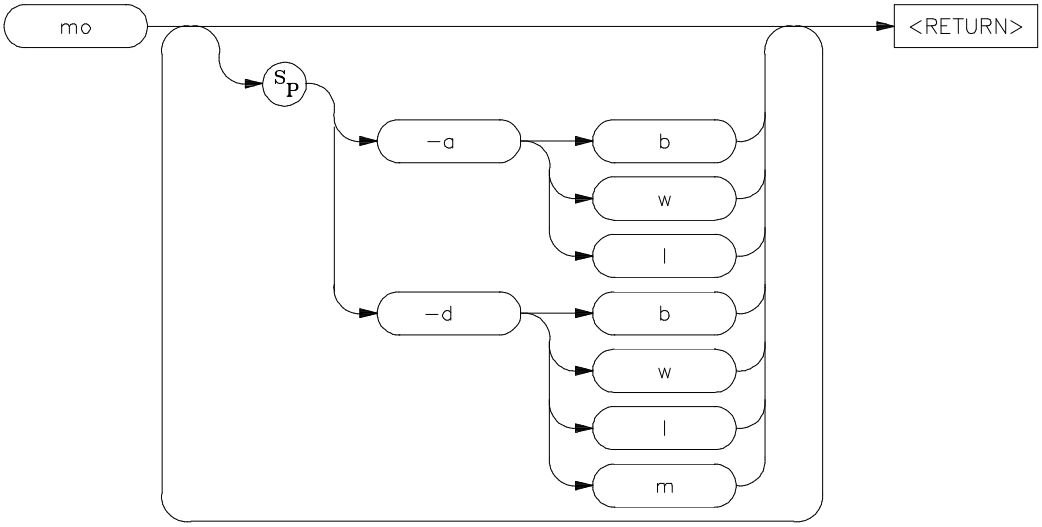
Show the present MMU mappings based on a TC register value of 81FF2000 instead of the present TC register value:

M> mmu tc=81ff2000

See Also

dmmu (Controlling the deMMUer.)

mo



The **mo** command allows you to modify the global access and display modes. Access mode is defined as the type of processor data cycles used by the emulation monitor to access a portion of user memory. Display mode is defined as the method used to display or modify data resident in memory.

The parameters are as follows:

-a — The **-a** parameter, in combination with a single character specifying mode type, sets the global access mode.

<ACCESS_MODE> — A single character used to specify the global access mode. Note that there is no space between the **-a** parameter and the mode specifier. The MC68020 and MC68030/EC030 emulators allow the following access modes:

- l long word (four bytes) access mode
- w word (two bytes) access mode
- b byte access mode

-d — The **-d** parameter, in combination with a single character, sets the global display mode default.

<DISPLAY_MODE> — A single character used to specify the global display mode default. Note that there is no space between the **-d** parameter and the mode specifier. The MC68020 and MC68030/EC030 emulators allow the following display modes:

l	long word (four bytes) display mode
w	word (two bytes) display mode
b	byte display mode
m	mnemonic display mode

If no parameters are specified, the current settings of the display and access modes are displayed.

Examples

To display locations Main through EndLoop in the demo program in mnemonic format, enter:

```
M> m -dm demo:Main..EndLoop
```

For other examples of the effects of changing the display mode, see the syntax pages for the **m** (memory) command in this manual.

View the current settings of the access and display modes:

```
R> mo
```

Set the access mode to words:

```
R> mo -aw
```

Change the access mode to words and the display mode to long words:

```
R> mo -aw -dl
```

Change the access mode to words, and the display mode to mnemonics:

```
R> mo -aw -dm
```

Reset the access and display modes to the powerup defaults:

```
R> mo -ab -dw
```

The emulator allows you to display and access memory in several ways for memory display and modification. You set the display and access size using the **mo** command. There are two types of mode settings.

Display Mode

Display mode defines how the emulator displays or modifies memory.

The mnemonic display mode allows you to display memory disassembled into processor instruction mnemonics using the **m** command. If you specify mnemonic display mode and then execute any other command that references the display mode, the command will behave as if "byte" display mode was selected. (Such commands include memory modifies and searches.)

The **mo** command only sets the initial display mode. It is changed by using the mode option in any of the memory access commands (such as **m** or **ser**).

Access Mode

Access mode defines how the emulator accesses target system memory.

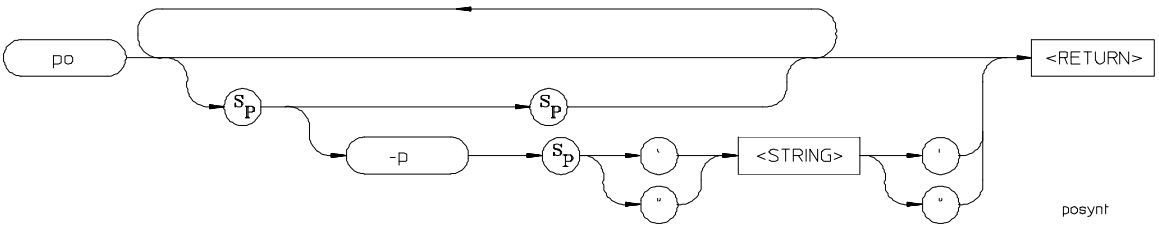
The emulation monitor uses the access mode to determine whether to use byte, word or long word instructions during accesses to target system memory and emulation memory that is not dual-port. It does *not* affect how that data is displayed on screen, or the way in which data is interpreted for memory modification. That is controlled by the display mode.

See Also

m (memory display/modify)

io (input/output display or modify)

po



The **po** command allows you to change the system prompt characters.

The parameters are as follows:

-p — The **-p** option allows you to change the emulator's command prompt to one specified by <**STRING**>.

<STRING> — <**STRING**> is any group of ASCII characters enclosed by single open quotes (‘) or double (") quote marks. This parameter, when used with **-p**, allows you to specify a new emulator command prompt.

Examples

If several people use the system, you may want to define macros which reset the prompt so each user knows who is currently using the emulator. For example:

```
M> mac yourid={po -p "\YOURID>"}
M> mac herid={po -p "\HERID>"}
M> mac hisid={po -p "\HISID>"}
```

You can redefine the emulator's command prompt string using the **po -p** <**STRING**> command. Upon powerup, the emulator prompt defaults to ">." (The character before the string, for example, **R**, **M**, **U**, etc., is used to indicate the current emulator status and is *not* affected by redefining the prompt string.)

pv

The **pv** command runs performance verification on the emulator and analyzer. The performance verification exercises all the emulator hardware and software to high confidence level.

The parameters are as follows:

<REPEAT_COUNT>

<REPEAT_COUNT> allows you to specify the number of times to repeat the performance verification. This is a required parameter.

If no parameters are given, a warning message about initialization of the emulator along with correct **pv** command syntax is displayed. To actually execute the **pv** command, you must provide a <REPEAT_COUNT> value.

Examples

Executing **pv** with no parameters provides a warning display, along with help for the correct syntax. Type:

```
M> pv
```

To loop through the performance verification twice, type:

```
M> pv 2
```

You should only run performance verification when the emulation probe is **not** plugged into a target system. You should also make sure to remove any conductive foam or plastic pin protectors from the emulator probe, as these will cause failures during performance verification.

When you use the **pv** command, the emulator is initialized as if power were cycled. Therefore, all equates, macros, memory map, configuration settings, system clock, software breakpoints, trace specifications, and other configuration items you have altered will be cleared. Do not use the **pv** command unless you can restore these items from a host, or have documented them so you can restore their states manually.

If **pv** reports failures, first check your hardware installation as documented in the manual. If the failures persist, call your local HP Sales and Service office for assistance. A list of offices is provided in the *Support Services* guide.

Note that providing multiple commands such as **pv 1;r** is invalid; the second command will not execute due to the system reset.

Typing in <CTRL>-C to abort the **pv** command may result in incorrect failure messages.

See Also **init** (reinitializes the emulator)

r

The **r** command starts an emulation run. Execution begins at the address specified by the <ADDRESS> parameter; if no address is specified, execution begins at the address in the program counter.

The parameters are as follows:

<ADDRESS> Specifies the address where execution is to begin. If you specify **$**, the processor runs from the current program counter value. If you specify **rst**, the processor runs from its reset address.

If no parameters are specified, the emulation run begins at the address specified by the processor's current program counter contents.

Examples

Load the demo program, run it, and then break to monitor:

```
R> demo;r;b
```

Run the processor from the Loop address in the demo program:

```
R> r demo:Loop
```

Although <ADDRESS> defaults to a hexadecimal number, the MC68020 and MC68030/EC030 emulators allow specification of function codes. See the <ADDRESS> syntax pages in Chapter 11.

See Chapter 4 for information on how the emulator handles an **r rst** command.

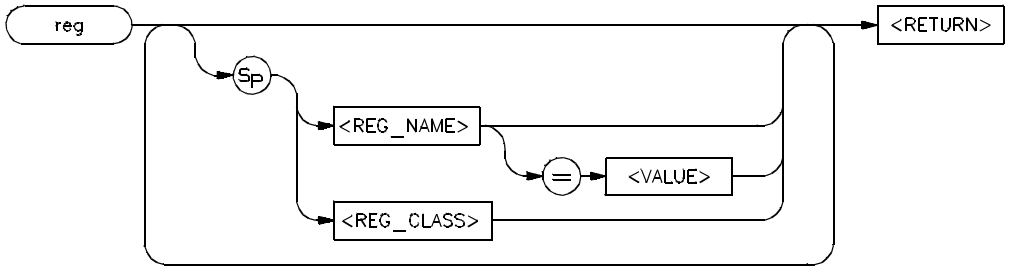
See Also

s (step; allows controlled stepping through program instructions)

rx (run only when CMB (Coordinated Measurement Bus) execute pulse is received)

x (pulse the CMB execute line if resident on the CMB)

reg



The **reg** command allows you to display and modify emulation processor register contents. Individual registers may be displayed or modified. Related groups of registers may be displayed. Combinations of display and modify are permitted on the same command line.

The parameters are as follows.

<REG_NAME> The <REG_NAME> parameter allows you to specify a single register to display or modify.

<REG_CLASS> The <REG_CLASS> parameter allows you to specify an entire group of registers for display.

<VALUE> To modify a register's contents, supply the new contents in the <VALUE> variable. This is a numeric value. The default is hexadecimal, other number bases may be specified. Floating point values cannot be used. Also, you cannot use symbols as the value for modifying the PC register.

Register Class	**Register Names**
*** (basic)**	**pc, st, usp, isp, msp, cacr, caar,d0..d7, a0..a7, vbr, dfc, sfc**
fpu (if your target system has a floating-point coprocessor)	**fpcr, fpsr, fpiar, fp0..fp7**
mmu (MC68030 only) **OR** **acu (MC68030/EC030)**	**tt0, tt1, mmusr, tc, srp, crp** **ac0, ac1, acusr**

Examples

View the contents of all registers:

```
M> reg
```

Modify the contents of register D0 to 50 hex:

```
M> reg d0=050
```

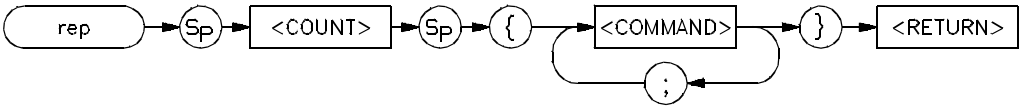
You can display more than one register at a time by listing all register names on the same line:

M> **reg d0 st**

See Also

s (step; allows you to step through program execution—in combination with the **reg** command, this is useful in debugging)

rep



The **rep** command allows you to repeat a group of commands a specified number of times. The command list is simply a group of valid HP 64700 commands separated by semicolons and delimited by braces.

The parameters are as follows.

<COUNT> An integer value specifying how many times to execute the command list. A count of zero is a special case, meaning "repeat forever" (the repetition can be stopped by entering <CTRL>-C, which issues a break to the emulator).

<COMMAND> Any valid HP 64700 Emulator command, including previously defined macros, may be specified with the options appropriate to the command. The list of commands must be preceded by an opening brace and followed by a closing brace. Also, the commands must be separated by semicolons. The commands will be executed in the same order as they are specified on the command line.

Both a count and at least one command must be specified.

Examples

Suppose that you're using an ANSI terminal and want to simulate a repetitive memory display of a certain memory range:

```
M> mac mem={echo \1b \5b \31 \3b \31 \48; m -db
4000..401f; w 0}
```

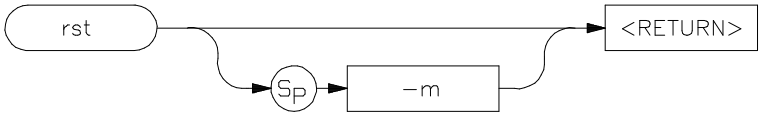
Command macros that you define using the **mac** command can be used within a command group for repetition.

No other command input will be accepted until the command group has executed the indicated number of repetitions.

See Also

mac (allows assignment of a name to a command group for easy recall of a specified command sequence)

rst



The **rst** command resets the emulation microprocessor. An option allows you to specify that the processor should begin executing the emulation monitor code immediately after the reset.

The parameters are as follows:

-m — Causes the emulator to begin executing monitor code immediately after the reset.

The default operation is to reset and remain in the reset state.

Examples

Reset the processor and keep it in the reset state:

```
M> rst
```

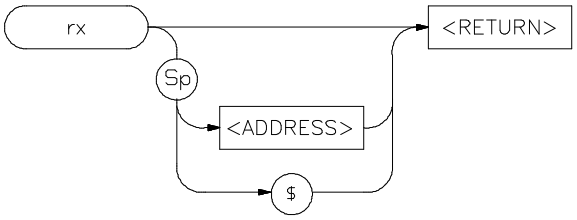
Reset the processor and have it immediately commence emulation monitor execution:

```
U> rst -m
```

If **-m** is not specified, the emulation processor remains in the reset state. Note that any commands which require the emulation processor to execute the monitor code for command processing will not execute while the processor is in the reset state; these include commands such as **reg**.

Commands or hardware signals which will take the emulator out of a reset state include **b**, **r**, **s**, and the CMB /EXECUTE pulse.

rx



The **rx** command allows you to set the starting address for synchronous CMB (Coordinated Measurement Bus) execution.

The parameters are as follows.

<ADDRESS>

The <ADDRESS> parameter specifies where to start program execution when the CMB EXECUTE pulse is detected. If $ is specified for address, the current program counter is used (default). The default base for <ADDRESS> is hexadecimal; other bases can be specified with the proper extension. (See the **expr** syntax pages for supported bases.) For the MC68020 and MC68030/EC030 emulators, you may also specify function codes. See the <ADDRESS> syntax pages in Chapter 11 for more information.

If you enter the **rx** command without any address parameters, the current address value setting is displayed. If no rx command has been entered since initialization of the emulator, then the default setting is **rx $**.

Examples

View the current address setting specified by **rx**:

```
M> rx
```

Begin execution at demo:Main when the CMB-EXECUTE pulse is received:

```
M> rx demo:Main
```

Start execution at the current value of the program counter when the CMB-EXECUTE pulse is received:

```
M> rx $
```

If the HP 64700 emulator is connected to the CMB, and the CMB-EXECUTE pulse is detected, followed by the CMB-READY line in the true state, the emulator will begin execution at the address specified by the **rx** command. If no **rx** command has been issued, execution begins at the current program counter value (same as **rx $**).

Execution will begin at the address specified by **rx** every time the conditions listed above are met. For example, if you type the command **rx 100**, the emulator will start executing at address 100 hex every time the CMB-EXECUTE line is pulsed.

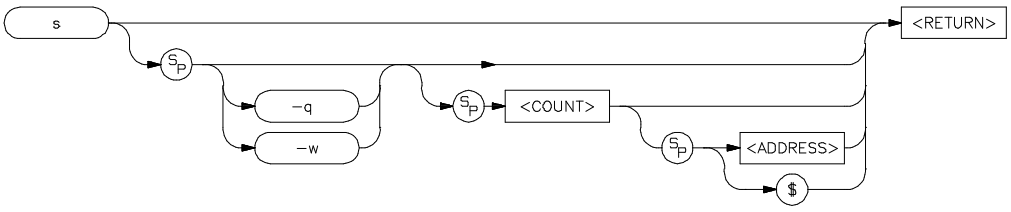
The **rx** command automatically turns on CMB interaction by effectively performing the equivalent of a **cmb -e** command whether or not you have done so.

See Also

cmb (enables or disables CMB interaction)

x (initiates a synchronous CMB interaction by pulsing the CMB-EXECUTE line)

s



The **s** command allows you to single-step the emulation processor through a program. You can specify the number of steps to execute at a single time; or, you can direct the emulator to step continuously. In addition, you may specify the starting address for stepping.

The parameters are as follows:

-q — If you enter the **-q** parameter, stepping will occur in quiet mode; that is, the instructions and program counter are not displayed upon execution of each step.

-w — If you enter the **-w** parameter, stepping will be done in whisper mode; only the final program counter value is displayed after the step is executed.

<COUNT> — The <COUNT> parameter allows you to specify the number of steps to execute in sequence before returning command control. For example, if you specify **s 5**, then five instructions will be executed in sequence.

The default base for <COUNT> is decimal. Other number bases may be specified; see the **EXPR** syntax pages for more information.

If you do not specify a value for <COUNT>, then a value of one (1) is assumed. If you specify a step count of zero (0), the emulator interprets this as "step continuously." Continuous stepping can be aborted with the <CTRL>-C command; or, it will be terminated upon receipt of an emulation break condition such as a write-to ROM.

<ADDRESS> — The <ADDRESS> parameter allows you to specify the starting address for stepping. The default is a hexadecimal value; see the **EXPR** syntax pages for information on specifying other number bases. The MC68020 and MC68030/EC030 emulators allow you to specify function codes as part of the address. See the <ADDRESS> syntax pages for more information.

If you specify **s** with no parameters, the processor is stepped one instruction from the current program counter location. If you specify <COUNT> but not <ADDRESS>, then the current program counter value is specified for <ADDRESS>.

Examples

Step one location from Main:

```
M> s 1 demo:Main
```

You can step all the way up to the READ_INPUT routine by typing:

```
M> s 2 demo:Main
```

You can also step through the program in "quiet" mode. This inhibits the display of any information about the stepping process. Type:

```
M> s -q 2 demo:Main
```

You can step the processor and display only the final program counter value after the step by using the "whisper" mode:

```
M> s -w 3 demo:Loop
```

Remember that you must specify a step count value if you specify an address. If you don't, a <CTRL>-C will abort the stepping. Type:

```
M> s 2000
```

The emulator will step 2000 times; enter **<CTRL>-C** to abort the step function.

You can assign values to label names using the **equ** command and then use these labels in specifying step information. For example, the number of instructions in the command input loop is 3. Type:

```
M> equ readcount=3
M> s readcount demo:Loop
```

If the emulator was in the run state (U> prompt) executing a user program when you request the step, it will break to the monitor program before executing the step.

When the Coordinated Measurement Bus (CMB) is being actively controlled by another emulator, the step command (**s**) does not work correctly. The emulator may end up running in user code (NOT stepping). Disable CMB interaction (**cmb -d**) while stepping the processor.

If you substitute **$** for the <ADDRESS> parameter, the current program counter value will be used as the <ADDRESS> value. The same will occur if no address parameter is specified.

If you specify a value for <ADDRESS>, then you must specify a value for <COUNT>. Otherwise, the address value will be interpreted as a step count; the emulator will step the number of locations specified.

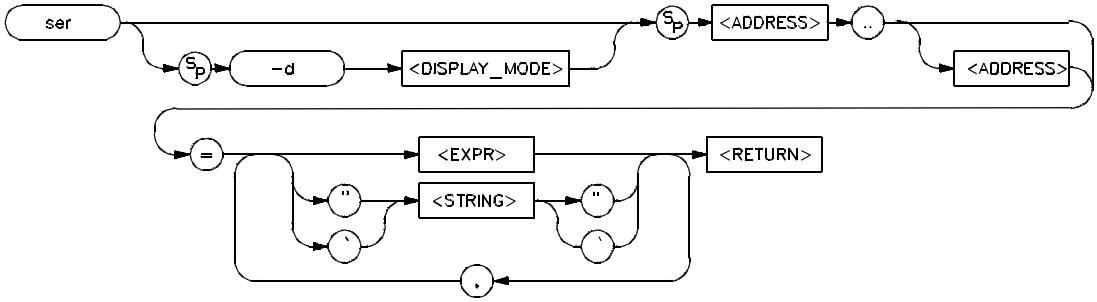
If you have loaded a symbol file or user defined symbols, you will see the module and symbol in the output when an instruction address has a corresponding symbol.

See Also

r (run emulation processor from a specified address)

reg (view or modify processor register contents)

ser



The **ser** command allows you to search memory for a data value, a character string, or a combination of both. For every pattern match, the starting address of the match is displayed.

The parameters are as follows:

-d — The **-d** operator, in combination with the <DISPLAY_MODE> parameter, allows you to specify the display mode used for the search. As a result, you can alter the method used by the system for interpreting the display list data and the resultant matches.

<DISPLAY_MODE> — This is a single character specifying the display mode to be used in the search. The MC68020 and MC68030/EC030 emulators support **b** (byte), **w** (word), and **l** (long word). For more information on the <DISPLAY_MODE> parameters, see the **mo** command.

<ADDRESS> — You use <ADDRESS> to specify first the lower, and possibly the upper, address boundaries of the memory range to search for the given data pattern. <ADDRESS> defaults to a hexadecimal number; expressions may also be provided. In addition, the MC68020 and MC68030/EC030 emulators allow you to specify function codes. See the <ADDRESS> pages in Chapter 11 for more details.

.. — The two periods (**..**) are used as a separator between the lower and upper address boundary specifications. Notice that no additional spaces are inserted. You can use "<ADDRESS>.." to specify the range from the address through the next 127 bytes.

<EXPR> — <EXPR> is a numeric expression to be used as a reference pattern in the search. The default is a hexadecimal number; other bases and expressions may be specified. See the <EXPR> syntax in Chapter 11 for more information.

<STRING>

You specify <STRING> if you want to search for an ASCII character pattern. Note that <STRING> must be bounded by single open quote marks (‘) or double quotes (").

Many keyboards (and printers) actually represent the single open quote mark ‘ as an accent grave mark. In any case, the correct key is the one which produces a character encoded as ASCII 60 hexadecimal. The correct double quote mark is the character encoded as ASCII 22 hexadecimal.

If the character string you are searching for contains double quotes, you must delimit the string with single open quotes and vice versa. For example, the string **"Type "C""** will return an error; the string **‘Type "C"‘** is correct.

At least one address range and data pattern must be specified. If no display mode is set with the **-d** option, the current global display mode from the **mo** command is used.

Examples

Search for the string “This” in the message area of the demo program:

```
M> ser handle_msg:Msg_A..End_Msgs="This"
```

You can also combine searches for numeric values, numeric expressions, and ASCII strings:

```
M> ser -db handle_msg:Msg_A..End_Msgs=20,"message",10+10
```

Data values in a search are interpreted according to the display mode. Search for the same string, but change the display mode:

```
M> ser -dw 1000..103f=20,"MESSAGE",20
```

The search fails because the end of the expression was not on a word boundary.

Using the **-d** (display mode) option, the method of interpreting the pattern supplied by the user can be altered. If no option is given, the display mode used is taken from global default set by the **mo** command.

If addresses specified in the search reside in target system memory, the emulator is broken to the monitor and returned to the user program when the command is completed.

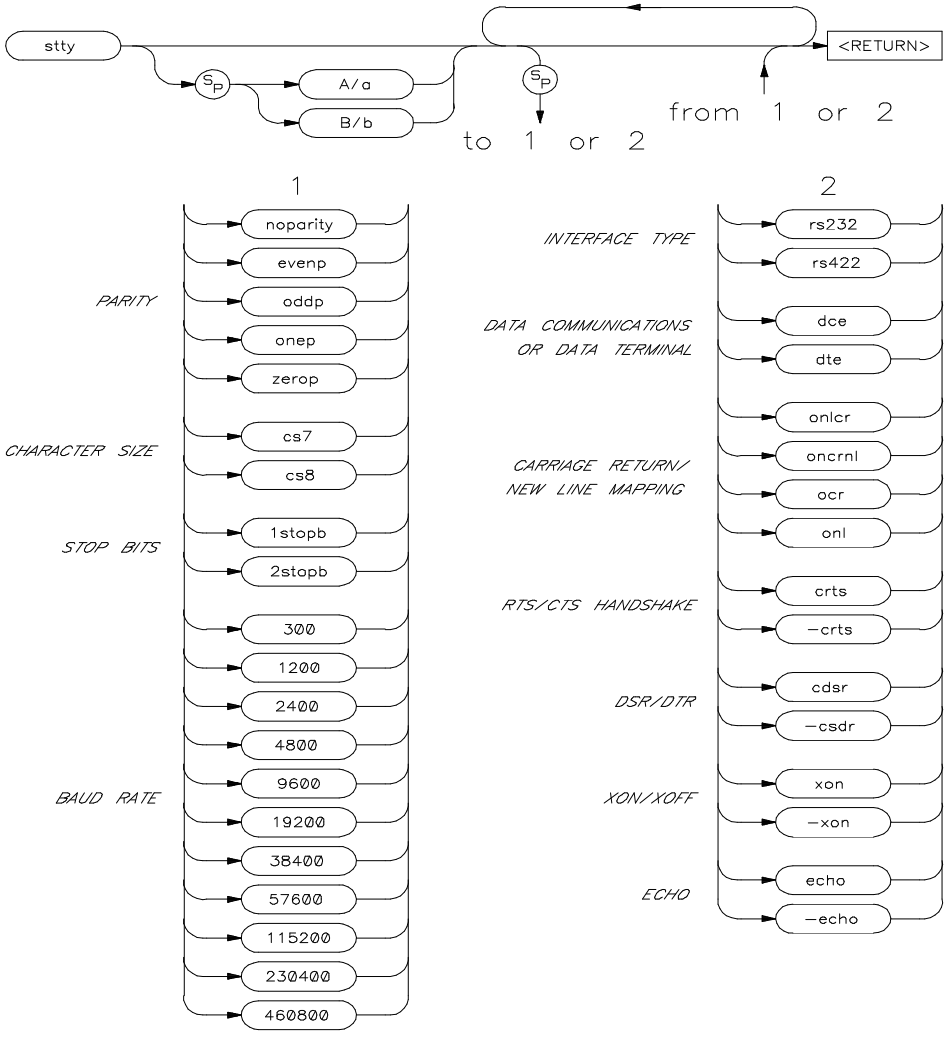
You can concatenate various combinations of <STRING> and <VALUE> to form more complex search patterns by separating the parameters with commas (**,**).

See Also

cp (used to copy the contents of one memory range to another)

m (used to display/modify memory locations)

stty


stty
SP
A/a
B/b
SP
to 1 or 2
from 1 or 2
<RETURN>
1
noparity
evenp
oddp
onep
zerop
PARITY
cs7
cs8
CHARACTER SIZE
1stopb
2stopb
STOP BITS
300
1200
2400
4800
9600
19200
38400
57600
115200
230400
460800
BAUD RATE
2
rs232
rs422
INTERFACE TYPE
dce
dte
DATA COMMUNICATIONS OR DATA TERMINAL
onlcr
oncrnl
ocr
onl
CARRIAGE RETURN/ NEW LINE MAPPING
crts
-crts
RTS/CTS HANDSHAKE
cdsr
-csdr
DSR/DTR
xon
-xon
XON/XOFF
echo
-echo
ECHO

The **stty** command allows you to modify the parameters of the data communications ports without changing the configuration switch settings.

The parameters are as follows.

PARITY	Parity for the serial port may be set odd, even, zero, one, or none.
CHARACTER SIZE	The length of each character sent by the system may be set to 7 bits or 8 bits.
STOP BITS	The number of stop bits used to terminate each character may be set to one (1) or two (2).
BAUD RATE	The baud rate (rate at which bits are transmitted and received) may be set to one of the following values: 300 1200 2400 4800 9600 19200 38400 57600 115200 230400 460800.
INTERFACE TYPE	The type of interface on the serial port may be set to either RS-232 or RS-422. RS-422 uses balanced transmission lines and therefore can achieve much higher data rates with reliability over longer distances than RS-232. Otherwise, the interfaces are similar.
DATA COM-MUNICATIONS OR DATA TERMINAL	The Serial Port may be set to operate either as Data Communications Equipment (DCE) or as Data Terminal Equipment (DTE). This configures the handshake lines and transmit/receive lines for the proper signal to pin relationships on the interface.
CARRIAGE RETURN/ LINE FEED MAPPING	You can select several different options for terminating lines of output from the system, depending on what is required by your hardware. The following choices are available: **onlcr**—generate new-line and carriage-return on output **ocrnl**—generate carriage-return and new-line on output **ocr**—generate carriage-return on output **onl**—generate new-line on output
RTS/CTS HANDSHAKE	The option **crts** enables the Request To Send/Clear To Send handshake. Specifying **-crts** disables this handshake.

DSR/DTR STATUS

The option **cdsr** enables exchange and recognition of the Data Set Ready/Data Terminal Ready status lines. Specifying **-cdsr** disables the exchange.

XON/XOFF HANDSHAKE

If you specify **xon**, the system generates XON/XOFF (DC1/DC3 characters) software handshaking to control the amount of data received at a given time. Specifying **-xon** disables this handshake sequence.

(When the emulator's receive buffer is full, it will send a DC3 (XOFF) character to the host to stop transmission; when it is ready for more data, it will send a DC1 (XON) character to restart transmission.)

ECHO

If you specify **echo**, all characters received by the emulator datacomm are echoed back to the sending system. Specifying **-echo** means the system will not echo back characters received.

You will normally use this with the echo settings required by your host computer and your terminal. Most Hewlett-Packard systems will require that you enable the echo feature, as HP host computers automatically echo characters back to data terminal devices.

The powerup default configurations for the serial port are determined by the rear panel configuration switches. See the *HP 64700 Card Cage Installation/Service Guide* for more information.

Examples

Display the current data communications setting for both ports:

```
M> stty
```

Set the baud rate for the serial port, port A, to 1200 baud:

```
M> stty A 1200
```

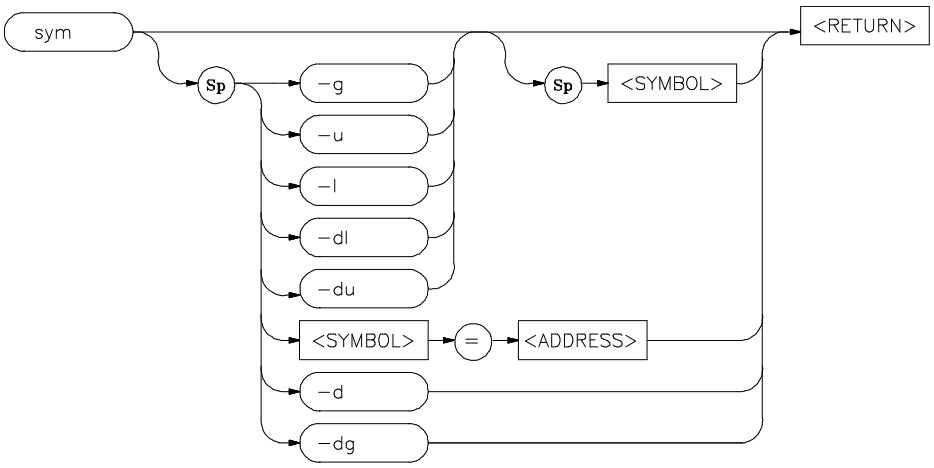
Change the baud rate back to 9600 and disable local echo on the Serial Port:

```
M> stty A 9600 -echo
```

Delete the XON/XOFF software handshake and add the RTS/CTS hardware handshake:

```
M> stty A -crts -xon
```

sym



The **sym** command defines, displays, or deletes symbols in the emulator.

The parameters are as follows.

<ADDRESS> The <ADDRESS> parameter specifies the value to assign to a user symbol.

-d The **-d** option deletes all symbols.

-du The **-du** option deletes user symbols. If a **<NAME>** parameter is not included, all user symbols are deleted. If a **<NAME>** parameter is included, only user symbols matching the entered name are deleted.

-dg The **-dg** option deletes all global symbols. No option exists to delete one global symbol.

-dl The **-dl** option deletes local symbols in a module. If a **<NAME>** parameter is not included, all local symbols are deleted for all modules. If a **<NAME>** parameter is included to specify a module name, only local symbols in the module matching the entered name are deleted.

-g The **-g** option specifies the display of global symbols. If a **<NAME>** parameter is not included, all global symbols are displayed. If a **<NAME>** parameter is included, only global symbols matching the entered name are displayed.

<NAME> This represents the symbol label to be defined or referenced. The format of the symbol name reference is determined by the type of symbol, where:

name is a user symbol or module name

:name is a global symbol name

name: is a local module name

module:name is a symbol name in a local module.

In addition, symbols can be referenced using a "wild card" expression when displaying and deleting names. Only one wildcard character can appear in a symbol name. An asterisk ("*") character is used to represent zero or more characters at the end of a symbol name. A wildcard can be used in any of the following symbol types:

name* represents a user symbol name followed by zero or more of any character or characters

:name* represents a global symbol name followed by zero or more of any character or characters

module:name* represents a local module:symbol followed by zero or more of any character or characters.

-l This option allows you to display local modules and symbols. If a **<NAME>** parameter is not included, all local modules are displayed. If a **<NAME>** parameter is included, only local symbols matching the symbol name or module are displayed.

-u This option allows you to display user symbols. If a **<NAME>** parameter is not included, all user symbols are displayed. If a **<NAME>** parameter is included, only user symbols matching the entered name are displayed.

The **sym** command without any parameters displays all of the symbols currently defined.

Examples

Display all symbols:

```
M> sym
```

Display all global symbols:

```
M> sym -g
```

Display the global symbol Int_Cmd:

```
M> sym -g :Int_Cmd
```

Define a user symbol named “mysymbol”:

```
M> sym mysymbol=107h
```

Display a user symbol named “mysymbol”:

```
M> sym -u mysymbol
```

Display all local modules:

```
M> sym -l
```

Display symbols in the local module named handle_msg:

```
M> sym -l handle_msg:
```

Display the symbol Msg_Dest in the module handle_msg, enter:

```
M> sym -l handle_msg:Msg_Dest
```

Delete all global symbols:

```
M> sym -dg
```

Display all symbols or local modules whose names begin with “hand”:

```
M> sym hand*
```

Three types of symbols are supported: global, local, and user. Global symbols reference addresses anywhere in memory using an absolute reference. Local symbols also use absolute addressing but are grouped within a “module.” User symbols are defined at the command line. Global and local symbols cannot be defined at the command line.

The definition of a module for grouping local symbols depends on the environment being used. For local symbols created by a high-level language, a module might be a function, a procedure, or a separately compilable source file. When you define local symbols through the use of a symbol file, a module, in effect, becomes a

technique to manage the symbols. It can be a mnemonic device to refer to modules, or it can be a simple way to group local symbols into a set for display and deletion purposes because the **sym** command facilitates manipulation of local symbols by their module name.

Symbols are used like equated variables. When using symbols in expressions, only the **+** and **-** operators can be used immediately before and after the symbol name. The expression can contain literals and equated (**equ**) labels, but not other symbols.

When using symbols, if a symbol and an equated value have the same name, the equated value will be used.

The symbol table can be updated in three ways:

- You can enter user symbols at the command line.
- You can update it from an external "symbol file" using the **load -So** command.
- You can load an absolute file (such as an Intel OMF file) which can contain symbols as well as program code.

A "symbol file" is a text file containing user-specified symbols. See Chapter 13 for more information.

See Also

equ (used to equate names to expressions)

load (used to load a program file with symbols, or a symbol text file)

t

The **t** command starts an emulation trace.

There are no parameters.

Examples

To begin a trace, enter:

```
M> t
```

The **t** command (or **tx** if making a synchronous CMB execution) must be entered to begin a measurement. Most other trace commands are used only for specification of triggering, sequencer, and storage parameters, or to display trace results or status.

See Also

r (starts a user program run; normally will be specified after entering the **t** command)

th (halts a trace in process)

ts (allows you to determine the current status of the emulation analyzer)

tx (specifies whether a trace is to begin upon start of CMB execution)

x (begins synchronous CMB execution)

ta

The **ta** command displays activity on each of the analyzer input lines. Each signal may be low, high, or moving. There are no parameters.

Examples

Display current analyzer signal activity:

```
M> ta
```

You will see a display similar to the following:

```
Pod 5        = 01?00100 0010?100
Pod 4        = 11?00?10 0??00100
Pod 3        = 0??????? ????????
Pod 2        = 11?00110 00000000
Pod 1        = 00000?00 1??????0
```

You can interpret the results as follows:

Bits 15, 12, 11, 6-9, 4, 2 and 1 of Pod 5 are low, bits 14, 10, 5 and 2 are low, and bits 13 and 3 are moving.

Bits 15, 14, 9 and 2 of Pod 4 are high, bits 12, 11, 7, 4, 3, 1 and 0 are low, and bits 13, 10, 6 and 5 are moving.

Bit 15 of Pod 3 is low; all other Pod 3 bits are moving.

Bits 9,10,14 and 15 of Pod 2 are high, bit 13 is moving; all others are low.

Bit 7 of Pod 1 is high; bits 1-6 and 10 are moving; all others are low.

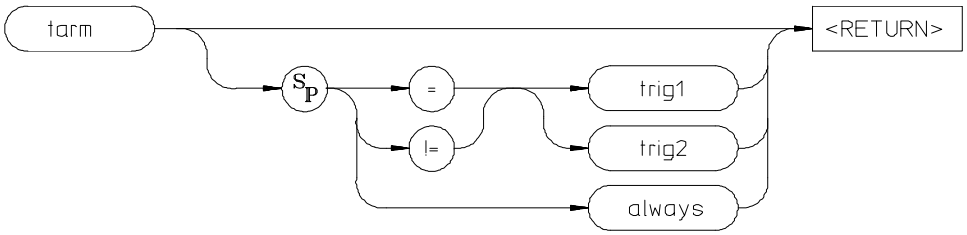
The trace activity measurement is interpreted as shown in the following table.

Type of signal activity	Symbol displayed
Signal is low	0
Signal is high	1
Signal is moving	?

Each pod (group of 16 lines) is displayed on a single line with bit 0 (LSB) at the far right and bit 15 (MSB) on the far left. Each pod represents the following analyzer bits:

Pod	Emulation Analyzer Bits
1	**Bits 0 through 15**
2	**Bits 16 through 31**
3	**Bits 32 through 47**
4	**Bits 48 through 63**
5	**Bits 64 through 79**

tarm



The **tarm** command allows you to specify an arming condition for the emulation-bus analyzer. You can specify the arm condition as the assertion of the trig1 or trig2 signals or as **tarm always**. The arm condition may then be used in specifying the analyzer trigger or in specifying branch conditions for the sequencer, as well as count or prestore qualifiers.

The parameters are as follows.

=, != The operators = and **!=** are used to respectively indicate that the arm condition is equal to, or not equal to, the specified **trig1** or **trig2** condition.

trig1 If you specify **tarm =trig1** as the arming condition, then the assertion of the trig1 signal will arm the analyzer. Conversely, if you specify **tarm !=trig1**, the analyzer will remain armed until the trig1 signal is asserted. The trig1 signal can be asserted from many sources including the analyzer itself or the rear panel BNC connector or the CMB. See **bnct**, **cmbt**, and **tgout** for examples.

trig2 If you specify **trig2** as the arming condition, then the assertion of the trig2 signal will arm the analyzer. Conversely, if you specify **tarm !=trig2**, the analyzer will remain armed until the trig2 signal is asserted. The trig2 signal can be asserted from many sources including the analyzer itself or the rear panel BNC connector or the CMB. See **bnct**, **cmbt**, and **tgout** for examples.

always If you specify **tarm always**, the analyzer is continuously armed.

If no parameters are supplied, the current **tarm** condition is displayed. The default setting after powerup or **tinit** is **tarm always**.

Examples

View the current state of **tarm**:

```
M> tarm
```

You may want to connect an external instrument, such as a logic analyzer, to the HP 64700 rear panel BNC port and have the external instrument trigger an emulation-bus analyzer trace:

```
M> bnct -r trig1
M> tcf -c
M> tarm =trig1
M> tg arm
```

This will cause the emulation-bus analyzer to trigger upon assertion of the rear panel BNC signal. To return the analyzer to the continuously armed state:

```
M> tarm always
```

Perhaps you want the analyzer to store only states received while there is NOT a trigger signal on the CMB (Coordinated Measurement Bus). To do this:

```
M> cmbt -r trig2
M> tcf -c
M> tarm !=trig2
M> tsto arm
```

The trig2 signal is set to receive the CMB trigger. Then the emulation-bus analyzer configuration is set to complex (this is required to use the **arm** parameter in analyzer expressions). Next, set the **tarm** condition to the logical NOT of the trig2 signal; finally, analyzer storage is qualified by the **arm** parameter.

See Also

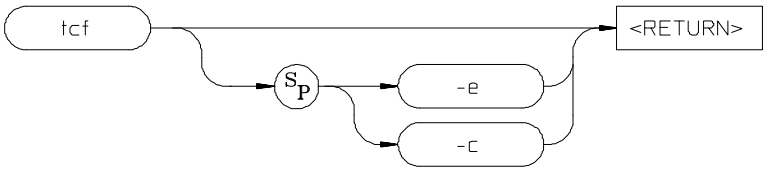
bc (can be used to cause the emulator to break to monitor execution upon receipt of the trig1 and/or trig2 signals)

bnct (used to define connections between the internal trig1 and trig2 signals and the rear panel BNC connector)

cmbt (used to define connections between the internal trig1 and trig2 signals and the CMB trigger signal)

tgout (defines whether or not the trig1 or trig2 signals are driven when the analyzer finds the trigger state)

tcf



The **tcf** command is used to set the configuration for the emulation-bus analyzer.

The parameters are as follows:

-e Specifying **-e** sets the analyzer to the easy configuration.

-c Specifying **-c** sets the analyzer to the complex configuration.

If no parameters are supplied, the current analyzer configuration is displayed. After powerup or **tinit**, the default analyzer configuration is **tcf -e**.

Examples

Display the current analyzer configuration:

```
M> tcf
```

Set the analyzer to complex configuration:

```
M> tcf -c
```

There are two possible configurations for the analyzer: easy configuration (**tcf -e**), and complex configuration (**tcf -c**). Below, each of the configurations is described briefly, along with some of the commands that modify the analyzer in each configuration. The command descriptions are not meant to be an exhaustive list of each command's features; refer to the syntax pages for a detailed description of a particular command.

Easy Configuration

When in easy configuration (**tcf -e**), much of the complexity of the analyzer is hidden. Some measurement power is lost. When you need the full power of the analyzer, switch to the complex configuration.

Expressions

In easy configuration, all analyzer commands take the general form of <**command**> <**simple_expression**>. The commands that use this form are **tcq, tif, telif, tg, tpq,** and **tsto**. A simple expression is the information that can fit into a single pattern or a single range (see **tpat**, **trng**, and **SIMPLE_EXPR** syntax for further information). Examples are **addr=2105**, **data!=15**, and **addr=4012..401a**.

Sequencing

The easy configuration allows you to have the analyzer search for a simple expression; when it is found, it can then search for a different simple expression. The ability to search for one expression, then search for another expression based on the first is known as sequencing.

In easy configuration, there are 4 sequencer terms available. Each has a primary sequence branch, which always branches to the next sequencer term (1 to 2, 2 to 3, and so on). The branch out of the last term defines the trigger term. A global restart term is also available, which will return the sequencer to term 1 if found. If both the primary branch and global restart term are satisfied simultaneously, the primary branch is always taken in preference to the restart.

Sequencer Manipulation

The simplest sequencer control is the **tg** command. This defines a one term sequence with the trigger occurring upon the branch out of the term. You can specify an occurrence count; that is, the number of times the given trigger qualifier must be found to satisfy the trigger condition.

You can exercise greater control over the easy configuration sequencer using the **tsq** command. This command allows you to insert additional sequence terms (up to the limit of four) or delete terms.

By using the **tif** command, you can define the primary branch condition for each sequence level. You can also specify an occurrence count for each branch condition. The primary branch out of the last sequence term in the list defines the trigger condition.

The **telif** command specifies the global restart condition. If both a primary branch and global restart condition are satisfied at the same time, the primary branch is always taken. However, if the primary branch has an occurrence count greater than one (1), and the global restart is encountered before the occurrence count is satisfied for the primary branch, the global restart is taken, and the primary branch occurrence count is reset to zero.

Storage Specification

You can specify which events should be stored by the analyzer using the **tsto** command. This is a global storage qualifier; that is, the qualifier is identical for all sequencer terms. Analyzer events that cause the sequencer to change states are always stored, regardless of the storage qualifier.

State/Time Counts

You can set up the analyzer to count time between states or count occurrences of a specific state using the **tcq** command.

Prestore

The analyzer has a two stage prestore pipeline. You set up the qualifier for this pipeline using the **tpq** command. When the qualifier is found, the event is stored in the pipeline; when a real storage event is found (matching the **tsto** qualifier), the pipeline is flushed and placed into trace memory immediately prior to the storage event. You can use the feature to observe the relationships between certain program variables and program routines or between two program routines. (For example, you might set a prestore state to a condition required to execute a specific routine.)

Complex Configuration

The full analyzer capability is available to you in the complex configuration (**tcf -c**). Using the multiple sequence terms, primary and secondary branch capability, and powerful expression capability, you can make just about any conceivable measurement.

Expressions

In complex configuration, all analyzer commands take the general form of **<command> <complex_expression>**. The commands that use this form are **tcq, tif, telif, tg, tpq,** and **tsto**. A complex expression is made up of pattern, range and arm labels, tied together with various operators that define the specific condition. Each of the pattern and range labels must be previously assigned to a specific simple expression using the **tpat** and **trng** commands. (These two commands are only available in the complex configuration.) So, you might define some pattern labels and a range label as follows:

```
U> tpat p1 addr=205a
U> tpat p5 data!=00
U> trng addr=4000..4011
```

And then make complex expressions as follows:

```
p1 or p5
r and p5
p1 | !r
```

See the **<COMPLEX_EXPR>** syntax pages for details on complex expressions.

Sequencing

The complex configuration allows you to have the analyzer search for a complex expression; when it is found, it can then search for a different complex expression.

In complex configuration, there are always 8 sequencer terms. Each has a primary sequence branch, which can branch to any sequencer term (1 to 5, 2 to 8, and so on). A secondary branch is also available. It can branch to any sequencer term. If both the primary branch and secondary branch are satisfied simultaneously, the primary branch is always taken in preference to the secondary branch.

Sequencer Manipulation

The simplest sequencer control is the **tg** command. As in easy configuration, it defines a two term sequence with the trigger in the second term. You can specify an occurrence count; that is, the number of times the given trigger qualifier must be found to satisfy the trigger condition.

You can exercise greater control over the complex configuration sequencer using the **tsq** command. Although you cannot add or delete sequence terms in complex configuration (there are always eight), you can specify the trigger term. You can also reset the sequencer (which clears all the branch specifiers and storage qualifiers).

By using the **tif** command, you can define the primary branch condition for each sequence level. You can also specify an occurrence count for each branch condition, and the destination term for each branch.

The **telif** command specifies the secondary branch condition, which can jump to any sequence term. If both a primary and secondary branch condition are satisfied at the same time, the primary branch is always taken. However, if the primary branch has an occurrence count greater than one (1), and the secondary branch is encountered before the occurrence count is satisfied for the primary branch, the secondary branch is taken, and the primary branch occurrence count is reset to zero.

Storage Specification

You can specify events to be stored by the analyzer using the **tsto** command. You may specify different storage qualifiers for each sequence term; if you have sequence term 5 active during execution of a particular procedure and you want to store all of the writes while that procedure is executing, you can use **tsto** to qualify writes while term 5 is active (**tpat p2 stat=write;tsto 5 p2**).
If you don't include a term number when specifying the storage qualifier, your storage qualifier will apply to all sequence terms.

State/Time Counts, Prestore

The state/time counting and prestore facilities are identical to those provided in the easy configuration; however, you must specify a complex expression instead of an easy expression in qualifying the state count or prestore.

Resetting the Analyzer Configuration

When the analyzer configuration is changed, the entire analyzer specification is reset. You can perform a reset back to the default sequencer setup in either configuration by using the **tsq -r** command.

When the trace configuration is changed, the count qualifier (**tcq**) is reset to "none" (instead of "time") if the clock mode (**tck**) is fast (F) or very fast (VF).

See Also

tarm (used to set the analyzer arm specification; this specification can only be used in analyzer expressions in complex configuration)

tcq (sets the expression for the trace count qualifier in either analyzer configuration)

telif (sets the global restart in easy configuration, secondary branch condition in complex configuration)

tg (used to set a trigger expression in either analyzer configuration)

tif (sets primary branch specification in either analyzer configuration)

tpat (used to label complex analyzer expressions with a pattern name; the pattern name is then used by the analyzer setup commands. Only valid in complex configuration)

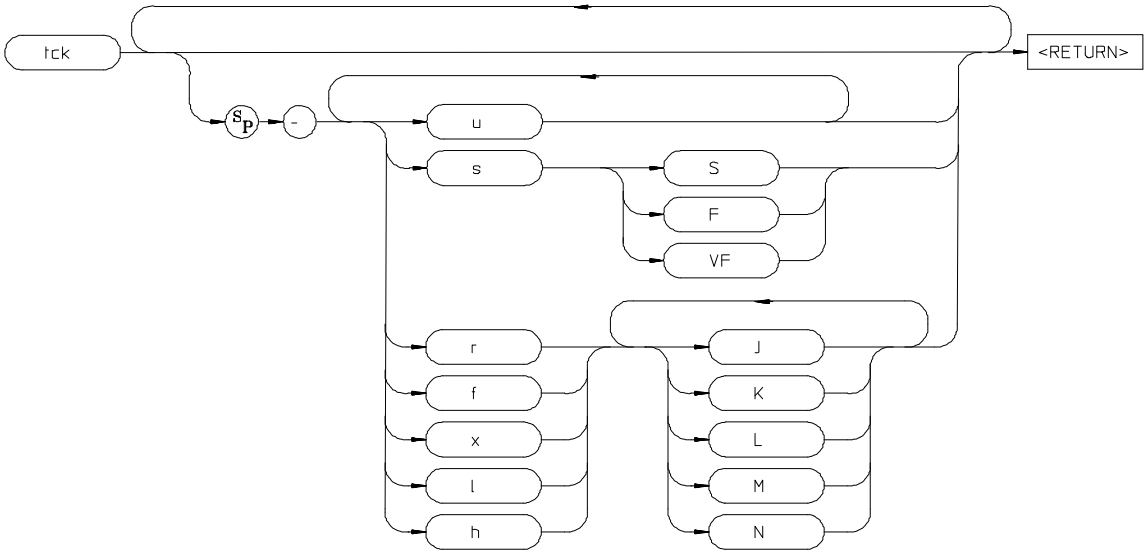
tpq (specifies trace prestore qualifier in either analyzer configuration)

trng (defines a range of values to be used in complex analyzer expressions)

tsto (specifies a qualifier to be used when storing analyzer states)

tsq (used to modify the trace sequencer's number of terms and trigger term)

tck



The **tck** command allows specification of clock qualifiers, master edges and maximum clock speed of the master clocks used for the emulation-bus analyzer.

The parameters are as follows:

b — If the **b** option is specified, only background monitor code will be qualified into the analyzer.

u — If the **u** option is specified, only user code will be qualified into the analyzer. This is the default.

The **u** and **b** qualifiers are ORed with all of the other qualifiers specified.

s — The **s** option indicates that the maximum clock speed is to be modified per a one or two letter code immediately following.

S — Specifies a clock speed of SLOW; less than or equal to 16 MHz.

F — Specifies a clock speed of FAST; between 16 MHz and 20 MHz.

VF — Specifies a clock speed of VERY FAST; between 20 MHz and 25 MHz.

r — Specifying **r** indicates that the analyzer is to be clocked on the rising edge of the indicated clock signal.

f — Specifying **f** indicates that the analyzer is to be clocked on the falling edge of the indicated clock signal.

x — Specifying **x** indicates that the analyzer should be clocked on both the rising and falling edges of the indicated clock signal.

l — Specifying **l** indicates that the analyzer should only be clocked by other clock signals when this clock signal is low (less positive/more negative voltage). Used as a qualifier (example: clock on rising edge of J only if K is low).

h — Specifying **h** indicates that the analyzer should only be clocked by other clock signals when this clock signal is high (more positive/less negative voltage). Used as a qualifier (example: clock on both edges of K only if J is high).

CLOCK SIGNALS — The **r, f, x, l,** and **h** operators may be used on the following clock signals: **J, K, L, M** or **N**.

If no parameters are specified, the current clock definitions are displayed. After powerup or **tinit**, the **u** option is always set. Other clock options set at initialization depend on the particular emulator in use and whether or not there is an external analyzer present.

Examples

Display the current settings of the master clocks after powerup or a **tinit**:

```
M> tck
```

Suppose that the target system clock rate for the MC68030/EC030 processor is 40 MHz, and **cf emwait=en**. The resulting data rate is 20 MHz, so you must set the clock rate to fast, and disable the state/time counter. (See Chapter 5 for more information.)

```
R> tcq none
R> tck -s F
```

Add tracing of background code to the current clock settings:

```
M> tck -ub
```

The **tck** command is included with the system for internal system initialization and system control through high-level software interfaces. You may also use this command to set the analyzer data rates, which depend on the target system clock rate and number of rate states. See the section on analyzer clocks in Chapter 5, "Using the Analyzer," for more information.

Changing the clock speed with the **s <SPEED>** option affects the **tcq** command parameters. When speed is set to **sS** (slow), the **tcq** command may either count states or time. When speed is set to **sF** (fast), the **tcq** command may be used to count states but not time. If clock speed is set to **sVF** (very fast), **tcq** cannot count either states or time and should be set to **tcq none.**

The clocking options operate on five different clock signals: **J, K, L, M** and **N**. Clocks **L, M,** and **N** are generated by the emulator; the emulation master clock edges are set at powerup for the particular emulator being used and should not be changed by you.

When several clock edges are specified, any one of the edges can clock the trace. If several qualifiers (**l** or **h**) are specified, they are ORed so that the trace is clocked when any of the qualifiers are met.

See Also

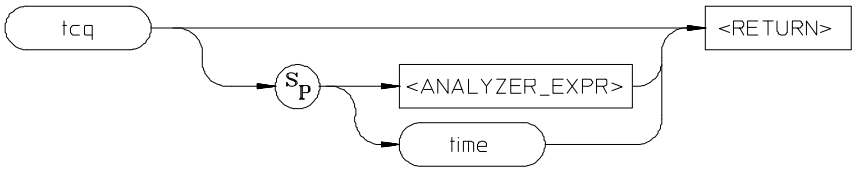
ta (display current trace signal activity. This can be useful after you have modified the clocks for the external analyzer; you can issue a **ta** command and verify that you are seeing activity on the signals of interest.)

tcq (used to specify trace count qualifier for states, time, or none; maximum clock speed set in **tck** affect which **tcq** parameters are valid)

tsck (used to define slave clock signals used by the analyzer; **tck** defines the master clock signals. Default mode for **tsck** is off on all pods.)

Also see Chapter 5, "Using the Analyzer," for information on analyzer clock speed.

tcq



The **tcq** command allows you to specify a qualifier for the emulation trace tag counter.

The parameters are as follows:

<ANALYZER_EXPR>

<**ANALYZER_EXPR**> allows you to specify an expression to be counted by the trace tag counter. This expression consists of a <**SIMPLE_EXPR**> in analyzer easy configuration and a <**COMPLEX_EXPR**> in complex configuration. See the syntax pages for expressions for specific details of analyzer expressions. In either configuration, the expression may consist of the states **any** (count all states) or **none** (disable trace tag counting).

The count qualifier **tcq arm** is not permitted in any configuration.

time

If you specify **time** rather than an analyzer expression, the trace tag counter measures the amount of time between stored states.

The **tcq time** qualifier is only available when the analyzer clock speed is set to the slow (**S**) speed setting (default). If the clock speed is set to very fast (**VF**), then trace tag counting must be turned off by specifying **tcq none**. Refer to the **tck** command (analyzer clock specification) for further information.

If no parameters are given, the current count qualifier is displayed. at powerup or after **tinit** initialization, the clock qualifier defaults to the state **tcq time**.

Examples

To view the current **tcq** setting, type:

```
R> tcq
```

To see the effects of counting no states, do the following:

```
R> init -c
R> demo
```

```
R> tg addr=demo:Loop
R> tcq none
R> t
U> r
```

When you display the trace list, note the asterisks in the **count** field.

To count time intervals, do the following:

```
U> tcq time
```

To change the trace listing so that the time intervals are displayed as an absolute value relative to the trigger state instead of the last state stored, type:

```
U> tf addr,h mne count,a seq
```

When you display the trace list, the time interval is now measured relative to the trigger state. To reset the trace format to count relative, type:

```
U> tf addr,h mne count,r seq
```

When the tag counter is active, the analyzer counts occurrences of the expression you specify (which may include **time**, **none**, or simple or complex expressions, depending on analyzer configuration. Each time a trace state is stored, the value of the counter is also stored and the counter is reset. The tag counter shares trace memory with stored states, so only half as many states can be captured by the analyzer when the tag counter is active. (The analyzer can store 1024 states with **tcq none**, and 512 states otherwise.)

See Also

tck (used to specify the clock source and clock parameters for the analyzer)

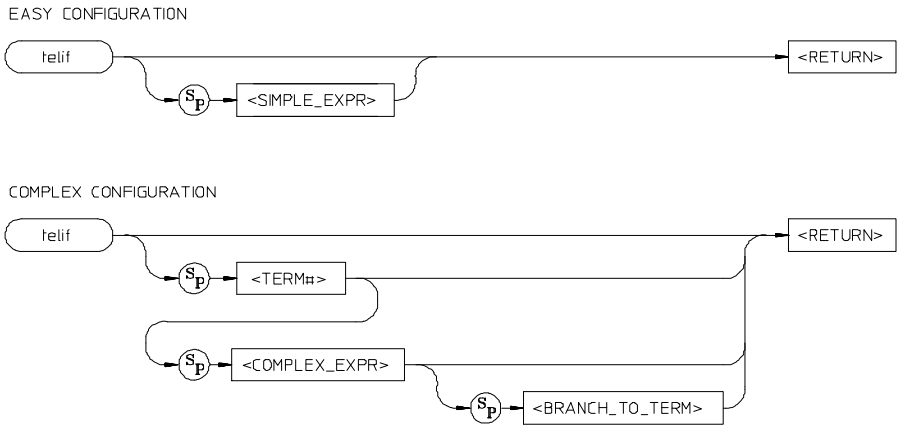
tp (specifies position of the trigger within the trace; note that **tcq** affects the number of states the analyzer can store and therefore may affect trigger positioning)

tpat (assigns analyzer expressions to pattern names in complex configuration; the pattern names are then used to specify qualifiers in other analyzer commands such as **tcq**)

trng (specifies a range of values to be used as a complex mode qualifier; this range definition can be used as a count qualifier by **tcq**)

tsq (used to manipulate the trace sequencer)

telif



The **telif** command allows you to set the global restart qualifier (in easy configuration) for the emulation-bus analyzer sequencer. In complex configuration, **telif** lets you set the secondary branch qualifier for each term of the emulation-bus analyzer sequencer.

The parameters for easy configuration are as follows:

<SIMPLE_EXPR> **<SIMPLE_EXPR>** lets you directly specify an analyzer expression to use as a global restart qualifier. For example, **<SIMPLE_EXPR>** might consist of the expression **addr=2000**. For detailed information on specification of simple expressions, see Chapter 11, “Expressions.”

The parameters for complex configuration are as follows:

<TERM#> **<TERM#>** lets you specify a sequencer term number to associate with the given **<COMPLEX_EXPR>**. When you associate a term number with a complex expression, that expression is only used as a secondary branch qualifier at the sequencer level specified by the term number. If you specify **<TERM#>** without an expression, the secondary branch qualifier currently associated with that term number is displayed.

<COMPLEX_ EXPR> **<COMPLEX_EXPR>** allows you to specify complicated analyzer expressions made up of relationships between simple analyzer expressions. When you create a complex expression, you must first assign pattern names (**p1-p8**) to simple expressions using the **tpat** command. You then use the pattern names and relational operators to create complex expressions. For example, if you wish to branch from

term 1 to term 2 when **address=2000** and **data=20** or when **address=2000** and **data=42**, you would use the following commands:

```
U> tpat p1 addr=2000 and data=20
U> tpat p2 addr=2000 and data=42
U> telif 1 p1 | p2 2
```

The | symbol represents an intra-set OR operator. For more information on complex expressions, operators, and pattern sets, see Chapter 11, "Expressions."

<BRANCH_ TO_TERM>

The <**BRANCH_TO_TERM**> parameter allows you to indicate the branch destination when the <**COMPLEX_EXPR**> is found. For example, you may wish to have the sequencer branch from term 1 to term 3 after the expression is found. This would be specified as **telif 1 <COMPLEX_EXPR> 3**. If you do not specify a term number, the default is to increment the sequencer level (**telif <TERM#> <COMPLEX_EXPR> (<TERM#> + 1)**).

If **telif** is entered with no parameters, the global restart qualifier or secondary branch qualifiers (depending on analyzer configuration) for all sequencer levels are displayed. If **telif** is entered with only a <**TERM#**> parameter in complex configuration, the secondary branch qualifier for only that term number is displayed.

Upon initialization via a powerup sequence or the **tinit** command, the secondary branch specifiers are set to **telif never**.

In complex configuration, if <**BRANCH_TO_TERM**> is not specified, the default is **(<TERM#> + 1)**.

At sequencer term number 8, the default branch to condition is <**TERM#**>; that is, branch to the same term.

Examples

To have the analyzer record the routine only when an "A" or "B" is input, do the following:

```
U> tsq -i 2
U> tif 1 addr=Int_Cmd
U> tif 2 addr=handle_msg:Print_Msg
U> telif addr=handle_msg:Cmd_I
```

In complex configuration, telif commands can branch to other terms. For example:

```
R> tcf -c
R> tpat p1 addr=501a
R> tpat p2 addr=703c
R> telif 4 p2 3
R> telif 1 p1 6
```

The **telif** command is used as a global restart qualifier in easy configuration and a secondary branch qualifier in complex configuration. The hierarchy of the **tif** and **telif** commands is such that either branch will be taken if found before the other; however, if both branches are found simultaneously, the **tif** branch is always taken over the **telif** branch.

When in easy configuration, the sequencer will restart by jumping to sequencer term number one (1) when the expression specified by **telif** occurs. The **telif** command allows you to specify a global restart qualifier. This means that the analyzer will restart the sequencer when the qualifier is satisfied.

When in complex configuration, the sequencer will branch to the sequencer level specified by the **<BRANCH_TO_TERM>** parameter when the expression specified is found. There are always eight sequencer terms available. Position of the trigger term is defined with the **tsq** command. If both the **tif** and **telif** expressions are satisfied simultaneously, the **tif** branch is taken. Otherwise, branching occurs according to which expression is first satisfied. The **telif** command allows you to branch to any sequence term from any other term.

If the **tif** expression for the given **<TERM#>** has a **<COUNT>** parameter other than one (1), the counter is reset to zero (0) if the **telif** branch is taken before the occurrence counter parameter is satisfied. For example, if the **tif** counter parameter is 7, and the **tif** expression has been found five times, then the **telif** expression is satisfied, the **telif** branch will be taken and the **tif** counter will be reset from 5 to 0. This might cause you difficulty if you happen to have **telif** branching back to the same term; your occurrence condition may or may not be satisfied.

See Also

tarm (allows you to specify that the **trig1** or **trig2** signal will arm the analyzer. This arm condition can then be used as part of the secondary branch qualifier)

tcf (used to select whether the analyzer is operated in easy configuration or complex configuration)

tif (used to specify a primary branch specification for the analyzer)

tg (used to set up a simple trigger qualifier in either analyzer configuration. Specifying the **tg** command overrides the current sequencer specification and will modify the existing **telif** qualifier stored in sequence term number 1)

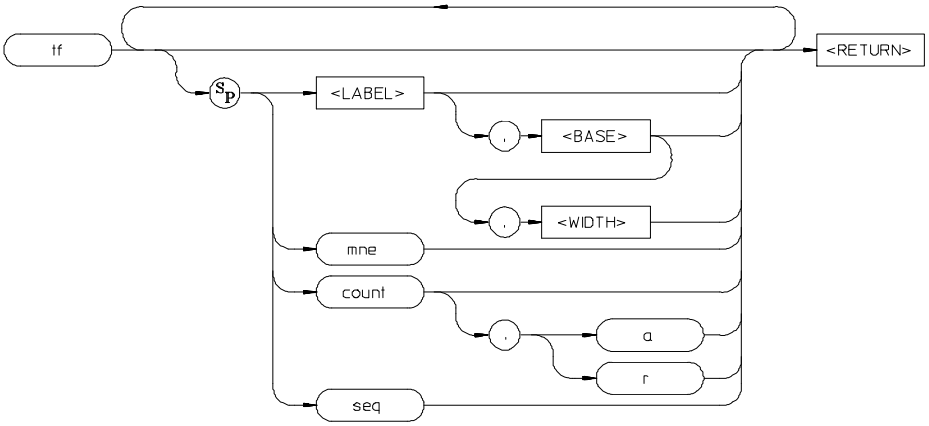
tpat (used to assign pattern names to simple expressions for use in specifying complex expressions. These complex expressions are used to specify **telif** qualifiers in analyzer complex configuration)

trng (used to set up an expression which assigns a range of values to a range variable. This range information may be used in specifying complex **telif** qualifiers)

tsto (specifies a global trace storage qualifier in both easy & complex configurations; also specifies a trace storage qualifier for each sequencer term in complex configuration. Used to control the types of information stored by the analyzer)

tsq (used to manipulate the trace sequencer)

tf



The **tf** command allows you to specify which pieces of information from the emulation-bus analyzer trace will be displayed by **tl** (trace list) commands.

The parameters are as follows:

<LABEL> If you specify **<LABEL>**, the analyzer bits associated with that label will be displayed in a column of the trace list with **<LABEL>** as the column header.

<BASE> **<BASE>** allows you to specify the numeric base in which <LABEL> is to display. The choices are **Y** (binary), **Q** or **O** (octal), **T** (decimal), **H** (hexadecimal), or **A** (ASCII). The specifiers are not case sensitive. In ASCII mode, non-printing characters are displayed as periods (.). If **<BASE>** is not specified, the default base is hexadecimal.

<WIDTH> This option allows you to set the width of only the address field to values from 4 to 50. If your emulator supports symbols, by setting **<WIDTH>**, you can view symbols in the address field when you display memory mnemonic.

<LABEL>, **<BASE>**, and **<WIDTH>** must each be separated by a comma (,).

mne If you specify **mne**, the disassembled mnemonic for each instruction captured by the analyzer is displayed. To ensure correct operation of **mne**, the labels **addr, data, stat** and **extra** (if applicable) must be defined according to their power up defaults for the target processor being emulated; otherwise, incorrect disassembly may occur.

count — If you specify **count**, the state or tag time counter defined by **tcq** is displayed in the trace list. If you have designated prestore states via the **tpq** command, these prestore states will be flagged in the **count** column of the trace list.

a — Specifying **count,a** causes the state/time counter to display the count in absolute mode. That is, each counter value is shown relative to the trigger state. Therefore, states before the trigger will show as negative values and states after the trigger will show as positive values. Prestore states do not have counts.

r — Specifying **count,r** causes the state/time counter to display the count in relative mode. That is, each counter value is shown relative to the previous state. As with **count,a**, prestore states do not have counts.

seq — If you specify **seq**, an indicator is printed for each state which caused the sequencer to branch from one term to another (whether the same term or a different term).

If no parameters are given, the current settings of the trace format are displayed. Upon powerup or after a **tinit** command, the trace format is **tf addr,H mne count,R seq**.

Examples

To view the default trace format, type:

```
M> tf
```

Set the format so the address and data values are displayed in hexadecimal:

```
U> tf addr,H data,H
```

Set the format so the address is displayed in decimal and the data in binary:

```
U> tf addr,T data,Y
```

Display processor status information in binary:

```
U> tf addr,H mne stat,Y
```

To see what types of ASCII information are transferred on the least-significant byte of the data bus, type:

```
U> tlb byte3 56..63
U> tf addr,H mne byte3,A
```

To display trace sequencer information along with a status display in hex, type:

```
U> tf addr,H mne stat,H seq
```

To display state/time counter information, type:

```
U> tf addr,H mne count seq
```

To change the counter display to count relative, type:

```
U> tf addr,H mne count,R seq
```

Various **tf** format items may be concatenated as desired on the command line by including a space between each format item.

Each format item specifies a column of the trace list display.

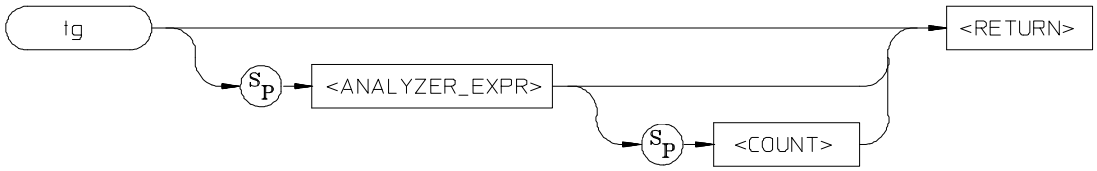
Changing the trace format *does not* change the type of information captured by the analyzer; it only specifies how the captured data should be displayed.

See Also

tl (displays the current data in the trace memory of the emulation-bus analyzer according to the specifications set up by **tf**)

tlb (define labels which represent groups of emulation-bus analyzer input lines; these labels may be used to create special trace list displays by including the labels in the **tf** definition)

tg



The **tg** command sets a trigger condition for the emulation-bus analyzer.

The parameters are as follows:

<ANALYZER_EXPR>

<ANALYZER_EXPR> allows you to specify the expression to recognize as a trigger. This expression consists of a **<SIMPLE_EXPR>** in analyzer easy configuration and a **<COMPLEX_EXPR>** when the analyzer is in complex configuration. See Chapter 11, "Expressions," for specific details of analyzer expressions. In either configuration, the expression may consist of the states **any** or **all** (trigger on any state) or **none** or **never** (don't trigger the analyzer).

<COUNT>

You use the **<COUNT>** parameter to specify the number of times the expression **<ANALYZER_EXPR>** must occur before the trigger condition is satisfied. **<COUNT>** is specified as a decimal integer value; if **<COUNT>** is not specified, the default is one (1).

If no parameters are specified, the current primary branch condition for sequencer term 1 is displayed. Note that this is not necessarily the trigger condition, depending on the analyzer commands leading up to this point. After powerup or **tinit** initialization, **tg** is set to **tg any**.

Examples

To trigger the analyzer when the Loop symbol has occurred 32 times, type:

```
U> tg addr=demo:Loop 32
```

When you start the trace and run the program, the trigger condition will be found; line 0 of the trace display will show the 32nd iteration of the trigger value. To trigger on any non-zero command in the demo program do the following:

```
U> tcf -c
U> tlb byte3 56..63
U> tpat p5 byte3!=00
```

```
U> tpat p1 addr=demo:Cmd_Input and stat=read
U> tg p1 and p5
```

When you start the trace, run the program, and input a non-zero command, the trigger condition will be found. Line 0 of the trace listing will show that the analyzer triggered on an address with data not equal to zero.

The **tg** command modifies the current analyzer sequence specification. The manner in which the sequencer is modified is dependent upon the analyzer configuration.

If the analyzer is in easy configuration (**tcf -e**), the sequencer is reduced to a one term sequence triggering upon exit from term 1. The global restart qualifier is set to never (**telif never**); the primary branch condition is set to the specified trigger expression (**tif 1 <EXPR> <COUNT>**).

When operating the analyzer in easy configuration, using the **tg** command resets the sequencer to a two term sequence with a primary branch in term number one corresponding to the trigger condition.

If the analyzer is in complex configuration (**tcf -c**), the sequencer is modified to trigger upon entrance to the second sequence term (**tsq -t 2**), the secondary branch qualifier is set to never (**telif 2 never**), and the primary branch qualifier for term number 1 is set to the specified expression (**tif 1 <EXPR> 2 <COUNT>**).

In analyzer complex configuration, the **tg** command defines simple sequence specification and overwrites sequencer terms 1 and 2 to create the new specification.

When the expression specified occurs the number of times specified in the **<COUNT>** parameter, the analyzer has found its trigger.

The analyzer storage qualifier (**tsto**) is not affected in either configuration; therefore, the analyzer uses the storage qualifier from the most recent **tsto** command.

See Also

bc (allows you to break the emulator to the monitor when various conditions occur; you can have the emulator break upon analyzer trigger by specifying **tgout trig1** and **bc -e trig1** (or you could use the trig2 signal to perform the same function))

t (starts an emulation trace)

tarm (used to specify an analyzer arm condition; the analyzer will not trigger until the arm condition is received if you specify **tg arm**)

tcf (used to specify whether the analyzer is operated in easy or complex configuration)

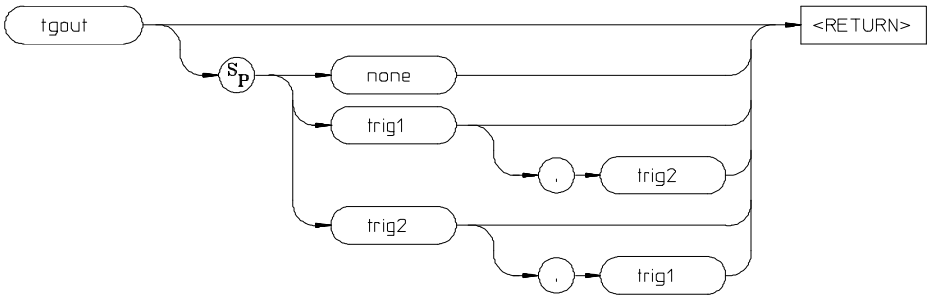
tpat (used to assign pattern names to simple analyzer expressions; the pattern names are then used in creating complex analyzer expressions which could be used with the **tg** command to trigger the analyzer)

trng (used to specify a range of values for a particular group of analyzer lines; this range may be used in specifying complex analyzer expressions for triggering the analyzer)

tsto (specifies which states encountered by the analyzer should be stored in trace memory)

tsq (used to manipulate the trace sequencer. Note that the sequencer's current status is affected by the **tg** command.)

tgout



The **tgout** command allows you to specify which of the internal trig1 and/or trig2 signals will be driven when the emulation-bus analyzer finds its trigger condition.

The parameters are as follows:

none — If **none** is specified, neither the trig1 nor trig2 signals are driven when the analyzer finds its trigger state.

trig1 — If **trig1** is specified, then the trig1 signal is driven by the analyzer when the trigger state is found.

trig2 — If **trig2** is specified, then the trig2 signal is driven by the analyzer when the trigger state is found.

To specify that both trig1 and trig2 should be driven when the analyzer trigger is found, concatenate both options with a comma: **tgout trig1,trig2**.

If no parameters are specified, the current state of **tgout** is displayed. Upon powerup or **tinit**, the default state is **tgout none**.

Examples

Display the state of **tgout:**

```
M> tgout
```

Set the emulator so that it will break to monitor execution upon receipt of the analyzer trigger:

```
M> tcf -e
M> bc -e trig1
```

```
M> tgout trig1
M> tg addr=demo:Loop
```

The emulator will break to the emulation monitor when it encounters the command input loop in the demo program.

Note that if the analyzer is receiving trig1 or trig 2 via the **tarm** command, then that signal cannot be driven, although no error message will be displayed.

See Also

bc (allows you to specify a break to emulation monitor when the **tgout** condition is satisfied)

bnct (specifies whether or not trig1 and trig2 are used to drive and/or receive the rear panel BNC connector signal line)

cmbt (specifies whether or not trig1 and trig2 are used to drive and/or receive the CMB trigger signal)

tarm (used to specify that the analyzer will be armed upon assertion or negation of trig1 or trig2)

th

The **th** command stops an emulation trace. This command has no parameters.

Examples

Start an emulation trace:

```
M> t
```

Stop the trace:

```
M> th
```

The analyzer will stop driving the **trig1** and **trig2** signals when the trace is halted. This may cause you difficulty in making measurements with instruments connected to the BNC. For example, if you set the HP 64700 analyzer to drive **trig1** (**tgout trig1**) when the trigger condition is found, then pipe this to the BNC connector with **bnct -d trig1**, the BNC signal will be driven high when the HP 64700 analyzer finds its trigger while a trace is in progress; it will fall low when the trace finishes.

You should start the HP 64700 trace after you have begun the external instrument's measurement. Otherwise, the following measurement errors may occur, depending on the type of external instrument you are using:

- With an edge sensitive instrument, starting the instrument after the HP 64700 finds the analyzer trigger will mean that the instrument never sees the transition of the **trig1** line and therefore never triggers.
- With a level sensitive instrument, starting the instrument after the HP 64700 finds the trigger will mean that the instrument triggers immediately; although many states of interest have probably already passed.

If the analyzer trigger specification has not been found, you will need to use the **th** command to halt the analyzer before you can display the trace list.

See Also

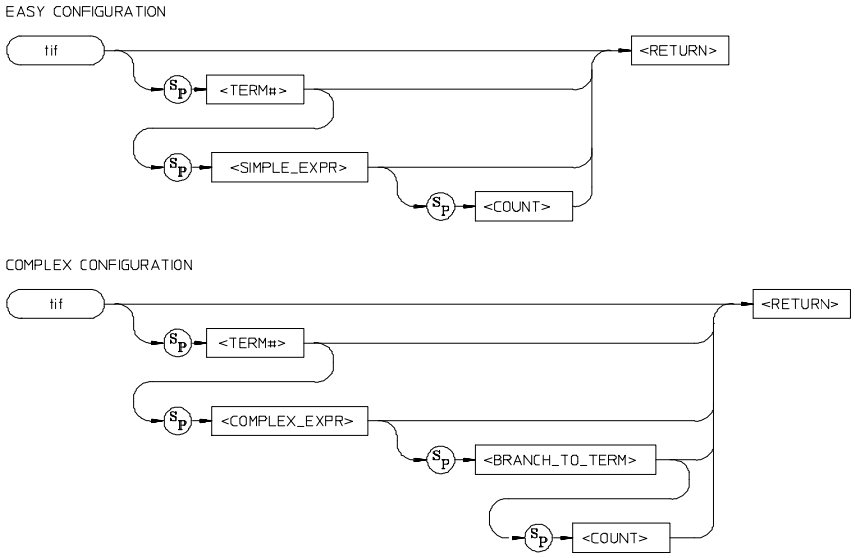
t (used to start an analyzer trace)

ts (allows you to determine the current status of the emulation-bus analyzer)

tx (starts an analyzer trace upon receipt of the CMB execute signal)

x (starts a synchronous CMB execution)

tif



The **tif** command allows you to set the primary branch qualifier for each term of the emulation-bus analyzer sequencer.

Easy configuration parameters:

<TERM#> When you specify **<TERM#>**, it indicates which sequencer term's primary branch qualifier is to be modified with the qualifier specified in the **<SIMPLE_EXPR>** parameter. If you specify **<TERM#>** without an expression, the **tif** qualifier for that term number is displayed.

<SIMPLE_EXPR> **<SIMPLE_EXPR>** lets you directly specify an analyzer expression to use as a storage qualifier. For example, **<SIMPLE_EXPR>** might consist of the expression **addr=2000**. For detailed information on specification of simple expressions, see Chapter 11, "Expressions."

<COUNT> You use the **<COUNT>** parameter to specify the number of times the expression **<SIMPLE_EXPR>** must occur before the primary branch condition is satisfied. **<COUNT>** is specified as a decimal integer value; if **<COUNT>** is not specified, the default is one (1).

Complex configuration parameters:

<TERM#>

<**TERM#**> lets you specify a sequencer term number to associate with the given <**COMPLEX_EXPR**>. When you associate a term number with a complex expression, that expression is used as a branch qualifier at the sequencer level specified by the term number. If you specify <**TERM#**> without an expression, the complex expression currently associated with that term number is displayed.

<COMPLEX_ EXPR>

<**COMPLEX_EXPR**> allows you to specify complicated analyzer expressions made up of relationships between simple analyzer expressions. When you create a complex expression, you must first assign pattern names (**p1-p8**) to simple expressions using the **tpat** command. You then use the pattern names and relational operators to create complex expressions. For example, if you wish to branch from term 1 to term 2 when **address=2000** and **data=20** or when **address=2000** and **data=42**, you would use the following commands:

```
U> tpat p1 addr=2000 and data=20
U> tpat p2 addr=2000 and data=42
U> tif 1 p1 | p2 2
```

The | symbol represents an intra-set OR operator. For more information on complex expressions, operators, and pattern sets, see Chapter 11, “Expressions.”

<BRANCH_ TO_TERM>

The <**BRANCH_TO_TERM**> parameter allows you to indicate the branch destination when the <**COMPLEX_EXPR**> is found. For example, you may wish to have the sequencer branch from term 1 to term 3 after the expression is found. This would be specified as **tif 1 <COMPLEX_EXPR> 3**. If you do not specify a term number, the default is to increment the sequencer level (**tif <TERM#> <COMPLEX_EXPR> (<TERM#> + 1)**).

<COUNT>

You use the <**COUNT**> parameter to specify the number of times the expression <**COMPLEX_EXPR**> must occur before the primary branch condition is satisfied. <**COUNT**> is specified as a decimal integer value; if <**COUNT**> is not specified, the default is one (1).

If you specify the <**COUNT**> parameter, you must also specify a <**BRANCH_TO_TERM**> parameter. If you omit the <**BRANCH_TO_TERM**> parameter when specifying <**COUNT**>, the system will interpret the count as “branch to term” information; if greater than eight (8), an error will be returned; otherwise, you will have just specified an incorrect branch.

If **tif** is entered with no parameters, the primary branch qualifiers for all sequencer levels are displayed. If **tif** is entered with only a <**TERM#**> parameter, the primary branch qualifier for only that term number is displayed.

Upon initialization via a powerup sequence or the **tinit** command, the primary branch specifiers are set to **tif <TERM#> any (<TERM#> + 1)**.

In complex configuration, if **<BRANCH_TO_TERM>** is not specified, the default is **(<TERM#> + 1)**; if **<COUNT>** is not specified, the default count is one (1).

At sequencer term number 8, the default branch to condition is **<TERM#>**; that is, branch to the same term.

Examples

Suppose that you want to trigger on the ninth character written to the output area after 4 "A" commands have been entered.

```
R> init -c
R> demo
R> equ cmd_count=4
R> equ char_written=8
R> tif 1 addr=handle_msg:Cmd_A cmd_count
R> tif 2 addr=handle_msg:Again char_written
R> t
R> r
```

Now, enter 4 "A" commands. The trace list will show the trigger at the ninth execution of the Again loop.

The **telif** command is used as a global restart qualifier in easy configuration and a secondary branch qualifier in complex configuration. The hierarchy of the **tif** and **telif** commands is such that either branch will be taken if found before the other; however, if both branches are found simultaneously, the **tif** branch is always taken instead of the **telif** branch.

When in easy configuration, the sequencer will increment to the next sequencer level when the expression specified by **tif** occurs the number of times specified by the **<COUNT>** parameter. There is a maximum of four sequence levels; only one is available at initialization. If you require more sequencer levels, you must insert them with the **tsq** command. (The term for which you are specifying a primary branch for with the **tif** command must be present in the sequence.) The branch out of the last sequencer term constitutes the trigger.

When in complex configuration, the sequencer will branch to the sequencer level specified by the **<BRANCH_TO_TERM>** parameter when the expression specified occurs the number of times indicated in the **<COUNT>** parameter. There

are always eight sequencer terms available. Position of the trigger term is defined with the **tsq** command.

See Also

tarm (allows you to specify that the **trig1** or **trig2** signal will arm the analyzer. This arm condition can then be used as part of the primary branch qualifier)

tcf (used to select whether the analyzer is operated in easy configuration or complex configuration)

telif (used to specify a secondary branch specification for the analyzer)

tg (used to set up a simple trigger qualifier in either analyzer mode. Specifying the **tg** command overrides the current sequencer specification and will modify the existing **tif** qualifier stored in sequence term number 1)

tpat (used to assign pattern names to simple expressions for use in specifying complex expressions. These complex expressions are used to specify **tif** qualifiers in analyzer complex configuration)

trng (used to set up an expression which assigns a range of values to a range variable. This range information may be used in specifying complex **tif** qualifiers)

tsto (specifies a global trace storage qualifier in both easy and complex configurations; also specifies a trace storage qualifier for each sequencer term in complex configuration. Used to control the types of information stored by the analyzer)

tsq (used to manipulate the trace sequencer)

tinit

The **tinit** command restores all trace specification items to their powerup default values. See “Defaults.”

The **tinit** command has no parameters.

Examples

To reset the analyzer parameters to the powerup defaults, type:

```
M> tinit
```

These are the powerup defaults for the trace specification:

Analyzer arm

```
tarm always
```

Trace Configuration

```
tcf -e
```

Note that if the trace configuration was complex, it is reset to easy configuration.

Analyzer master clocks

```
tck -r L -u -s S
```

Trace count qualifier

```
tcq time
```

Trace format

```
tf addr,H mne count,R seq
```

Trace trigger

```
tg any
tgout none
```

Analyzer signal line labels

```
#### Emulation trace labels
tlb addr 0..31
tlb data 32..63
tlb stat 64..79
```

Trigger Position

```
tp s
```

Trace Prestore Qualifier

```
tpq none
```

Trace sequencer (includes branch and store conditions)

```
tif 1 any
tsto all
telif never
```

Trace slave clocks

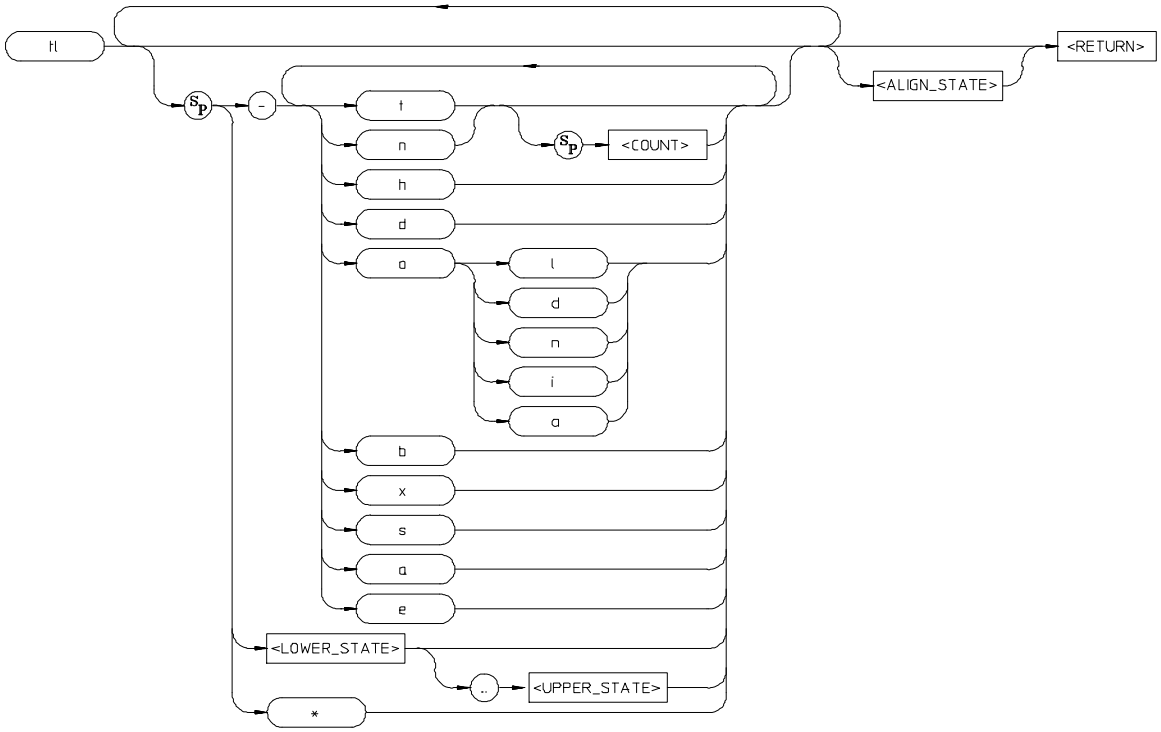
```
tsck -o 1
tsck -o 2
tsck -o 3
tsck -o 4
tsck -o 5
```

Trace Upon Execute

```
tx -d    # ignore the execute signal
```

See Also **init** (used to initialize selected portions of the emulator or the entire emulator, dependent on the options given)

tl



The **tl** command allows you to display the current emulation-bus analyzer trace list information.

The parameters are as follows:

t — Display the top states of the trace. If you have specified the number of states to display with the **<COUNT>** parameter, that number of states is displayed. Otherwise, the default is to display the same number of states as the last time **tl** was invoked to display part (but not all) of the trace.

n — Display the next states of the trace. If you have specified the number of states to display with the **<COUNT>** parameter, that number of states is displayed. Otherwise, the same number of states will be displayed as the last time you used **tl** to display part (but not all) of the trace.

<COUNT> <**COUNT**> allows you to specify the number of states to display with the **-t** or **-n** options.

h Normally, column headers are displayed at the top of each trace list. These label the state number, count, and each trace field specified by the **tf** command. Specifying the **-h** option allows you to suppress printing of the column headers.

d This option causes the analyzer to disassemble the content of its trace list, starting at the trace-list line number you include in this command. This results in a trace list that appears to be an assembly language program listing.

o By specifying **-o** and **<IALOPTS>**, you can control disassembly of the trace list. The following table lists the **<IALOPTS>** supported.

Option	**Meaning**
-od	Dequeue the trace list; that is, match opcodes with the associated operands. You can help the match of opcodes with operands by including the line number of the first instruction to be disassembled and the line number of its corresponding operands in your command (e.g. **tl -d -od 50 62**, meaning align operands on trace-list line number 62 with the instruction on line number 50).
-on	Don't dequeue the trace list; that is, show the list in the order that the bus cycles appeared.
-oi	Display instructions only; that is, don't display operand cycles as separate states.
-oa	Display both instruction and operand bus states.
-ol	Disassemble the trace list beginning with the low word at the specified trace-list line number (the default is to disassemble from the high word).

b The **-b** option dumps the trace list in binary format using the HP 64000 **transfer** protocol. See Chapter 13, "Data File Formats," in this manual for details on the binary trace list format.

x The **-x** option dumps the trace list in hexadecimal format using the HP 64000 **transfer** protocol. See Chapter 13, "Data File Formats," in this manual for details on the hexadecimal trace list format.

The **-h**, **-d**, and **-o** options cannot be used with either **-b** or **-x**. Also, the **-b** and **-x** options cannot be used together.

s	This allows you to display symbols in the address column.
a	This (the default) allows display of absolute addresses in the address column.
e	This allows you to display symbols and absolute addresses in the address column.
	The HP 64700 remembers the last option specified for the address field (**-s**, **-a**, or **-e**), and uses it for the next **tl** command if no other option is specified.
*	If you specify *****, the entire trace list is displayed. Notice that **tl** does not recognize displaying the entire trace as the last default count. (This helps avoid filling your screen with lots of trace list data on subsequent **tl** commands.)
<LOWER_ STATE>	If you specify <**LOWER_STATE**>, the trace display starts with the state on that trace-list line number.
<UPPER_ STATE>	If you specify both <**LOWER_STATE**> and <**UPPER_STATE**>, the trace list contains all states between the lower and upper trace-list line numbers, inclusive.
	If you specify a lower state, it must be done without using the **-t** or **-n** options, because the Terminal Interface will interpret your lower state specification as a <**COUNT**> parameter. However, you can specify a range of states while using these options; the range will be interpreted and displayed correctly.
<ALIGN_STATE>	If you specify the **-od** option to dequeue the trace list, and you specify <LOWER_STATE> for the first state to display, then <ALIGN_STATE> specifies the operand associated with the instruction fetched in <LOWER_STATE>.
	If no parameters are given, the trace list is displayed starting with the first state that has not yet been displayed. The number of states displayed is identical to the number of states displayed by the last **tl** command.
	For example, if the last trace list display was **tl -td 5**, then the next **tl** command will start the display at state 6 and display a total of five states.
	The **-a** option is in effect by default, which causes the address field to display absolute addresses.
	The trace list also defaults to the last disassembly state used (that is, if **-d** was specified previously in a **tl** command, it will continue).

Examples

To return to the top of the trace list and disassemble instructions, type:

```
U> tl -td
```

To vary the number of states displayed, type:

```
U> tl -td 5
```

To display a range of states, type

```
U> tl -td 20..30
```

To suppress display of the column headers, use the **-h** option:

```
U> tl -h
```

To align the instruction on trace list line number 38 with the operand cycles on line number 47, enter the command:

```
U> tl -d -od 38 47
```

You may also dump the trace list to a host computer using the **-b** (binary) or **-x** (hexadecimal) options in conjunction with the HP 64000 **transfer** software. This allows you to perform post processing of the trace data on your host. See Chapter 13, "Data File Formats," for details of trace list formats.

To display a trace list from a trace in progress, the trigger specification must be satisfied. Otherwise, halt the trace with the **th** command. Entering **tl** before the trace is halted displays the message "**** Trigger not in memory ****." If the analyzer was halted before any states were captured, the message "**** No trace data ****" is displayed upon entry of the **tl** command.

See Also

t (starts an analyzer trace)

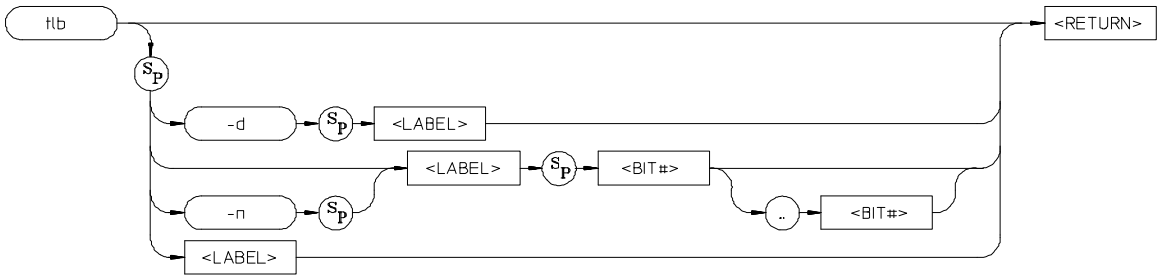
tf (specifies the display format for the trace)

th (halts a trace in process)

tlb (defines analyzer signal line labels; these may be used by **tf** in specifying the trace list display format)

ts (allows you to determine the current status of the emulation-bus analyzer)

tlb



The **tlb** command allows you to define new labels for emulation-bus analyzer lines, as well as display or delete previously defined analyzer labels.

The parameters are as follows:

-d — If you specify the **-d** option with a <LABEL>, the named label is deleted from the definition table. If the <LABEL> is currently used in a trace specification or in the trace display format (**tf** command), it will not be deleted until removed from all of the specifications. If <LABEL> is given as *****, all labels are deleted.

-n — Specifying **-n** causes the named <LABEL> to be defined with negative polarity. That is, after label definition, one (1) bits indicate a signal lower than the threshold voltage and zero (0) bits indicate a signal higher than the threshold voltage. If **-n** is not specified, the <LABEL> defaults to positive.

<LABEL> — You use <LABEL> to specify a name for the group of signals indicated by <BIT_RANGE>. <LABEL> is an alphanumeric designator; upper and lower case are significant. Labels can have up to 31 characters. If <LABEL> is supplied without an option, the named label is displayed; if <LABEL> is given as *****, all of the label definitions are displayed.

<BIT#> — <BIT#> specifies first the lower (or only), then upper, bits of the range to be assigned to the named <LABEL>. If more than one bit is specified (creating a range), the bit numbers are separated by two periods (..).

If no parameters are specified, the current label definitions are displayed. At emulator powerup, or after **tinit**, the only label definitions are the address, data, and status labels needed to operate the emulation-bus analyzer. All new label definitions default to positive polarity unless the **-n** option is given.

Examples

Define a label which will overlap the lower data bus byte:

```
M> tlb byte3 56..63
```

View the label definitions:

```
M> tlb
```

In the trace list, view only the output write data on the lower data byte in ASCII format:

```
M> tf lowerdata,A
```

<BIT>..<BIT> specifies the range of analyzer lines to be associated with <LABEL>. Note that it is not necessary to specify an upper boundary; if only one bit number is given, it is the only one that will be associated with the given label.

The emulation-bus analyzer, dependent on the particular emulator in use, has between 32 and 80 lines, where 0 is the least significant bit.

In emulation-bus analyzer labels, no more than 32 signal lines may be assigned to a given label. Also, an emulation-bus analyzer label may not cross more than a multiple of 16 boundary. For example, a label cannot be defined for emulation-bus analyzer lines 15..32 because one multiple of 16 boundary is crossed from 15 to 16 and another boundary is crossed from 31 to 32.

Labels can be made to overlap; for example, you may wish to define a label for a particular status line or data bit so that you can easily track its state in the trace list.

The number of labels that can be defined is limited only by system memory.

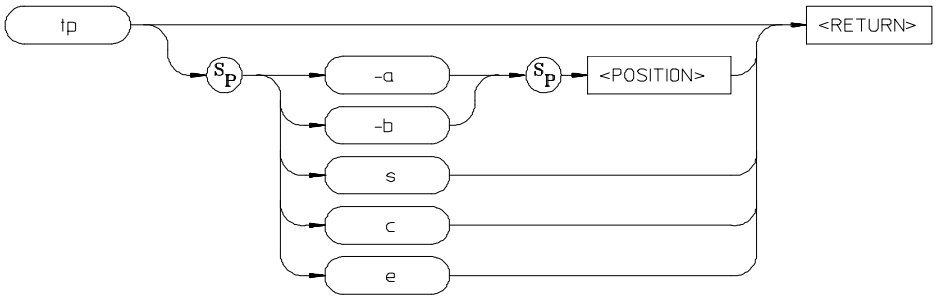
See Also

tf (used to specify the trace list format; **tlb** <LABEL> definitions can be specified as output columns in the trace listing through the **tf** command)

tpat (trace pattern definition; labels defined in **tlb** can be used in pattern definitions)

trng (trace range, used to specify a range of valid values to be used in a trace specification; labels defined by **tlb** may be used in defining the trace range)

tp



The **tp** command allows you to specify where the trigger state will be positioned within the trace list.

The parameters are as follows:

-a — Specifying **-a** along with a <POSITION> parameter indicates that the trigger is to be placed in the trace list with <POSITION> number of states after the trigger position to the end of the trace. That is, there will be <POSITION> number of states between the trigger position and the end of the trace.

-b — Specifying **-b** along with a <POSITION> parameter indicates that the trigger is to be placed in the trace list with <POSITION> number of states before the trigger position to the beginning of the trace. That is, there will be <POSITION> number of states between the beginning of the trace and the trigger position.

<POSITION> — **<POSITION>** is a decimal value from 0 to 1023 (or 0 to 511 if **tcq** is in effect) specifying the number of states positioned before or after the trigger state, depending on the option supplied.

s — If you specify the **s** parameter, the trigger is positioned at the start of the trace list.

c — If you specify the **c** parameter, the trigger is positioned at the center of the trace list.

e — If you specify the **e** parameter, the trigger is positioned at the end of the trace list.

If no parameters are supplied, the current trigger position setting is displayed. Upon powerup or after **tinit**, the trigger position is **tp s**.

Examples

To display the current setting of the trigger position, type:

```
M> tp
```

To define a trigger at address demo:Loop, type:

```
M> tg addr=demo:Loop
```

When you run the program and display the trace list, note that the trigger (always state zero (0)) will be positioned at the start of the trace.

To move the trigger to the end of the trace, type:

```
M> tp e
```

When you display the trace, note that state 1 will be empty. (You must rerun the trace to see the changes.)

To position the trigger at the center of the trace list, type:

```
M> tp c
```

To position the trigger so that 10 states are displayed after it, type:

```
M> tp -a 10
```

When you display the trace list, note that 11 states will be displayed after the trigger. This is within the specified accuracy of the system.

To position the trigger so that 5 states are displayed before it, type:

```
M> tp -b 5
```

When you display the trace list, note that four states will be displayed before the trigger, which again is within the system's positioning accuracy.

If the trace tag counter (**tcq**) is disabled, the position number specified has an accuracy of +/- 3 states; otherwise, the accuracy is +/- 1 state.

See Also

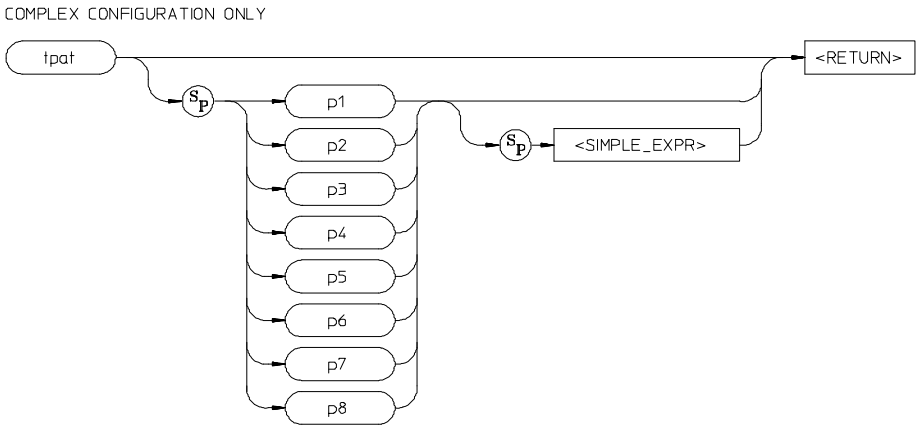
tcq (used to specify the trace count qualifier; affects the number of states that can be stored by the analyzer)

tg (defines the trigger expression)

tl (used to display the trace list)

tsq (used to specify the trigger position within the trace sequencer; reference the sequencer operation when deciding where to position the trigger in the trace list, if you want to capture all of the sequence conditions)

tpat



The **tpat** command allows you to assign pattern names to simple analyzer expressions.

The parameters are as follows:

p1 - p8 — The labels **p1** through **p8** are the names assigned to each simple expression. (The **p** in the label must be lowercase.)

<SIMPLE_EXPR> — **<SIMPLE_EXPR>** lets you directly specify an analyzer expression to use as a storage qualifier. For example, **<SIMPLE_EXPR>** might consist of the expression **addr=2000**. For detailed information on specification of simple expressions, see Chapter 11, “Expressions.”

Simple expressions assigned to patterns are restricted from the standard <SIMPLE_EXPR> definition in that you may not assign a range of values to a given label; only one value is permitted. (However, in actual practice, it is sometimes possible to circumvent this restriction by careful choice of don’t care values in the expression.)

Also, patterns can be specified that encompass more bits than the number of bits defined for the specified label. When this occurs, the upper bits are truncated.

If no parameters are given, or if the pattern name is given as *****, all eight of the current pattern assignments are displayed. If one of the pattern names is given, the expression assigned to that pattern is displayed.

Upon entering complex configuration after powerup or a **tinit** initialization, all eight patterns are defined as **tpat <pattern#> any**.

Examples

Set pattern assignments using the symbols Cmd_A, Cmd_B, and Cmd_I:

```
M> tpat p1 addr=handle_msg:Cmd_A
M> tpat p2 addr=handle_msg:Cmd_B
M> tpat p3 addr=handle_msg:Cmd_I
```

To set up a trigger when any one of the above patterns will trigger the analyzer, type:

```
M> tg p1|p2|p3
```

Set equates to trigger the analyzer on various commands received at the command input location:

```
M> equ inputa=41
M> equ inputb=42
M> equ notacommand=00
```

To set up various pattern combinations for these equates, type:

```
M> tpat p1 addr=demo:Cmd_Input and data=inputa
M> tpat p2 addr=demo:Cmd_Input and data=inputb
M> tpat p3 addr=demo:Cmd_Input
M> tpat p5 data=inputa
M> tpat p6 data=inputb
M> tpat p7 data!=notacommand
M> tpat p8 data=notacommand
```

To ensure that a symbol is recognized on a long-word-aligned boundary:

```
R> tpat p1 addr=~3&demo:Loop
```

The **tpat** command is only valid in the complex analyzer configuration (**tcf -c**).

See Also

tcf (defines whether the analyzer is in easy configuration or complex configuration; the **tpat** command is only valid in complex configuration)

tcq (specifies a trace count qualifier; **tpat** patterns may be used in complex configuration qualifier specification)

telif (specifies a secondary branch qualifier in analyzer complex configuration; **tpat** patterns may be used in qualifier specification)

tg (used to specify a simple trigger in either easy configuration or complex configuration; **tpat** patterns may be used in complex configuration trigger specification)

tif (used to specify a primary branch qualifier in either analyzer configuration; **tpat** patterns may be used in complex configuration branch specifications)

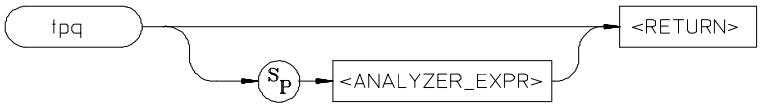
tpq (specifies a trace prestore qualifier; **tpat** patterns may be used in qualifier specification)

trng (defines a range of values on a set of analyzer input lines; this range may be used in conjunction with the patterns defined by **tpat** in setting up complex analysis qualifiers)

tsq (used to manipulate the trace sequencer)

tsto (used to define global storage qualifiers in both analyzer configurations; may also be used to define storage qualifiers for each sequencer level in complex configuration. The patterns defined by **tpat** may be used in complex configuration storage qualifier definition.)

tpq



The **tpq** command allows you to specify a prestore qualifier for the trace.

The parameters are as follows:

<ANALYZER_EXPR>

<ANALYZER_EXPR> allows you to specify the expression to be recognized as a prestore state. This expression consists of a **<SIMPLE_EXPR>** in analyzer easy configuration and a **<COMPLEX_EXPR>** when the analyzer is in complex configuration. See Chapter 11, "Expressions," for specific details of analyzer expressions. In either configuration, the expression may consist of the states **any** (prestore all states) or **none** (disable prestore).

If no parameters are given, the current prestore qualifier setting is displayed. Upon powerup or after **tinit** initialization, the prestore qualifier defaults to **tpq none**.

Examples

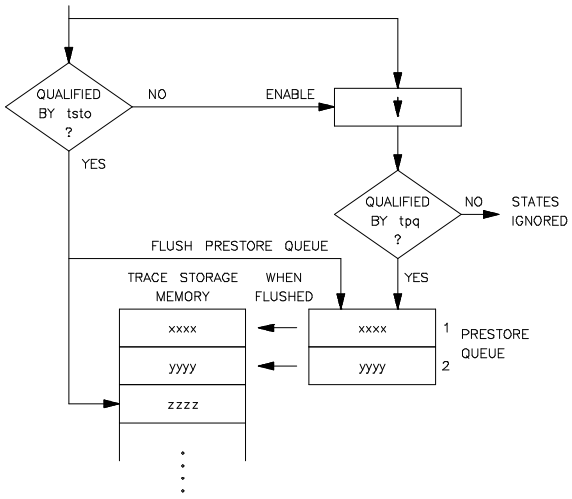
Display the current prestore qualifier:

```
R> tpq
```

Assume that you have three routines called wait_keyboard, wait_mouse, and wait_tablet. All three call a routine named delay_loop. You can see which routine called delay_loop by defining a prestore qualifier:

```
R> tcf -c
R> tpat p1 addr=wait_keyboard
R> tpat p2 addr=wait_mouse
R> tpat p3 addr=wait_tablet
R> tpq p1|p2|p3
```

During the trace, the analyzer fills a two stage pipe with states that satisfy the prestore qualifier. Each time a trace state is stored into the trace buffer, the prestore qualifier is also stored and then cleared. Therefore, up to two prestore events may be stored for each normal store event. The prestore events in the trace buffer will



correspond to the most recent states that satisfied the prestore qualifier immediately prior to a store event but following the previous store event.

Because the prestore memory shares trace memory with store events, the number of store events recorded will be reduced by the number of prestore states recorded.

See Also

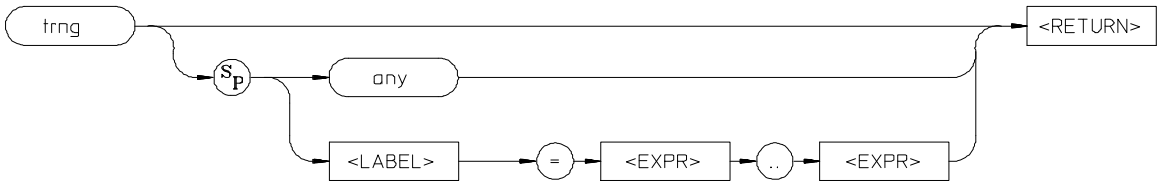
tcf (specifies whether the analyzer is to operate in easy configuration or complex configuration)

tsq (used to manipulate the trace sequencer)

tsto (used to specify a global storage qualifier for both easy configuration and complex configuration; also used to specify individual sequence term storage qualifiers in complex configuration)

trng

COMPLEX CONFIGURATION ONLY



The **trng** command lets you specify a range of acceptable values for an analyzer trace label.

The parameters are as follows:

any — When you specify **any**, all possible patterns on all labels will satisfy the range specification.

<LABEL> — **<LABEL>** specifies the group of signal lines to which a range is assigned. These might be **addr, data,** or **stat**; or, they may be a label that you have defined. See the **tlb** command syntax pages for information on defining labels.

<EXPR> — **<EXPR>** allows you to specify first the lower, then upper, boundaries of the range of patterns to be considered valid range entries. For example, to define the address range of 2000 through 21ff hex, you would specify the **<EXPR>** range as **2000..21ff**. Note the two periods used as a separator between the lower and upper range bounds; no additional spaces are included.

Also, the first boundary specified must be less than or equal to the second boundary specified (example: **trng addr=2000..21ff** is correct; **trng addr=21ff..2000** is incorrect). You may also specify a single value for the range (example: **trng addr=2000**).

See Chapter 11, “Expressions,” for details on expression syntax.

Ranges can be specified that encompass more bits than the number of bits defined for the specified label.

If no parameters are supplied, the current range definition is displayed. After powerup or **tinit** initialization, the **trng** command is set to **trng any**. (Note that **trng** is not directly available after analyzer initialization; the analyzer is set to easy configuration when initialized. You must then switch to complex configuration to access **trng**.)

The **tcf -e** (set trace configuration to easy) command also will reset **trng**. In other words, any **trng** defined when the analyzer was in complex configuration is destroyed when the analyzer is set to easy configuration; you cannot return to complex configuration and use the old **trng**.

Examples

Trigger the analyzer on any access to the message storage area of the demo program:

```
M> tcf -c
M> trng addr=handle_msg:Msg_A..End_Msgs
M> tg r
```

The range of values specified by **trng** may then be used in complex qualifiers for the trace specification. The **trng** command is only available in the analyzer's complex configuration (see **tcf** syntax pages).

There is no need for a not equals operator in specifying ranges, as the trace specification commands which allow "range" as a parameter also accept "not range" in the form **!r**.

See Also

tcf (sets analyzer to complex or easy configuration; analyzer must be in complex configuration to utilize the **trng** command)

tcq (trace state/time counter; in complex configuration, states can be counted using the range specification)

telif (specifies the sequencer secondary branch expression; in complex configuration, this expression can include references to the range)

tg (specifies analyzer trigger; may trigger on references to range)

tif (specifies the sequencer primary branch expression; in complex configuration, branch expression may include range qualifier)

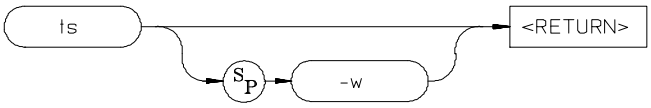
tpat (trace pattern definition; assigns pattern names to simple expressions for later use in analyzer specifications. **tpat** essentially commits only one pattern to a label; whereas **trng** allows a range of values to be assigned to the range pattern)

tpq (defines trace prestore qualifier; the range specification may be used in complex configuration prestore qualifier expressions)

tsq (trace sequencer definition)

tsto (defines trace storage qualifier; that is, specifies exactly what states are actually to be stored by the analyzer. In complex configuration, this can include states that fall within the specification defined by **trng**)

ts



The **ts** command allows you to determine the current status of the emulation-bus analyzer.

The parameters are as follows:

-w

The **-w** option indicates that the trace status should be printed in whisper mode; which gives an abbreviated version of the status.

If the whisper option is not specified, the long version of trace status is displayed.

Examples

To view the trace status, type:

```
U> ts
```

To display the short form of the status, type:

```
U> ts -w
```

Trace Status Displays

The trace status is displayed in the following form:

```
---Emulation Trace Status---
(NEW) [User | CMB ] trace [complete | halted | running ]
Arm [ ignored | (not) received ]
Trigger (not) found
Arm to trigger armcount
States visible (history) first..last
Sequence term term
Count remaining count
```

Status Display Interpretation

The first line of the trace status indicates the initiator of the trace, whether the trace is completed, running, or halted, and whether or not this trace has been displayed.

NEW	This trace has not been displayed. The **tl** command will clear this flag until the next trace is started. Halting a trace that is running (as opposed to complete), marks the trace as being NEW even though the trace may have been displayed while running. The next **tl** command with no options will list the trace from the top.
User	The operator initiated this trace with the **t** command.
CMB	This trace was initiated by a /EXECUTE pulse on the CMB after a **tx** command was entered.
complete	The trace has found its trigger and completed.
halted	The trace was halted in response to a **th** command.
running	The trace is still running; either the complete sequencer specifications have not yet been satisfied; or not enough qualified store states have been found to fill trace memory.

The second line of the trace display indicates the analyzer arm status.

ignored	The arm condition specified for this trace was **tarm always.**
received	The arm condition has been satisfied.
not received	The arm condition was not satisfied. (If you specified an arm condition but didn't use it in trigger qualification, this will be displayed if the arm condition is not satisfied. However, the analyzer may still find the correct trigger and complete the trace.)

The third line of the state trace display indicates the trigger status. Because of the pipelined analyzer architecture, it is possible that the trace status may display "not found" when in fact the trigger has been found. This will occur when not enough states satisfying the storage specification are found to push the trigger out of the pipeline and into trace memory. In any case, the trace will not be displayable until the trigger is in trace memory (unless you halt the analyzer).

found	The trigger condition has been found.
not found	The trigger condition has not yet been satisfied.

The fourth line of the trace display indicates the amount of time that passed between the arm signal and the trigger condition.

armcount — This will be from -0.04 usec to 41.94288 ms. The arm to trigger counter may underflow or overflow, in which case "<-0.04 uS" or ">41.94288 mS" are reported, respectively. If the arm signal was ignored, if the trigger was not found, or if the clock setting (tck) is fast (F) or very fast (VF), the character "?" (unknown) is displayed.

The fifth line of the display indicates the number of states displayable by **tl**.

visible — Number of states which can be displayed by **tl**. This will be a number from 0 to 1024 (or 0 to 512 if **tcq** is active).

history — Number of states which can be displayed if the current trace is halted; this may include history states which may be overwritten and thus unavailable if the current trace runs to completion.

first — Number of the first state stored in trace memory, relative to the trigger state. This will be a number from -1024 to 0 (-512 to 0 if **tcq** is active). The character "?" is displayed if the trigger state is not yet in memory.

last — Number of the last state stored in trace memory, relative to the trigger state. This will be a number from -1 to 1023 (-1 to 511 if **tcq** is active). The character "?" is displayed if the trigger state is not yet in memory.

The sixth line of the trace display indicates the current sequencer term position.

term — Current sequence term position (1 through 5 in easy configuration; 1 through 8 in complex configuration). If the trace is completed or halted, the last sequence term number is displayed. A "?" is displayed if the trace is running and the sequencer is running too quickly for the current term number to be read.

The seventh line of the trace display indicates the count qualifier status for the primary branch condition of the current sequence term, see **tif** for further details.

count — Remaining number of occurrences of the primary branch qualifier needed to satisfy the qualifier so that the primary branch will be taken. A "?" is displayed if the trace is running and the counter is updating too quickly to be read.

Whisper Mode Trace Display

If the **-w** option is given, an abbreviated version of the trace status is given as follows:

Trace run status:

R - trace running

C - trace completed

H - trace halted

Trace arm status:

A - Arm has been received

a - arm has not yet been received

x - arm signal is being ignored

Trace trigger status:

T - trace trigger has been found

t - trace trigger has not yet been found

Trace list status:

***** - indicates that this trace has not been displayed

See Also

es (allows you to determine general emulator status)

t (starts a trace)

tarm (arm the analyzer based on state of the trig1 and trig2 signals)

tcq (specify trace tag counter; affects number of states that the analyzer can store)

tg (specify the analyzer trigger state)

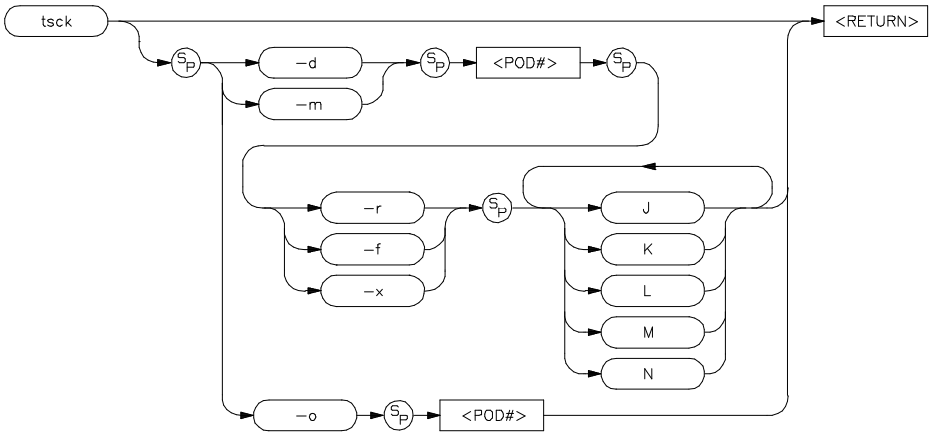
th (halt the current trace in process)

tif (specify sequencer primary branch condition and number of occurrences)

tx (specify that trace is to begin upon receiving the CMB /EXECUTE pulse)

x (begin a synchronous CMB execution)

tsck



The **tsck** command allows you to specify the slave clock edges used for the emulation-bus analyzer trace.

The parameters are as follows:

d

The **-d** option allows you to specify that the slave clock operates in demultiplexed mode. In this mode, the lower eight channels of the analyzer pod (bits 0-7) are latched with the slave clock and the upper eight channels (bits 8 through 15) are replaced with the lower eight channels. In other words, the upper eight bits are identical to the lower eight at the pod.

However, the data is not clocked into the analyzer itself until the next master clock occurs. Therefore, if no slave clocks have occurred since the last master clock, the data on the lower eight analyzer lines is identical to the upper eight. If one or more slave clocks have occurred since the last master clock, the data on the lower eight bits is the only data available to the analyzer.

When using the **-d** option, you must specify one of the **-r**, **-f**, or **-x** options to indicate the active edge(s) of the slave clock.

m

The **-m** option specifies that the slave clock operates in mixed mode. In the mixed mode, the lower eight channels of the analyzer pod (bits 0-7) are latched with the slave clock, and the master clock latches in the entire pod. Therefore, if no slave clock has occurred since the last master clock, the data on the lower eight bits of the pod will be clocked into the analyzer at the same time as the upper eight bits. If

more than one slave clock has occurred since the last master clock, only the first slave clock data will be available to the analyzer.

When using the **-m** option, you must specify one of the **-r**, **-f**, or **-x** options to indicate the active edge(s) of the slave clock.

<POD#> Specifies one of five groups of analyzer input lines. These are as follows:

r Indicates that the pod should latch data on the **rising** edge of the slave clock.

f Indicates that the pod should latch data on the **falling** edge of the slave clock.

x Indicates that the pod should latch data on **both** edges of the slave clock.

CLOCK SIGNALS The **r**, **f**, and **x** operators may be used on the following clock signals: **J, K, L, M** or **N**. Clocks **L**, **M**, and **N** are generated by the emulator. Clocks **J** and **K** are not used.

If you specify multiple clocks, any one of the clock edges (as defined by the **r**, **f**, and **x** options) will clock the trace.

o If you specify **-o** with a <**POD#**>, the slave clock is ignored on that pod.

If no parameters are specified, the current slave clock definitions are displayed. The default for all slave clocks is **off** after powerup or **tinit** initialization.

Examples

To display the current state of the slave clock specifications, type:

```
M> tsck
```

Each analyzer pod has the capability of latching certain signals with a slave clock instead of the master clock. (You set up the master clock with the **tck** command.) You should generally not use this command. It is provided for use by HP 64700 high-level interfaces.

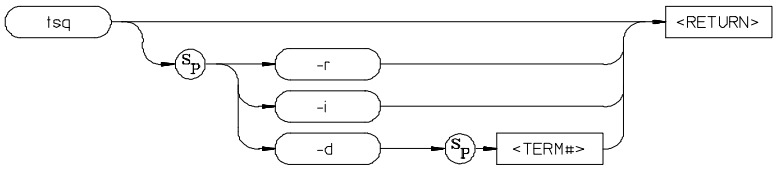
See Also

ta (allows you to display active signals on the analyzer input lines; useful in verifying that you have selected the correct clock conditions)

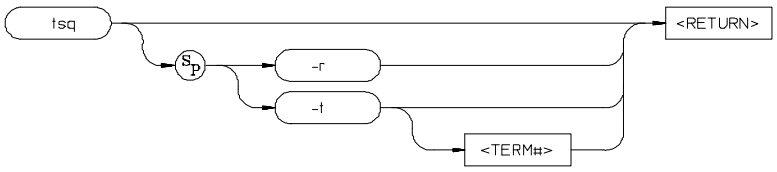
tck (used to define master clock signals used by the analyzer; **tsck** defines the slave clock signals. Default mode for **tsck** is off on all pods.)

tsq

EASY CONFIGURATION



COMPLEX CONFIGURATION



The **tsq** command allows you to manipulate or display the trace sequencer.

The parameters for easy configuration are:

r

If you specify **r**, the sequencer is reset to a simple one term sequence which stores all states and triggers on the first occurrence of any state. This is equivalent to issuing the commands:

tg any
tsto any
telif never

i

Specifying **i** in conjunction with a **<TERM#>** inserts a new sequence term at **<TERM#>**. The new sequence term will use the default storage qualifier (which can be modified with the **tsto** command). It will also use the secondary branch qualifier (global restart in easy configuration) specified by the **telif** command.

If there is already a sequence term with number **<TERM#>**, terms with number **<TERM#>** and above will be renumbered (**<TERM#>** becomes **<TERM#> + 1**) to make room for the new term.

The primary branch qualifier for the new term will be defined as **tif <TERM#>** any unless it is the last term in the sequence (by definition, the trigger term), in which case the primary branch qualifier is set to **tif <TERM#> never**.

d — Specifying **d** in conjunction with a **<TERM#>** deletes the term specified and renumbers higher numbered terms downward to fill the gap.

<TERM#> — **<TERM#>** specifies a term number in the range 1 through 4 to insert in the sequencer (**-i**) or remove from the sequencer (**-d**). You must insert terms in a contiguous manner; for example, you cannot insert a term number 4 if the sequencer only has two terms defined. Instead, you must next insert a term numbered 1, 2 or 3.

The parameters for complex configuration are:

r — If you specify **-r**, the sequencer is reset to an eight term sequence with the trigger term at term number 2. The sequencer will be set to **tsto any** (store any state). All secondary branch qualifiers are turned off (**telif <TERM#> never**), and all primary branch qualifiers will jump to the next higher numbered term on any state (**tif <TERM#> any (<TERM#> +1)**).

t — Specifying **-t** by itself displays the trigger term. You can define which term is to be the trigger term by specifying **-t** along with a **<TERM#>.** The analyzer will trigger on the first entrance to the term from either a primary or secondary branch.

<TERM#> — **<TERM#>** specifies a term number in the range 2 through 8 to use as the trigger term.

If no options are given, all of the sequencer storage and branch qualifiers are displayed along with the trigger term position. Upon powerup or after **tinit** initialization, the sequencer defaults to the following state:

```
tif 1 any
tsto all
telif never
```

In other words, the sequencer powers up with two sequence terms; the second sequence term is the trigger term. Any state will cause a branch from the first term to the second term; global restart is set to never and all states are stored by the analyzer.

Switching analyzer configurations from easy to complex or vice versa also resets the sequencer (that is, **tcf -c** or **tcf -e**).

Examples

View the state of the sequencer after powerup or a **tinit**:

```
M> tsq
```

While still in easy configuration, insert two sequence terms:

```
M> tsq -i 2
M> tsq -i 3
```

To delete a sequence term in easy configuration, type:

```
M> tsq -d 3
```

To change the trigger term in complex configuration:

```
M> tcf -c
M> tsq -t 5
```

When the analyzer is in easy configuration (**tcf -e**), the sequencer has a maximum of four sequence terms with a minimum of one term.

If the analyzer is in complex configuration (**tcf -c**), the sequencer always has eight terms (although your sequencer setup may only use two terms). Any term except term 1 can be the trigger term. Each term has a primary and secondary branch, which can dictate progression to other sequence terms.

With microprocessors that prefetch instructions, it is often more accurate to base trace conditions on data movement resulting from an instruction rather than the instruction itself. When the data pattern is found, it is more likely that the instruction executed. Such methods must be used with care; in some programs, several different routines may execute the same data movement.

See Also

tcf (defines whether analyzer is operated in complex or easy configuration)

telif (sets global restart qualifier in easy configuration; secondary branch qualifier in complex configuration)

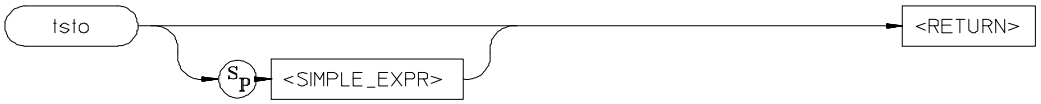
tg (defines the trigger qualifier)

tif (sets the primary branch qualifier in both easy and complex configuration)

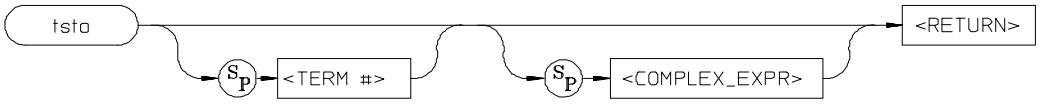
tsto (defines the analyzer global storage qualifier)

tsto

EASY CONFIGURATION



COMPLEX CONFIGURATION



The **tsto** command allows you to specify a trace storage qualifier for the emulation-bus analyzer.

The parameters for easy configuration are:

<SIMPLE_ EXPR>

<SIMPLE_EXPR> lets you directly specify an analyzer expression to use as a storage qualifier. For example, **<SIMPLE_EXPR>** might consist of the expression **addr=2000**. For detailed information on specification of simple expressions, see Chapter 11, “Expressions.”

The parameters for complex configuration are:

<TERM#>

<TERM#> lets you specify a sequence term number to associate with the given **<COMPLEX_EXPR>**. When you associate a term number with a complex expression, that expression is only used as a storage qualifier at the sequencer level specified by the term number. If you specify **<TERM#>** without an expression, the complex expression currently associated with that term number is displayed. If you specify an expression without including a **<TERM#>**, the expression is used as a global storage qualifier; that is, the storage qualifiers of all eight sequence terms are set to the same value as the global storage qualifier you specified.

If you’ve specified a global storage qualifier, you can override any of the sequence term storage qualifiers by specifying the term number along with the new qualifier. For example, you might specify a global storage qualifier of **tsto any**; you could override this for term 3 by specifying **tsto 3 none**.

<COMPLEX_ EXPR>

<COMPLEX_EXPR> allows you to specify complicated analyzer expressions made up of relationships between simple analyzer expressions. When you create a

complex expression, you must first assign pattern names (**p1**-**p8**) to simple expressions using the **tpat** command. You then use the pattern names and relational operators to create complex expressions. For example, if you wish to store only the states having **address=2000** and **data=20** or the states having **address=2000** and **data=42**, you would use the following commands:

```
U> tpat p1 addr=2000 and data=20
U> tpat p2 addr=2000 and data=42
U> tsto p1 | p2
```

The "|" symbol represents an intra-set OR operator. For more information on complex expressions, operators, and pattern sets, see Chapter 11.

If no parameters are given, the current trace storage qualifier settings are displayed. Upon powerup or after **tinit** initialization, the trace storage qualifier defaults to **tsto all**. Using the **tcf** command to switch from complex configuration to easy configuration or vice versa will also reset the storage qualifier to **tsto all**.

Example

See Chapter 5, "Using the Analyzer," for an example of using storage qualifiers.

The expression parameter, whether <**SIMPLE_EXPR**> or <**COMPLEX_EXPR**>, specifies the type of data to be stored by the analyzer.

If the analyzer is in easy configuration (**tcf -e**), the expression is specified by <**SIMPLE_EXPR**> and this serves as a global storage qualifier. In other words, the same expression is used as a storage qualifier, regardless of the current sequencer state.

If the analyzer is in complex configuration (**tcf -c**), the expression is specified by <**COMPLEX_EXPR**> and may be assigned to a sequencer state with the <**TERM#**> parameter. When an expression is assigned to a specific term number, the analyzer will only store states corresponding to the given expression when at the given sequencer level. If no <**TERM#**> is given, the associated expression is defined as global; the analyzer stores states satisfying the expression, regardless of the sequencer level.

Remember that the analyzer only stores states for a given sequence term which satisfy the **tsto** qualifier for that term **while at that sequencer level**. If you specify storage of items in a particular term that occur **after** that term has been satisfied, the sequencer will no longer be at that level and therefore won't store the states you specified.

See Also

tcf (used to specify whether the analyzer is in easy configuration or complex configuration)

telif (used to specify a global restart qualifier in easy configuration; specifies a secondary branch qualifier for each sequencer level in complex configuration)

tg (used to specify a trigger condition in either easy configuration or complex configuration; overrides the current sequencer specification. Note that **tg** does not affect **tsto**; therefore, the current **tsto** specifications remain in effect whenever a **tg** command is entered)

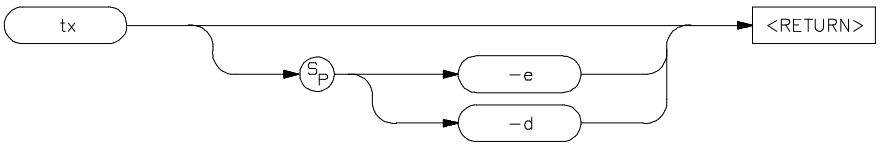
tif (used to specify a primary branch qualifier in either analyzer configuration)

tpat (used to assign pattern names to simple analyzer expressions for use in constructing complex analyzer expressions; these expressions can be used in specifying storage qualifiers for the **tsto** command)

trng (used to specify a range of values of a set of analyzer inputs; this range information can be used in constructing complex configuration qualifiers for the **tsto** command)

tsq (used to manipulate the trace sequencer)

tx



The **tx** command allows you to specify that the analyzer will begin a measurement when the CMB /EXECUTE line is asserted.

The parameters are as follows:

-e If you specify the **-e** option, the analyzer will start a measurement upon receiving the CMB /EXECUTE signal.

-d If you specify the **-d** option, the analyzer will NOT start a measurement upon receiving the CMB /EXECUTE signal.

If no options are specified, the current state of **tx** enable/disable is displayed. Upon powerup or after a **tinit**, the system defaults to **tx -e**.

Examples

Verify the current setting of **tx**:

```
M> tx
```

To set up a CMB measurement such that the emulator starts running and an analyzer measurement begins at address demo:Main whenever the CMB /EXECUTE pulse is received, type the following commands:

```
M> cmbt -d none
M> tx -e
M> tg addr=demo:Main
M> rx demo:Main
```

If **tx -e** is given, enabling measurement on execute, the CMB trigger is immediately driven true upon receiving the /EXECUTE signal. If the analyzer is not driving either trig1 or trig2, it is then started. The CMB trigger is then disabled and the HP 64700 waits for all other participants in the measurement to release the CMB

trigger. When the last instrument releases the CMB trigger, the trigger will go false; at this point any analyzers driving trig1 or trig2 will be started.

See Also

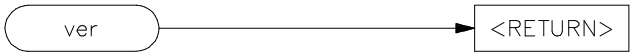
cmbt (specifies whether the CMB trigger signal is driven or received by the internal trig1 and trig2 signals)

tarm (specifies the arm condition for the analyzer)

tg (specifies a trigger condition for the analyzer)

This page intentionally left blank.

ver



The **ver** command instructs the emulator to return the current emulator Terminal Interface software version numbers. You should use this command when you need to know the version number of your emulator Terminal Interface software to compare it to the *Firmware/Software Compatibility Note* for the HP64700 PC Interface or Softkey Interface software versions.

Examples

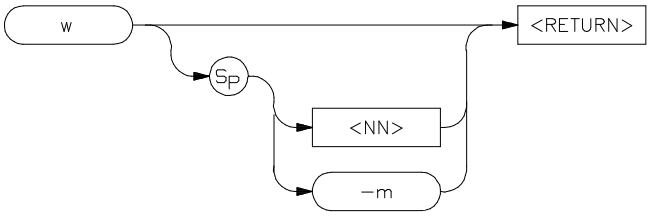
To determine the current emulator Terminal Interface software version numbers, type:

```
M> ver
```

The system returns a display similar to the following:

```
                  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.

  HP64700 Series Emulation System
    Version:   A.00.00 20Nov87


  HP64742 Motorola 68000 emulator
    Version:   A.00.00 20Nov87

    Speed:     12.5 MHz
    Memory:    126 KBytes

  HP64740 Emulation Analyzer
  Version:   A.00.00 20Nov87
```

W



The **w** command is used to program automatic waits into macros, repeats, and command files. Normal operation is to wait for any keystroke before executing the next operation; optionally, the wait can be programmed for a specific time period or for completion of a measurement in process (such as a trace).

The parameters are as follows:

<NN> Wait for NN number of seconds before proceeding.

-m Wait for completion of the current measurement before proceeding.

The default is to wait for any keystroke on the command port before proceeding.

Examples

To cause the emulator to wait for any keystroke before proceeding to the next command, type:

```
U> w
```

You might use this in a situation where you wish the operator to make a judgement regarding some other condition before proceeding with the next measurement. For example, if some LEDs in the target system should reach a certain state before a measurement is made, use the basic form of the wait command (**w**), which will allow the operator to verify that the LEDs have reached the proper state; then proceed with the next command by pressing any key.

To cause the emulator to wait for 32 seconds or for any keystroke, type:

```
U> w 32
```

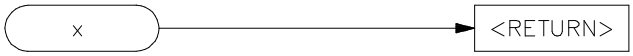
This might be used where you know the desired system state will be reached in a definite amount of time (or should be reached within that time).

To have the emulator wait until another measurement is completed or for any keystroke entry, type:

```
U> w -m
```

Note that the above examples, taken exactly as shown, don't provide you with a useful function—they are provided only to show correct examples of command line syntax. To use the wait command effectively, it should be applied within macros, repeat commands, or command files.

x



The **x** command allows you to initiate a synchronous CMB (Coordinated Measurement Bus) measurement execution.

Examples

To initiate a synchronous CMB measurement and have this HP 64700 emulator participate in the measurement, type the following commands:

```
M> rx 2000
M> tcf -e
M> tg addr=2000
M> tx
M> x
```

This enables the CMB and sets the run at execute address to 2000. The analyzer trigger is also set to 2000 hex and trace at execute is enabled. Finally, the **x** command is issued, initiating the coordinated execution. Other emulators on the CMB will respond per their **rx**, **tx,** and **cmb** commands.

When **x** is performed, the CMB /EXECUTE line is pulsed. If **tx** (trace at execute) is enabled, an analyzer measurement will begin. If the CMB is enabled via the **cmb -e** command, a break will occur, followed by a run at execute as specified by the **rx** command.

The **x** command is available whether CMB and trace at execute are enabled or not. Specifically, the **cmb** and **tx** commands control how this HP 64700 emulator will respond when an /EXECUTE or READY is detected. The **x** command only controls when this emulator will issue an /EXECUTE signal.

See Also

cmb (used to enable or disable interaction with the CMB)

rx (used to specify an address to start a program run when the /EXECUTE pulse is received from the CMB)

tx (used to specify that an analyzer measurement should begin when the /EXECUTE pulse is received from the CMB)

11

Expressions

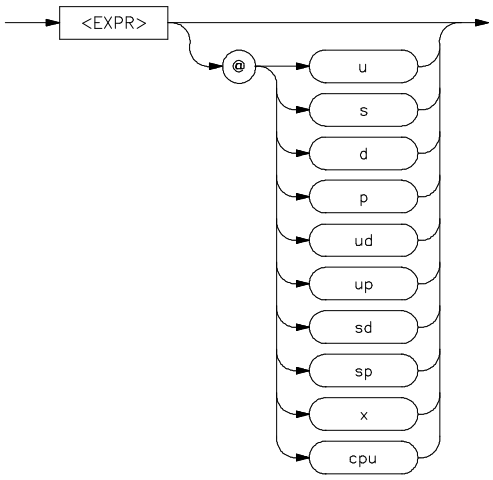
Numeric and logical expressions used in the Terminal Interface

This chapter includes information about these expression types:

- ADDRESS (address expressions)
- ANALYZER_EXPR (expressions in trace specifications)
- COMPLEX_EXPR (complex configuration expressions)
- EXPR (numeric expressions)
- SIMPLE_EXPR (easy configuration expressions)

The syntax, functional description, and related information is included for each expression type.

ADDRESS



The address expression (EXPR) allows you to enter an address in a form recognized by the MC68020 or MC68030/EC030 emulator. When you see the address variable in various syntax diagrams, remember that it is unique to the MC68020 and MC68030/EC030 emulators.

The <EXPR> must be a 32 bit-number. (If you supply less than 32 bits, the number is sign-extended to 32-bits). When you don't specify a base, such as "y" for binary, "o" for octal, or "t" for decimal, the default is "h" for hexadecimal. You can specify a function code to further qualify an address. The @ symbol is required if you specify a function code. Otherwise, it must be omitted.

Function codes can be any of the following:

u—User
s—Supervisor
d—Data
p—Program
ud—User Data
up—User Program
sd—Supervisor Data
sp—Supervisor Program
x—Don't Care
cpu—CPU space

Examples

Suppose you create the following memory map:

```
R>map 0..0fff eram
R>map 1000..1fff@d eram
R>map 1000..1fff@p eram
```

Now, the following memory display commands are valid:

```
R>m 0..0f
R>m 1000..100f@p
R>m 1000..100f@d
```

The following command is invalid, because the emulator can't determine which one of the ranges (program or data) you want to reference:

```
R>m 1000..100f
```

You can specify the base with the address. For example:

```
100t (100 base ten)
701o (701 base eight)
234o@sd (234 base eight in supervisor data space)
```

See Also

m (memory display/modify command)

map (specify mapping of memory)

mo (display or modify global access and display modes)

ANALYZER_EXPR

Analyzer expressions are used in specifying triggers, time qualifiers, primary and secondary branch conditions, prestore qualifiers, and other analyzer setup items. There are two types of analyzer expressions, simple and complex.

In a **simple expression**, the analyzer label is related to a numeric expression within an analyzer command. These expressions are required when the analyzer is in easy configuration (**tcf -e**).

Some examples include:

```
tg addr=2000

tif 1 data=20..30

telif addr!=3000 or data!=5
```

In a **complex expression**, the relationship between an analyzer label and an expression is assigned one of eight pattern identifiers or a range label. These patterns and the range are then used to create the actual expressions. Complex expressions are required when the analyzer is in complex configuration (**tcf -c**).

Some examples include:

First we assign a pattern name:

```
tpat p1 addr=2000

tpat p2 addr!=3000

tpat p5 data!=5

trng data=20..30
```

Then we create the actual complex expressions within the analyzer commands:

```
tg p1

tif 1 r
```

(**r** specifies the range defined with the **trng** command)

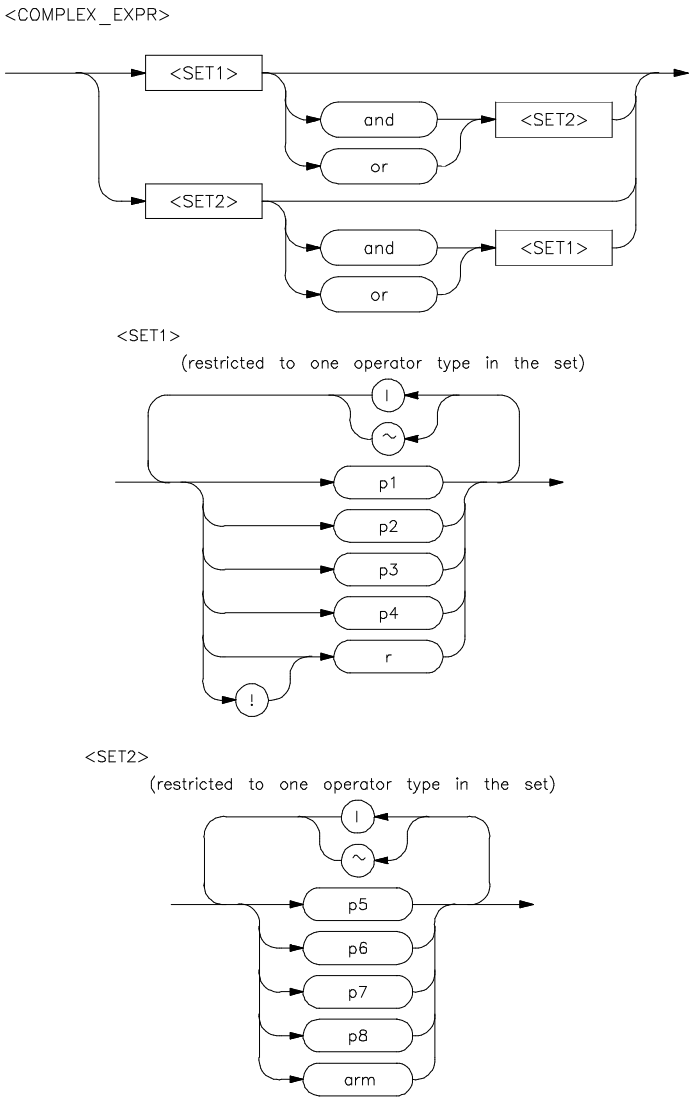
```
telif 1 p2 or p5 3
```

Any syntax diagram in this manual that indicates <ANALYZER_EXPR> means that a simple expression is required when the analyzer is in easy configuration, and a complex expression is required when the analyzer is in complex configuration.

See Also

See the <SIMPLE_EXPR> and <COMPLEX_EXPR> syntax pages for complete details on each expression.

COMPLEX_EXPR



In analyzer complex configuration (**tcf -c**) 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 **tpat** command. For example:

```
tpat p1 addr=2000
tpat p2 data!=00
tpat p3 stat=dma
tpat p4 addr=2000 and data=23
tpat p5 addr!=2105 and data!=0fc
```

You use the **trng** command to provide assign the range label:

```
trng data=42..44
```

Sets

The pattern labels, along with the range and arm specifications, are divided into two sets.

Set 1:

p1,p2,p3,p4,r,!r

Set 2:

p5,p6,p7,p8,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 the following types of intraset expressions:

```
p1~p2~r
```

(Pattern 1 NOR pattern 2 NOR range.)

```
p2 | !r
```

(Pattern 2 OR (NOT range).)

```
p5 | arm
```

(Pattern 5 OR arm.)

```
p6 ~ p8
```

(Pattern 6 NOR pattern 8.)

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**:

```
p2~p3|p4
```

(This is incorrect because the operator must remain the same throughout the set.)

```
p2~p5
```

(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.

```
p1~p2 and p5|arm
p3 or p6~p7~p8
```

However, you cannot repeat different sets to extend the expression. The following is **invalid:**

```
p1~p2 and p5 and p3 and p7
```

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:

```
tpat p1 addr!=2000
tpat p2 data!=23
```

Then you would OR these together using the intraset operators:

```
p1|p2
```

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:

```
tpat p1 addr!=2000
tpat p2 data!=23
```

Then you would NOR these together using the intraset operators:

```
p1~p2
```

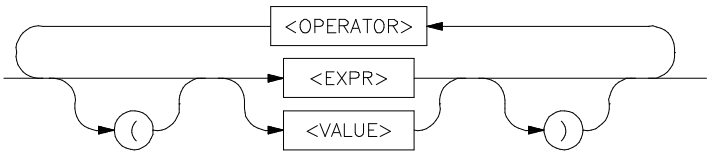
This is effectively the same as:

(NOT addr=2000) NOR (NOT data=23) = (addr=2000) AND (data=23)

See also

See the <EXPR> syntax pages for information on numeric expression specifications. See the <SIMPLE_EXPR> syntax pages for information on the types of simple expressions that may be assigned pattern names. See the <ADDRESS> syntax pages for information on address specifications.

EXPR

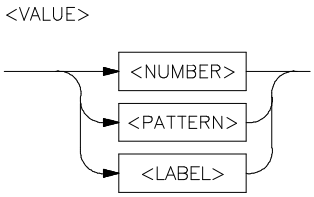


Numeric expressions are the root of all HP 64700 Terminal Interface expression types, including analyzer expressions, address specifications, equates, and expressions you might want to calculate using the **echo** command.

The expression capability in the Terminal 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.

Terminal Interface expressions consist of other **expressions** (recursion) 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), **labels** (only labels pointing to other numbers or patterns, assigned by the **equ** command), and symbols.

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 equates, echo, address specification, analyzer expressions, and any other specification relating to a microprocessor address, data or status value default to hexadecimal.

Numbers used to specify repeat count values, such as in the sequence branch commands, trigger, step, repeat command, and so on, default to decimal.

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. A place where you might want to use don't care values is to simulate a second range variable in complex mode specifications. For example, you might have:

trng addr=4000..4020

And you need a second range of **data** from 11 through 14 hex. Although it isn't perfect, you can simulate a second range by assigning the pattern label:

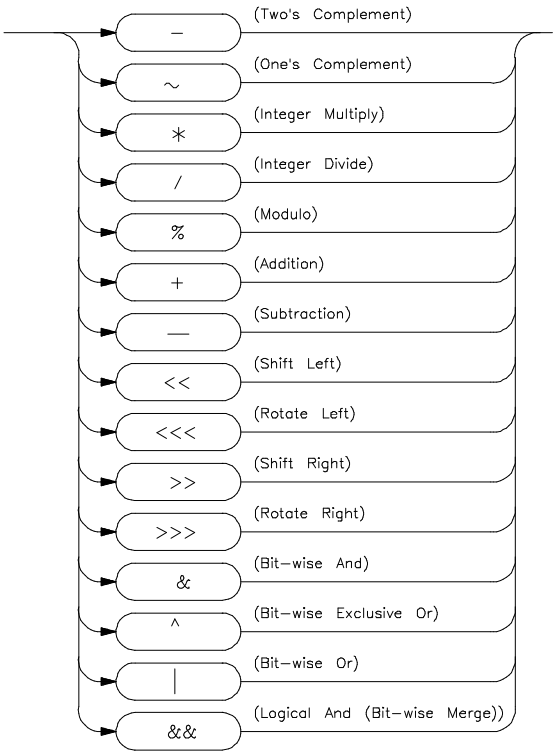
tpat p1 data=00010XXXy

(This actually gives a range from 10 to 17 hex.)

Note Don't care values are not allowed in expressions for the **echo** command.

Labels refer to names equated to numbers, patterns, or other expressions using the **equ** command.

Operators



The expression capability includes a powerful set of operators, freeing you from the need to calculate 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).

The operators are listed in the diagram above 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: **m -dm 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 = 1000000000000000000000000000001x0Y

0xxf0abcdH>>>4 = 0dxxf0abcH

& This symbol (&) represents a bit-wise AND operation. The truth table is:

&	**0**	**1**	**X**
0	0	0	0
1	0	1	X
X	0	X	X

For example:

10xxy&11x1Y = 10xxY

^ This symbol (^) represents a bit-wise exclusive OR operation. The truth table is:

^	**0**	**1**	**X**
0	0	1	0
1	1	0	X
X	0	X	X

For example:

10xxY^11x1Y = 01xxY

| This symbol (|) represents a bit-wise inclusive OR operation. The truth table is:

|	**0**	**1**	**X**
0	0	1	0
1	1	1	1
X	0	1	X

For example:

10xxY|11x1Y = 11x1Y

&& This symbol (&&) represents a bit-wise merge operation. The truth table resembles:

&&	**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.

For example:

10xxY&&11x1Y = 10x1Y

Using Expressions in Addressing and in 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 four address bytes, you can specify this as:

tg addr=(080h+(13T*4))

You could simplify the above even further using the equate command to assign names to some of the values. For example:

equ trapvectorbase=080h
equ trapvectorlength=4

Then:

tg addr=(trapvectorbase+(13*trapvectorlength))

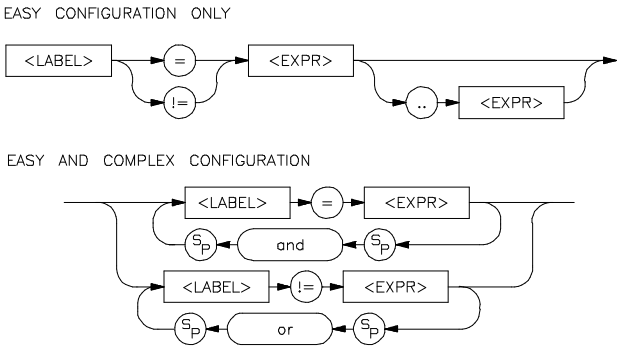
See also

See the <ANALYZER_EXPR>, <SIMPLE_EXPR>, and <COMPLEX_EXPR> pages for information on the use of expressions in forming analyzer expressions.

See the **echo** and **equ** command syntax pages for information on use of expressions in expression calculation and equates.

See the <ADDRESS> syntax pages for information on use of expressions in addressing.

SIMPLE_EXPR



Easy Configuration

When the analyzer is in easy configuration (**tcf -e**), simple expressions are used to set up trace qualifiers for sequencer branches, triggers, state counting, and so on. These expressions can take the following forms:

label=expression

Examples addr=2000h

data=25h+20h

stat=0110xxxxY

label!=expression

Examples stat!=suprdata (notice that the expression can also be an equate label)

data!=00

label=expression..expression

Examples addr=4000..401

data=41..42

label!=expression..expression

Examples addr!=1000..1038

data!=00..40

No more than one simple expression can exist at any given time which is in the form of a range (expr..expr).

label=expression and label=expression

Examples addr=3000 and data=41

addr=start and data=00

label!=expression or label!=expression

Examples addr!=3000 or data!=41

Complex Configuration

In analyzer complex configuration (**tcf -c**), you assign each simple expression a pattern name using the **tpat** command. These pattern names are then combined to form complex expressions involving relationships between multiple simple expressions.

With the exception of these two expressions:

label=expression..expression

label!=expression..expression

all of the simple expression types can be assigned pattern names by **tpat** in complex configuration. To form ranges of expressions in complex configuration, you use the **trng** command.

Examples tpat p1 addr!=3000 or data!=41

tpat p2 data=23

trng addr=1000..1038

(You don't need the **!=** relation in ranges because all complex expressions provide for the logical **not** of the range specifier.)

Invalid Simple Expressions

The following simple expressions are invalid in either analyzer configuration. If you need expressions of these types, you must switch to complex configuration, assign pattern names to subparts of these expressions, and then combine them using the complex expression capability.

label=expression and label!=expression

This is incorrect because you must use only the **=** relation with the **and** operator. To represent this, switch to complex configuration and do the following:

```
tpat p1 label=expression

tpat p5 label!=expression
```

Now, you would represent the above (incorrect) simple expression as a complex expression of the form:

```
p1 and p5
```

label!=expression or label=expression

A similar problem exists here. You must use only the **!=** relation with the **or** operator. To represent this, switch to complex configuration and do one of the following.

```
tpat p1 label!=expression

tpat p2 label=expression
```

You would represent the above (incorrect) simple expression as a complex expression of the form:

```
p1 | p2
```

You could also do this:

```
tpat p1 label!=expression

tpat p5 label=expression
```

Represent this in complex form as:

p1 or p5

See the <COMPLEX_EXPR> syntax pages for more details on forming complex expressions.

See also

See the <EXPR> syntax pages for information on numeric expression specifications.

12

Emulator Error Messages

The following messages may be seen when using the HP Emulators that support MC68020/MC68EC020 and MC68030/MC68EC030.

This chapter contains descriptions of error and status messages that can occur while using the Terminal Interface. The error messages are listed in numerical order, and each description includes the cause and the action you should take to remedy the situation.

The HP 64700-Series emulators can return messages to the display only when they are 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.

The emulator can return synchronous and/or asynchronous messages after executing commands. Synchronous messages are the result of the command being executed. Asynchronous messages are the result of some command executed previously (ie: software breakpoint hit).

A maximum of eight error messages can be displayed at one time. If more than eight errors are generated, only the last eight are displayed.

Emulator error messages

1 **!ERROR 1! I/O port access not supported**

Cause: You used the **io** command. The MC68020 and MC68030/MC68EC030 processors do not support separate I/O address space.

Action: Use the **m** command to modify memory mapped I/O ports on these emulators.

21 **!ERROR 21! Insufficient emulation memory**

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.

40 **!ERROR 40! Restricted to real time runs**

Cause: The **cf rrt=en** option is set (restrict to real time runs) and you have entered a command that requires a temporary break to the monitor for processing (such as a request to display target system memory locations).

Action: Break to the monitor using the **b** command, and then execute the desired command or disable real time mode with **cf rrt=dis**.

61 **!ERROR 61! Emulator is in the reset state**

Cause: This message is displayed if you request an operation that requires entry into the emulation monitor, such as display of target system memory locations while the **cf rrt=en** option is set (restrict to real time runs).

Action: If the prompt is R>, indicating an emulation system reset, break to the monitor using the **b** command, and then retry the command. Otherwise, disable the real time mode with **cf rrt=dis**.

80 **!ERROR 80! Stack pointer is odd**

Cause: You tried to modify the stack pointer to an odd value for a processor that expects the stack to be aligned on a word boundary.

Action: Modify the stack pointer to an even value.

84 **!ERROR 84! Program counter is odd**

Cause: You tried to modify the program counter to an odd value using the **reg** command. The processor expects even alignment of opcodes.

Action: Modify the program counter only to even numbered values.

84 **!ERROR 84! Program counter is odd or uninitialized**

Cause: You tried to run the processor from the current PC, but the PC was odd.

Action: Modify the PC to an even value. See Error message 84.

140 **!ERROR 140! Invalid attribute for memory type : <attribute>**

Cause: The memory type attributes **dp** and **dsi** are valid only for emulation memory (**eram** or **erom** memory types). You tried to assign one of these attributes to target memory (**tram** or **trom**).

Action: See Chapter 3 for information on the memory type attributes.

141 **!ERROR 141! Dual ported memory limited to 4K bytes**

Cause: There are only 4 Kbytes of dual-port emulation memory on the emulator probe. You tried to map an emulation memory term whose address range spanned more than 4 Kbytes by using the **dp** attribute.

Action: You can:

- Reenter the **map** command, using the **dp** attribute. Be sure to restrict the address range to 4 Kbytes (0..fff).
- Reenter the **map** command, and use regular emulation memory. That is, omit the **dp** attribute.

142 **!ERROR 142! Dual ported memory already in use**

Cause: There is only one 4-Kbyte block of dual-port, emulation memory available for mapping. You mapped this term and tried to map another using the **dp** attribute.

Action: Decide which block of memory should be in dual-port emulation memory, and define that block using the **map** command with the **dp** attribute. Use the **map** command without the **dp** attribute to define the other memory block.

143 **!ERROR 143! Dual ported memory in use by foreground monitor**

There is only one 4-Kbyte block of dual-port, emulation memory available for your use. If you select the foreground monitor (**cf mon=fg**), this block is used by the monitor and isn't available for mapping.

Action: Reenter the map command without the **dp** attribute, or select a background monitor and reenter the map command with the **dp** attribute.

144 **!ERROR 144! Dual ported memory not mapped to <range> for downloaded monitor**
!ERROR 144! Continuing with default foreground monitor

Cause: You tried to load a custom foreground monitor, but the load failed because the addresses in the absolute file didn't match the address range reserved for the monitor. The emulator aborted the load and reverted to the default foreground monitor. The **<range>** parameter shows the range of addresses covered by the custom monitor definition.

Action: To use a custom foreground monitor, you must:

- Select the foreground monitor: **cf mon=fg**
- Set the foreground monitor base address:
  cf monaddr=<ADDRESS>
- Load an absolute file that fits in the 4-Kbyte range that starts with **<ADDRESS>** using the **load -f** command.

145 **!ERROR 145! Downloaded monitor spans multiple 4K byte block boundaries**

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.

146 **!ERROR 146! Monitor must be mapped on a 4K byte boundary**

Cause: You tried to change the base address of the monitor to one that doesn't start on a 4-Kbyte boundary (address ending in 000).

Action: Reenter the **cf monaddr** command and specify an address that ends in 000.

150 **!ERROR 150! Program counter is located in guarded memory**

Cause: You tried to run the processor from the current PC, but the PC pointed to guarded memory.

Action: Modify the PC to an even value that points to a valid program memory area.

151 **!ERROR 151! <STACK> stack pointer is odd or uninitialized**

Cause: You tried to run from some address or perform a monitor operation that required the stack named **<STACK>**, but the stack was set to an odd value, which is invalid.

Action: Use the **reg** command to set the stack pointer to an even value that points at a memory region that can be used for stack operations. For the ISP (interrupt stack pointer), you can use the **cf rv** command to preset the initial value when the emulator enters the monitor from reset. (The ISP must point to a memory region that may be used as an interrupt stack.) Or, check your program to make sure it doesn't modify the ISP to an odd value. If the M bit in the status register is set, the processor uses the master stack instead of the interrupt stack. The master stack must also have an even value.

151 **!ERROR 151! <STACK> stack is located in guarded memory: <address>**

Cause: You tried to run from some address or perform a monitor operation that required the stack named **<STACK>**, but the stack pointer points to guarded memory, which is invalid.

Action: Use the **reg** command to set the stack pointer to an even value that points at a memory region that can be used for stack operations. For the ISP (interrupt stack pointer), you can use the **cf rv** command to preset the initial value when the emulator enters the monitor from reset. (The ISP must point to a memory region that may be used as an interrupt stack.) Or, check your program to make sure that it doesn't modify the ISP to an odd value. If the M bit in the status register is set, the processor uses the master stack instead of the interrupt stack. The master stack must also have an even value.

151 **!ERROR 151! <STACK> stack is located in ROM: <address>**

Cause: You tried to run from some address or perform a monitor operation that required the stack named **<STACK>**, but the stack pointer points to emulation or target ROM, which is invalid.

Action: Use the **reg** command to set the stack pointer to an even value that points at a memory region that can be used for stack operations. For the ISP (interrupt stack pointer), you can use the **cf rv** command to preset the initial value when the emulator enters the monitor from reset. (The ISP must point to a memory region that may be used as an interrupt stack.) Or, check your program to make sure that it doesn't modify the ISP to an odd value. If the M bit in the status register is set, the processor uses the master stack instead of the interrupt stack. The master stack must also have an even value.

152 **!ERROR 152! Invalid number of bytes for word access**

Cause: The access mode was set to word (**mo -aw**), but you supplied an odd number of bytes for a memory modify.

Action: Either change the access mode to byte (**mo -ab**) or supply an even number of bytes. (Note: this message will not appear for memory mapped with the **dp** attribute.)

152 **!ERROR 152! Invalid number of bytes for longword access**

Cause: The access mode was set to long word (**mo -al**), but the number of bytes you supplied for a memory modify was not a multiple of four.

Action: Either change the access mode to byte or word (**mo -ab** or **mo -aw**), or supply a number of bytes that is a multiple of four. (Note: this message will not appear for memory mapped with the **dp** attribute.)

152 **!ERROR 152! Invalid number of words for longword access**

The access mode was set to long word (**mo -al**), but the number of bytes you supplied for a memory modify was not a multiple of four.

Action: Either change the access mode to byte or word (**mo -ab** or **mo -aw**), or supply a number of bytes that is a multiple of four. (Note: this message will not appear for memory mapped with the **dp** attribute.)

155 **!STATUS 155! Trace vector modified to <address> for single stepping**

Cause: You used the **s** (step) command. The emulator modified the trace vector to <address> so that the single step function would operate correctly.

Action: See “To step the processor” in Chapter 4 for more information.

156 **!ERROR 156! Unable to verify trace vector; vector table in guarded memory**

Cause: You used the **s** (step) command. The emulator tried to set the trace vector to the correct value for stepping and failed due to a guarded memory access.

Action: Check the VBR register to make sure the vector table is where you expect it to be. You need to remap memory so that the vector table will be located in a valid memory area. See “To step the processor” in Chapter 4 for more information.

156 **!ERROR 156! Unable to verify trace vector; vector table read failed**

Cause: You used the **s** (step) command. The emulator tried to set the trace vector to the correct value for stepping. It failed because the vector table region was mapped to target RAM or ROM, and (most likely) no physical memory was located at that address.

Action: Make sure you initialized the vector correctly in your vector table. You need to remap memory so that the vector table will be located in a valid memory area. See “To step the processor” in Chapter 4 for more information.

156 **!ERROR 156! Unable to set trace vector to <address>; vector table write failed**

Cause: You used the **s** (step) command. The emulator tried to set the trace vector to the correct value for stepping, but failed. The most likely cause is that the vector table is mapped to target RAM (**tram**) but the physical memory is target ROM or there is no memory at that address.

Action: Make sure you initialized the vector correctly in your vector table. You need to remap memory so that the vector table will be located in a valid memory area. See “To step the processor” in Chapter 4 for more information.

156 **!ERROR 156! Unable to set trace vector to <address>; vector table in TROM**

Cause: You used the **s** (step) command. The emulator can't modify the trace vector to the correct value for stepping because the vector table region is mapped to target ROM.

Action: Make sure you initialized the vector correctly in your vector table. You need to remap memory so that the vector table will be located in a valid memory area. See "To step the processor" in Chapter 4 for more information.

156 **!ERROR 156! Trace vector points to unreadable memory**

Cause: You used the **s** (step) command. The emulator found that the trace vector was even and in a mapped memory range; however, the processor entered an indefinite wait state when trying to read the location pointed to by the vector. This may be caused by a memory system failure, either in the target system or the emulator.

Action: Make sure the VBR register table is where you expect it to be. See "To step the processor" in Chapter 4 for more information.

157 **!STATUS 157! Monitor type changed to foreground monitor**

Cause: You enabled the MMU via a configuration command while the background monitor was selected. The emulator automatically changed to use the default foreground monitor when you enabled the MMU. Only a foreground monitor can be used when the MMU is enabled.

158 **!ERROR 158! Background monitor unavailable while MMU is enabled**

Cause: You attempted to switch to the background monitor while the MMU is enabled.

Action: Use only a foreground monitor when the MMU is enabled.

160 **!ERROR 160! MMU not enabled via configuration**

Cause: You tried to use the **mmu** or **dmmu** command to display MMU translations or load the deMMUer, but the MMU has not been enabled with **cf mmu**.

Action: Enable the MMU before trying to use these commands.

160 **!STATUS 160! MMU not enabled via TC register**

Cause: You used the **mmu** or **dmmu** command to display MMU translations or load the deMMUer, but the MMU has not been enabled in software by setting the enable bit in the translation control register. The MMU does not have to be enabled in software in order to use these commands and this status is only used to warn you that the MMU is not actually enabled yet.

161 **!STATUS 161! Address(es) transparently translated via TT0/TT1**

Cause: You used the **mmu** command to display MMU address translations and one or more addresses are also transparently translated by the processor TT0 and/or TT1 registers. Other than this status message, the **mmu** command completely ignores transparent translations when looking up logical-to-physical address translations. This status is only used to warn you that the MMU is currently set up to transparently translate one or more addresses 1:1 instead of using the displayed address translations.

162 **!ERROR 162! MMU configuration error - invalid <register> value**

Cause: You tried to use the **mmu** or **dmmu** command to display MMU translations or load the deMMUer, but one of MMU registers (TC, CRP or SRP) is invalid. You will also see this message if invalid alternate register values are specified on the command line for the **mmu** command.

Action: To solve this problem, use the **reg** command to initialize the MMU registers to valid values or use the **mmu <reg>=<value>** command to display MMU translations using alternate register values.

162 **!ERROR 162! No translation for CPU space addresses**

Cause: You tried to use the **mmu** command to display MMU translations, but the address you specified indicated CPU space address (ie: 123@cpu). By definition, there are no translations performed by the MMU on CPU space addresses and the logical address is the physical address.

Action: Try some other function code space.

163 **!STATUS 163! Out of DeMMUer resources**

Cause: You used the **dmmu** command to load the deMMUer but there was not enough resouces to reverse translate all address translations defined by your translation tables. Depending upon the MMU page size selected, the deMMUer can reverse translate 16M or 256M physical addresses into logical addresses.

Action: See the deMMUer chapter for more information on how to make the most efficient use of the deMMUer resources.

164 **!STATUS 164! Transparent addresses denoted as physical in analysis trace**

Cause: You used the **dmmu** command to load the deMMUer and one or more address ranges are transparently translated via TT0/TT1 registers. The deMMUer will reverse translate your transparently translated addresses 1:1; however, these addresses will be tagged as physical instead of logical. Unfortunately for these addresses, symbolic lookup is not performed on addresses tagged as physical so you will not be able to obtain symbolic support for these addresses.

If you have specified a read or write qualification for TT0 or TT1, the deMMUer will not provide the correct translation for addresses that do not meet the read/write qualification. Because the deMMUer cannot distinguish between read and write cycles, the deMMUer always interprets the TT0/TT1 registers as if the read/write mask bit is set to ignore the read/write line (RWM=1).

164 **!ERROR 164! DeMMUer has not been loaded**

Cause: You used the **dmmu -e** command to enable the deMMUer before it had been loaded.

Action: Use the **dmmu -le** to load and enable the deMMUer.

164 **!ERROR 164! Unable to access the deMMUer while analysis trace is in process**

Cause: You used the **dmmu** command while an analysis trace was running. Because the **dmmu** command will adversely affect the analysis data path, it can only be used when an analysis trace is not in process.

Action: Let the current analysis trace complete or use the **th** command to halt the analysis trace. Then retry the operation.

170 **!ERROR 170! Monitor failure; bus grant**

Cause: During a monitor command, 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.

170 **!ERROR 170! Monitor failure; CPU in wait state**

Cause: Request to access memory failed because the target system did not provide cycle termination for this address; the processor is in a continuous wait state. A continuous wait state may indicate target system problems.

Action: If this occurred during an access to emulation memory, then the **dsi** memory attribute must have been used. If the target is not supposed to provide cycle terminations for this range, delete and reenter the map term without **dsi**.

170 **!ERROR 170! Monitor failure; no target power**

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.

170 **!ERROR 170! Monitor failure; slow clock**

Cause: The target system is providing target power but no clock signal.

Action: Make sure the clock oscillator is installed correctly.

170 **!ERROR 170! Monitor failure; target reset**

Cause: During a monitor command, the target system ascerted (and continues to ascert) the reset signal; the monitor is no longer responding.

Action: Prevent your target system from ascerting the reset signal when you are using monitor commands.

170 **!ERROR 170! Monitor failure; halted**
170 **!ERROR 170! Monitor failure; no bus cycles**

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.

170 **!ERROR 170! Request failed; bus error**

Cause: The monitor caught a bus error exception while attempting to display or modify memory. This can occur if the target system terminates a memory access with a bus error.

Action: See if this bus error occurred in emulation memory; if it did, the **dsi** memory attribute must have been used. If the target is not supposed to provide cycle terminations for this emulation memory range, delete and reenter this map term without the **dsi** attribute.

The 68030 MMU can also terminate an access with a bus error if there is no valid address transalation. Use the **mmu** command to verify that the address translation tables are set up correctly.

170 **!ERROR 170! Request failed; level 7 interrupt**

Cause: The monitor caught a non-maskable level 7 interrupt (autovector) which was generated by the target system while the emulator was executing a monitor command. During monitor commands, the monitor uses its own exception vector table and attempts to block all interrupts. However, a non-maskable level 7 interrupt causes the monitor to abort the command, and restore the original vector table and interrupt priority mask. The target system level 7 interrupt handler is not executed.

170 **!ERROR 170! Request failed; no MC68881/882 floating point coprocessor**

Cause: You entered a command to display or modify an FPU register, but the processor generated an exception indicating there is no coprocessor in your system.

170 **!ERROR 170! Request failed; unexpected exception <vector>**

Cause: The monitor detected an exception that was generated during execution of a monitor command.

175 **!ERROR 175! Coverage not supported**

Cause: The **cov** (memory coverage) command doesn't exist in the MC68020/MC68EC020 and MC68030/MC68EC030 emulators because there is no supporting hardware.

175 **!ERROR 175! Copy target image not supported**

Cause: The **cim** (copy image memory) command doesn't exist in the MC68020/MC68EC020 and MC68030/MC68EC030 emulators. Normally, this 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 **dump** command. Then remap the target memory range to emulation memory, and load the absolute file into emulation memory using the **load** command. See Chapter 4 for information on saving and loading absolute files.

176 **!ERROR 176! Update HP64700 firmware to version A.03.01 or newer**

Cause: The emulator has found that one or more modules of your system firmware is too old to be compatible with the emulation firmware.

Action: Update the system firmware to the lastest release.

176 **!ERROR 176! Update HP64740 firmware to version A.02.02 or newer**

Cause: The emulator has found that your analysis firmware is too old to be compatible with the emulation firmware.

Action: Update the analysis firmware to the lastest release.

178 **!ERROR 178! Unable to run HP64747 tests without target power**
178 **!ERROR 178! Unable to run HP64748 tests without target power**

Cause: The demo board does not have proper power connected to it.

Action: Check the connection of the external power cable to the demo board.

178 **!ERROR 178! Unable to run HP 64747 performance verification tests**
178 **!ERROR 178! Unable to run HP 64748 performance verification tests**

Cause: You entered the **pv** command, but the emulator was unable to start performance verification.

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 and that the switches on the demo board are set to the test position. Also verify that the demo board power cable is connected to the HP 64700 Card Cage.

179 **!ERROR 179! HP 64748 M68020 probe not connected**
179 **!ERROR 179! HP 64747 M68030/EC030 probe not connected**

Cause: The emulator is reading an invalid identifier for the emulation probe.

Action: Make sure that the probe cables are connected correctly. Also make sure that the probe is the MC68020/MC68EC020, or the MC68030/MC68EC030 probe, as applicable. Make sure correct firmware is flashed on the control card (020 firmware for the 020 probe or 030 firmware for the 030 probe).

201 **!ERROR 201! Out of system memory**

Cause: Macros and equates that you have defined have used all of the available system memory.

Action: Delete some of the existing macros (**mac -d <NAME>**) and equates (**equ -d <NAME>**), which will free additional memory.

204 **!ERROR 204! FATAL SYSTEM SOFTWARE ERROR**
205 **!ERROR 205! FATAL SYSTEM SOFTWARE ERROR**
208 **!ERROR 208! FATAL SYSTEM SOFTWARE ERROR**

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.

300 **!ERROR 300! Invalid option or operand**
305 **!ERROR 305! Invalid option or operand: <option>**

Cause: You have specified incorrect option to a command. <option>, if printed, indicates the incorrect option.

Action: Use online help by typing **help <command>** or **? <command>**. Reenter the command with the correct syntax. See Chapter 10 for more information.

307 **!ERROR 307! Invalid expression: <expression>**

Cause: You have entered an expression with incorrect syntax; therefore, it cannot be evaluated. <expression> is the bad expression.

Action: Use online help by typing **help gram**. Reenter the expression, following the syntax rules for that type of expression. See Chapter 10 to determine the expression type; then see Chapter 11 to determine the correct syntax for that type.

308 **!ERROR 308! Invalid number of arguments**

Cause: You either entered too many options to a command or an insufficient number of options.

Action: Reenter the command with correct syntax. Use online help by typing **help <command>**. See Chapter 10 in this manual for information.

310 **!ERROR 310! Invalid address: <address>**

You specified an invalid address value as an argument to a command (other than an analyzer command). For example, you may have specified digits that don't correspond to the base specified, or you forgot to precede a hexadecimal letter digit with a number (even zero (0)).

Action: Reenter the command and the address specification. Use online help by typing **help proc**. See the <ADDRESS> and the <EXPRESSION> syntax pages in this manual for information on address specifications.

311 **!ERROR 311! Invalid address range: <address_range>**

Cause: You specified an invalid address range as an argument to a command (other than an analyzer command). For example, you may have specified digits that don't correspond to the base specified, or you forgot to precede a hexadecimal letter digit with a number, or the upper boundary of the range you specified is less than the lower boundary.

Action: Reenter the command and the address specification. Use online help by typing **help proc**. See the <ADDRESS> syntax pages 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).

312 **!ERROR 312! Ambiguous address: <address>**

Cause: You mapped memory using function codes, but didn't enter a function code in an address specification for a run or trace measurement. The emulator is unable to determine which of two or more address ranges you are referring to, based on the information you entered.

Action: Reenter the command and fully specify the address, including function code information.

313 **!ERROR 313! Missing option or operand**

Cause: You have omitted a required option to the command.

Action: Reenter the command with the correct syntax. Use online help by typing **help <cmd>**. See Chapter 10 in this manual for further information on required syntax.

314 **!ERROR 314! Option conflict: <option>**

Cause: You have entered a command with two options which cannot be used together. For example, you might have entered **tl -bx**; you cannot ask for both a binary and hexadecimal trace list dump.

Action: Reenter the command, specifying only non-conflicting options. See the syntax information for the command in Chapter 10 of this manual to determine which options may be used together.

315 **!ERROR 315! Invalid count: <count>**

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.

316 **!ERROR 316! Invalid range expression: <range>**

Cause: In the **tl** command, you specified an illegal range. For example, you might have specified **tl -10..a**.

Action: Use only legitimate range numbers in the **tl** command (-1024..1023 with counting off, or -512..511 with counting on); the second range value must be greater than the first.

317 **!ERROR 317! Range out of bounds: <address range>**

Cause: In the **tl** command, you specified a range number that was greater than the number of states available in the analyzer. For example, you might have specified **tl -2048..2048**; the analyzer only has 1024 states.

Action: Specify range numbers between -1024 and 1023 when counting is turned off, or between -512 and 511 when counting is turned on.

318 **!ERROR 318! Count out of bounds: <number>**

Cause: You specified an occurrence count less than 1 or greater than 65535 for **tg** or **tif**. For example, you might have entered **tif 1 any 2 69234**.

Action: Reenter the command, specifying a count value from 1 to 65535. For example: **tif 1 any 2 65535**.

319 **!ERROR 319! Invalid base: <base>**

Cause: This error occurs if you have specified an invalid base in the **tf** command.

Action: Enter the **help tf** or command to view the valid base options.

320 **!ERROR 320! Invalid label: <label>**

Cause: You tried to define a label with characters other than letters, digits, or underscores.

Action: Reenter the **tlb** command with a label consisting only of letters, digits, or underscores.

321 **!ERROR 321! Label not defined: <label>**

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 **tg lowerdata=24t**. This error also occurs if you try to delete a label that does not exist.

Action: You can reenter the command, using one of the previously defined labels, and adjust the expression as necessary to accommodate the fit of that label to the analyzer input lines. You can also define a new label using the **tlb** command, and then reenter the analyzer command using the newly defined label.

400 **!ERROR 400! Record checksum failure**

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.

401 **!ERROR 401! Records expected: <number>; records received: <number>**

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.

410 **!ERROR 410! File transfer aborted**

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.

411 **!ERROR 411! Severe error detected, file transfer failed**

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.

412 **!ERROR 412! Retry limit exceeded, transfer failed**

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.

520 **!ERROR 520! Equate not defined: <name>**

Cause: You tried to delete an equate that did not exist in the equate table. For example suppose the equates **a=1** and **b=2** were in the equate table. If you typed **equ -d c**, you would receive the above error message.

Action: Use **equ** to display the list of named equates before deleting equates.

603 **!ERROR 603! Read PC failed during break**

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.

604 **!ERROR 604! Disable breakpoint failed: <address>**

Cause: System failure or target condition.

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

605 **!ERROR 605! Undefined software breakpoint: <address>**

Cause: The emulator has encountered a BKPT instruction in your program that was not inserted with the **bp** command.

Action: You can choose one of BKPT 1 through BKPT 7 for the software breakpoint instruction, using the **cf sw** command. Otherwise, remove the breakpoints inserted in your code before assembly and link, and then reinsert them using the **bp** command. If this message was received after you enabled the MMU of the MC68030, read "Undefined software breakpoint in Chapter 9.

606 **!ERROR 606! Unable to run after CMB break**

Cause: System failure or target condition.

Action: Run performance verification (**pv** command), and check target system.

608 **!ERROR 608! Unable to break**

Cause: This message is displayed if the emulator is unable to break to the monitor because the emulation processor is reset, halted, or is otherwise disabled.

Action: First, look at the emulation prompt and other status messages displayed to determine why the processor is stopped. If reset by the emulation controller, use the **b** command to break to the monitor. If reset by the emulation system, release that reset. If halted, try **rst -m** 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.

If this message appears after you have turned on the MMU in an MC68030 emulator, the MMU may have relocated your breakpoint to an address unknown to the breakpoint table. Refer to Chapter 9 for more information.

610 **!ERROR 610! Unable to run**

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.

611 **!ERROR 611! Break caused by CMB not ready**

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).

613 **!ASYNC_STAT 613! Analyzer Break**

Cause: Status message. No action necessary.

615 **!ERROR 615! Software breakpoint: <address>**

Cause: This status message is displayed when a software breakpoint entered with **bp** and enabled with **bc -e bp** is encountered during a program run. The emulator breaks to the monitor. <address> shows the address where the breakpoint was encountered.

616 **!ASYNC_STAT 616! BNC trigger break**

Cause: This status message will be displayed if you have set **bc -e bnct** and the BNC trigger line is activated during a program run. The emulator is broken to the monitor.

617 **!ASYNC_STAT 617! CMB trigger break**

Cause: This status message will be displayed if you have set **bc -e cmbt** and the CMB trigger line is activated during a program run. The emulator is broken to the monitor.

618 **!ASYNC_STAT 618! trig1 break**

Cause: This status message will be displayed if you have set the analyzer to drive **trig1** upon finding the trigger, **bc -e trig1** is set, and the analyzer has found the trigger condition while tracing a program run. The emulator is broken to the monitor.

619 **!ASYNC_STAT 619! trig2 break**

This status message will be displayed if you have set the analyzer to drive **trig2** upon finding the trigger, **bc -e trig2** is set, and the analyzer has found the trigger condition while tracing a program run. The emulator is broken to the monitor.

620 **!ERROR 620! Unexpected software breakpoint**
621 **!ERROR 621! Unexpected step break**

Cause: System failure.

Action: Run performance verification (**pv** command).

623 **!ASYNC_STAT 623! CMB execute break**

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.

624 **!ERROR 624! Configuration aborted**

Cause: Occurs when a <CTRL> c is entered during **cf** display command.

625 **!ERROR 625! Invalid configuration value: <value>**

Cause:You have entered a configuration option incorrectly, such as typing **cf sw=44** instead of **cf sw=4**.

Action: Type **help cf <item>** for a description of configuration items and valid values. Reenter the configuration command, specifying only the correct values.

626 **!ERROR 626! Configuration failed; setting unknown: <item>=<value>**

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

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

627 **!ERROR 627! Invalid configuration item: <item>**

Cause: You specified a non-existent configuration item in the **cf** command. For example, because the MC68020/MC68EC020 and MC68030/MC68EC030 emulators don't support an internal clock, you would see this message if you entered **cf clk=int** because there is no **clk** configuration item for your emulator.

Action: Type **help cf** to see valid items. Reenter the command, specifying only configuration items that are supported by your emulator. Refer to the **cf** syntax pages in Chapter 10.

628 **!ASYNC_STAT 628! Write to ROM break: <address>**

Cause: You set **bc -e rom** and your program attempted to write to a memory location mapped as **trom** or **erom**. The <address> parameter indicates the address and function code at which the write occurred. The emulator breaks into the monitor.

628 **!ASYNC_STAT 628! Guarded mem break: <address>**

Cause: Your program attempted to read or write a memory location mapped as guarded (**grd**). The <address> parameter indicates the address and function code where the bus cycle occurred. The emulator breaks into the monitor.

628 **!ASYNC_STAT 628! Monitor handled target exception; <exception>**

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.

630 **!ERROR 630! Register access aborted**

Cause: Occurs when a <CTRL> c is entered during register display.

631 **!ERROR 631! Unable to read registers in class: <name>**

Cause: The emulator was unable to read the registers you requested.

Action: To resolve this, you must look at the other status messages displayed. Most likely, the emulator was unable to break to the monitor to perform the register read. See message 608.

632 **!ERROR 632! Unable to modify register: <register>=<value>**

Cause: The emulator was unable to modify the register you requested.

Action: To resolve this, you must look at the other status messages displayed. It's likely that the emulator was unable to break to the monitor to perform the register modification. See message 608.

634 **!ERROR 634! Display register failed: <register>**

Cause: The emulator was unable to display the register you requested.

Action: To resolve this, you must look at the other status messages displayed. It's likely that the emulator was unable to break to the monitor to perform the register display. See message 608.

637 **!ERROR 637! Register class cannot be modified: <register class>**

Cause: You tried to modify a register class instead of an individual register. You can only modify individual registers.

Action: See the **reg** syntax pages in Chapter 10 of this manual for a list of register names.

640 **!ERROR 640! Unable to reset**

Cause: Target condition or system failure.

Action: Check target system, and run performance verification (**pv** command).

641 **!ERROR 641! Unable to reset into monitor**

Cause: You have entered a **rst -m** command and the emulator is unable to break into the monitor.

Action: Reload the monitor (**rst**). (If you are using a custom foreground monitor, use the **load -f** command to reload the monitor).

652 **!ERROR 652! Break condition must be specified**

Cause: You entered **bc -e** or **bc -d** without specifying a break condition to enable or disable.

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

653 **!ERROR 653! Break condition configuration aborted**

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

661 **!ERROR 661! Software breakpoint break condition is disabled**

Cause: You entered the **bp** command and options; however, the software breakpoint break condition is disabled.

Action: Enable the software breakpoint feature with **bc -e bp**. Then enter the desired breakpoints with **bp**.

663 **!ERROR 663! Specified breakpoint not in list: <address>**

You tried to enable a software breakpoint (**bp -e <ADDRESS>**) that was not previously defined. <address> prints the address of the breakpoint you attempted to enable. Insert the breakpoint into the table and memory by typing **bp <ADDRESS>**.

664 **!ERROR 664! Breakpoint list full; not added: <address>**

Cause: The software breakpoint table is full. The breakpoint you just requested, with address <address>, was not inserted.

Action: Remove breakpoints that are no longer in use with **bp -r <ADDRESS>**. Then insert the new breakpoint.

665 **!ERROR 665! Enable breakpoint failed: <address>**

Cause: System failure or target condition.

Action: Check memory mapping and configuration questions.

666 **!ERROR 666! Disable breakpoint failed: <address>**

Cause: System failure or target condition.

Action: Check memory mapping and configuration questions.

667 **!ERROR 667! Breakpoint code already exists: <address>**

You attempted to insert a breakpoint with **bp <ADDRESS>**; however, there was already a software breakpoint instruction at that location which was not already in the breakpoint table. Your program code is apparently using the same breakpoint instruction as **bp**. If multiple breakpoint instructions are available on your processor, either change those in your program code or modify the one **bp** uses with your emulator's configuration options (**cf** command). If only one instruction is available, remove the breakpoints from your program code and use **bp** to insert breakpoints.

668 **!ERROR 668! Breakpoint not added: <address>**

You tried to insert a breakpoint in a memory location which was not mapped or was mapped as guarded memory. Insert breakpoints only within memory ranges mapped to emulation or target RAM or ROM.

669 **!ERROR 669! Breakpoint remove aborted**

Cause: Occurs when <CTRL> c is entered during a **bp -r** command.

670 **!ERROR 670! Breakpoint enable aborted**

Cause: Occurs when <CTRL> c is entered during a **bp -e** command.

671 **!ERROR 671! Breakpoint disable aborted**

Cause: Occurs when <CTRL> c is entered during a **bp -d** command.

680 **!ERROR 680! Stepping failed**

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. (See error message 151.)

682 **!ERROR 682! Invalid step count: <count>**

Cause: You specified an non-cardinal value for a step count in the **s** command (such as entering **s 22.1**).

Action: Reenter the step command, using only cardinal values (positive integers) for the step count.

684 **!ERROR 684! Failed to disable step mode**

Cause: System failure. Run performance verification (**pv** command).

685 **!ERROR 685! Stepping aborted**

Cause: This message is displayed if a break was received during a **s** (step) command with a stepcount of zero (0). The break could have been due to any of the break conditions in **bc** or a <CTRL> c break.

686 **!ERROR 686! Stepping aborted; number steps completed: <steps completed>**

Cause: This message is displayed if a break was received during a **s** (step) command with a stepcount greater than zero. The break could have been due to any of the break conditions in **bc** or a <CTRL> c break. The number of steps completed is displayed.

688 **!ERROR 688! Step display failed**

Cause: System failure or target condition.

Action: Check memory mapping and configuration questions.

689 **!ERROR 689! Break due to cause other than step**

Cause: An activity other than a **step** command caused the emulator to break. This could include any of the break conditions in a **bc** command or a <CTRL> c break.

692 **!ERROR 692! Trace error during CMB execute**

Cause: System failure.

Action: Run performance verification (**pv** command).

693 **!ASYNC_STAT 693! CMB execute; run started**

Cause: This status message is displayed when you are making coordinated measurements. The CMB /EXECUTE pulse has been received; the emulation processor started running at the address specified by the **rx** command.

694 **!ASYNC_ERR 694! Run failed during CMB execute**

Cause: System failure or target condition.

Action: Run performance verification (**pv** command), and check target system.

700 **!ERROR 700! Target memory access failed**

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 608.

702 **!ERROR 702! Emulation memory access failed**

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 608.

707 **!ERROR 707! Request access to guarded memory: <address>**

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.

720 **!ERROR 720! Invalid map term number: <map term number>**

Cause: You attempted to delete a mapper term that does not exist. For example, you may have tried **map -d 8**, but the emulator only has seven map terms. You may have tried **map -d 2**, when only one mapper term has been defined.

Action: Use the **map** command to determine the numbers of the terms currently mapped. Then delete the appropriate mapper term.

721 **!ERROR 721! No map terms available; maximum number already defined**

Cause: You tried to add more mapper terms than are available for this emulator. For example, with the MC68020/MC68EC020 or MC68030/MC68EC030 emulator, there are only seven 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.

723 **!ERROR 723! Invalid map address range: <address range>**

Cause: You specified an invalid address range as an argument to the **map** 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 **map** command and the address specification. See the <ADDRESS> and the <EXPRESSION> syntax pages in this manual for information on address specifications. Also, make sure that the upper boundary specification is greater than the lower boundary specification (the lower boundary must always precede the upper boundary on the command line).

724 **!ERROR 724! Address not mappable: <address>**

Cause: You tried to map an address range using a non-mappable function code. For example, you may have entered map 1000..10ff@cpu.

Action: Enter your address using an acceptable function code. The only function codes that may be used in map specifications are x, sp, sd, up, up, s, u, p, and d.

725 **!ERROR 725! Unable to load new memory map; old map reloaded.**

Cause: There is not enough emulation memory left for this request.

Action: Reduce the amount of emulation memory requested.

726 **!ERROR 726! Unable to reload old memory map; hardware state unknown**

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.

730 **!ERROR 730! Invalid memory map type: <type>**

Cause: You specified a memory type while mapping that is not one of the supported types: **eram**, **erom**, **tram**, **trom**, or **grd**.

Action: Reenter the **map** command, specifying only one of the five supported types, listed above.

731 **!ERROR 731! Invalid memory map attribute: <attribute>**

Cause: The only valid memory map attributes for the MC68020/MC68EC020 emulator are **dp** (dual-port memory) and **dsi** (interlock DSACKs). The MC68030/MC68EC030 emulator supports these attributes and the **ci** (cache inhibit) attribute. For example, the following command will cause an error: “map 0..100 eram ds2”.

Action: Reenter your command, using only valid memory map attributes.

732 **!ERROR 732! Invalid memory type for ’other’ range: <type>**

Cause: The memory types for **map other <type>** are restricted to **tram**, **trom**, or **grd**. If you see the above message, you have tried to map the “other” range to **eram** or **erom**.

Action: Map the “other” range to **tram**, **trom**, or **grd**.

734 **!ERROR 734! Map range overlaps with term: <term number>**

Cause: You entered a map term whose address range overlaps with one already mapped. For example, you may have entered a term **map 1000..2fff eram**, and then tried to enter a term **map 2000..3fff erom**.

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 Chapter 11 of this manual.

752 **!ERROR 752! Copy memory aborted; next destination: <address>**
754 **!ERROR 754! Memory modify aborted; next address: <address>**
756 **!ERROR 756! Memory search aborted; next address: <address>**

Cause: One of these messages is displayed if a break occurs during processing of the **cp**, **m**, or **ser** commands, respectively. The break could result from any of the break conditions (except **bp**) 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 with the **bc** command.

800 **!ERROR 800! Invalid command: <command>**

Cause: You entered a command that is not part of the standard Terminal Interface command set (documented in this manual) and was not found in the currently defined macros.

Action: Enter only commands defined in this manual or in the macro set. You can display the macro set using **mac**. You can rename commands or name command groups using the **mac** command.

801 **!ERROR 801! Invalid command group: <group name>**

Cause: This error occurs when you specify an invalid group name in the **help -s <group>** command.

Action: Enter the **help** command with no options for a listing of the valid group names.

802 **!ERROR 802! Invalid command format**

Cause: This error occurs when an invalid macro is entered, for example, **mac {help;{}**.

Action: See the **mac** command description.

807 **!ERROR 807! Macro list full; macro not added**

Cause: The maximum number of macros have been defined.

Action: You must delete macros before adding any new macros.

809 **!ERROR 809! Macro buffer full; macro not added**

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.

812 **!ERROR 812! Invalid macro name: <name>**

Cause: You tried to delete a macro that did not exist; or you tried to define a new macro with a name containing characters other than letters, digits, or underscores.

Action: Use the **mac** command to display the names of macros in the macro table before deleting them with **mac -d <NAME>**. Define new macro names using only letters, digits, and underscore characters.

813 **!ERROR 813! Command line too long; maximum line length: <number of characters>**

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

Action: Split the command line into two command lines.

814 **!ERROR 814! Command line too complex**

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

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

815 **!ERROR 815! Missing macro parameter: <parameter>**

Cause: This error occurred because you did not include a parameter with the specified **mac** command for macro expansion.

Action: Enter the command again, and include the appropriate parameter for the macro expansion.

816 **!ERROR 816! Command line too complex**

Cause: Too many expression operators are used.

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

818 **!ERROR 818! Command line too complex**

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

Action: Reduce the number of nesting levels.

820 **!ERROR 820! Unmatched quote encountered**

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”**.

822 **!ERROR 822! Unmatched command group encountered**

Cause: You entered the **mac** or **rep** command group without matching braces **{}**. For example: **mac test={rst -m;cf** or **rep 2 {rst -m;map**.

Action: Reenter the command, making sure to match braces around commands you want grouped into the macro or repeat. For example: **mac test={rst -m;cf}**.

824 **!ERROR 824! Maximum number of arguments exceeded**

Cause: You exceeded the limit of 100 arguments per command.

Action: Reduce the number of arguments in the command.

826 **!ERROR 826! Maximum argument buffer space exceeded**

Cause: You exceeded the space limits for argument lists.

Action: Reenter the command with less arguments, or simplify the expressions in the arguments.

840 **!ERROR 840! Invalid date: <date>**

Cause: You specified the date format incorrectly in the **dt** command.

Action: Reenter the command with the correct date format. See the **dt** command syntax pages in this manual for the correct format.

842 **!ERROR 842! Invalid time: <time>**

Cause: You have incorrectly specified the time format in the **dt** command.

Action: Reenter the command with the correct time format. See the **dt** command syntax pages in this manual for the correct format.

844 **!ERROR 844! Invalid repeat count: <count>**

Cause: You entered a non-cardinal value for the repeat count in the **rep** command, such as **rep 22.1 <command_group>**

Action: Reenter the **rep** command, specifying only a cardinal number (positive integer) for the repeat count.

850 **!ERROR 850! Attempt to load code outside of allocated bounds**

Cause: This error occurs when the **lcd** command attempts to load an absolute file that contains code or data outside the range allocated for system code. Generally, you will not use the **lcd** command. The **lcd** command is intended to be used by high-level interfaces to the HP 64700.

875 **!ERROR 875! Invalid syntax for global or user symbol name: <symbol>**

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, **:glb_sym**). When specifying a user symbol (created with the **sym** command), make sure that you enter the name correctly without a colon.

876 **!ERROR 876! Invalid syntax for local symbol or module: <symbol/module>**

Cause: This error occurs when you enter a local symbol or module name with incorrect syntax.

Action: When entering a local symbol name using the **sym** command, make sure that you specify the module name, followed by a colon, and then the symbol name (for example **module:loc_sym**). Make sure that you specify the module name correctly.

877 **!ERROR 877! Symbol not found: <symbol>**

Cause: This occurs when you try to enter a symbol name that doesn't exist.

Action: Enter a valid symbol name.

878 **!ERROR 878! Symbol cannot contain wildcard in this context**

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.

879 **!ERROR 879! Symbol cannot contain text after the wildcard**

Cause: You tried to include text after the wildcard specified in the symbol name (for example, **sym*text**).

Action: Enter the symbol again, but don't include text after the wildcard (*).

880 **!ERROR 880! Conflict between expected and received symbol information**

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.

881 **!ERROR 881! Ascii symbol download failed**

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.

882 **!ERROR 882! No module specified for local symbol**

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.

Analyzer Error Messages

1000 **!ERROR 1000! Conflicting disassembler option: <option>**

Cause: This error occurs when you attempt to specify inverse assembly options (**tl -o<ialopts>**) which are not allowed with each other (for example, **tl -oai**).

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

1001 **!ERROR 1001! Invalid disassembler option: <option>**

Cause: The <**ialopts**> option specified with the "**tl -o**" command is not valid. The valid disassembler options are **a** (display all bus cycles), **i** (display only instruction cycles), **d** (dequeue the trace list), **n** (don't dequeue the trace list), and **l** (disassemble starting with the lower word of the instruction).

Action: Use valid inverse assembly options in your command.

1102 **!ERROR 1102! Invalid bit range; crosses two multiples of 16: <sig#>..<sig#>**

Cause: This error occurs when defining trace labels. A trace label may not contain trace signals crossing two 16-bit boundaries. For example, the command "**tlb name 1..32**" will cause this error because "name" contains signals that cross the 15-16 and 31-32 16-bit boundaries.

Action: Redefine your trace label so that no more than one 16-bit boundary is crossed.

1103 **!ERROR 1103! Invalid bit range; out of bounds: <sig#>..<sig#>**

Cause: This error occurs when defining trace labels, and you have attempted to assign non-existent trace signals to a label.

Action: Enter the trace activity command to view the trace signals present, and use only these signals when defining trace labels.

1104 **!ERROR 1104! Invalid bit range; too wide: <sig#>..<sig#>**

Cause: This error occurs when defining trace labels, and you have attempted to assign more than 32 trace signals to a label.

Action: Use more than one trace label to define over 32 trace signals.

1105 **!ERROR 1105! Unable to delete label; used by emulation analyzer: <label>**

Cause: This error occurs when you attempt to delete an emulation trace label that is currently being used as a qualifier in the emulation trace specification or is currently specified in the emulation trace format.

Action: Display the emulation trace sequencer specification in the configuration, display the emulation trace patterns in the complex configuration, or display the trace format to see where the label is used. Also, you should check **tcq** and **tpq** for uses of that label. You must change the pattern or format specification to remove the label before you can delete it.

1108 **!ERROR 1108! Unable to redefine label; used by emulation analyzer: <label>**

Cause: This error occurs when you attempt to redefine an emulation trace label that is currently used as a qualifier in the emulation trace specification.

Action: Display the emulation trace sequencer specification in the easy configuration, display the emulation trace patterns in the complex configuration, or display the emulation trace format to see where the label is used. You must change the pattern or format specification to remove the label before you can redefine it.

1130 **!ERROR 1130! Illegal base for count display**

Cause: When specifying the trace format, counts may only be displayed relative or absolute. When counting states, the count is always displayed as a decimal number.

Action: Respecify the trace format without using a base for the count column. Also, you can use "**,A**" to specify that counts be displayed absolute, or you can use "**,R**" to specify that counts be displayed relative.

1131 **!ERROR 1131! Illegal base for mnemonic disassembly display**

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.

1132 **!ERROR 1132! Illegal base for sequencer display**

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.

1133 **!ERROR 1133! Trace format command failed; using old format**

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.

1138 **!ERROR 1138! Illegal width for symbol display: <width>**

Cause: This error occurs when the value specified for the trace format address field width is not valid.

Action: Enter the **tf** command again, and specify the width of the address field for symbol display within the range of 4 to 55.

1139 **!ERROR 1139! Illegal width for addr display, mne not specified**

Cause: This error occurs when you specify a width for the address field in the **tf** command, but do not include the **mne** option.

Action: Enter the command again, and include the **mne** option.

1141 **!ERROR 1141! Symbol display unavailable without mne field**

Cause: This error occurs when you try to display symbols, but have not included the **mne** option to the **tf** command.

Action: Don't try to display symbols unless the **mne** field has already been specified.

1202 **!ERROR 1202! Trigger position out of bounds: <bounds>**

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).

1207 **!ERROR 1207! Invalid clock channel: <name>**

Cause: Valid clock channels are L, M, and N.

Action: Respecify the command using valid clock channels.

1209 **!ERROR 1209! Operator must be "and" or "or": <expression>**

Cause: When combining trace labels to specify trace patterns (in simple expressions or with the **tpat** command), an operator of either "and" or "or" must appear between the label qualifiers.

Action: See Chapter 11 for information on valid patterns.

1210 **!ERROR 1210! Illegal mix of = and !=**

Cause: When combining trace labels to specify patterns (in simple expressions or with the **tpat** command), all labels must either be equal to values or not equal to values.

Action: See Chapter 11.

1211 **!ERROR 1211! Illegal mix of and/or**

Cause: When combining trace labels to specify patterns (in simple expressions or with the **tpat** command), all label qualifiers must either be ANDed together or ORed together. You cannot mix these operators.

Action: See Chapter 11 for more information.

1212 **!ERROR 1212! Conflict with overlapping label: <label>**

Cause: When combining trace labels to specify patterns (in simple expressions or with the **tpat** command), you cannot combine labels which are defined for common trace signals. For example, the following easy configuration commands will result in this error: **tlb low8 0..7; tlb low16 0..15; tg low8=0 and low16=1**.

Action: Either omit one of the overlapping labels, or redefine your labels so they do not contain common trace signals. You could also circumvent this error by using don't cares in the appropriate places; for the example shown in cause, you could specify patterns **tg low8=0xx0xY and low16=1**.

1213 **!ERROR 1213! Illegal mix of !=/and**

Cause: When combining trace labels to specify patterns (in simple expressions or with the **tpat** command), labels that are not equal to values must be ORed together so that the entire pattern specifies a "not equals" condition.

Action: See Chapter 11 for information on valid patterns.

1214 **!ERROR 1214! Illegal mix of =/or**

Cause: When combining trace labels to specify patterns (in simple expressions or with the **tpat** command), labels that are equal to values must be ANDed together so that the entire pattern specifies an "equals" condition.

Action: See Chapter 11 for information on valid patterns.

1215 **!ERROR 1215! Comparator must be = or !=: <label>**

Cause: When combining trace labels to specify patterns (in simple expressions or with the **tpat** command), the value of the label can only be specified with the "=" or "!=" operators.

Action: See Chapter 11 for more information.

1217 **!ERROR 1217! Illegal pattern name: <name>**

Cause: Valid pattern names are p1 through p8.

Action: Use only valid pattern names.

1218 **!ERROR 1218! Illegal comparator for range qualifier: !=**

Cause: When specifying a range with the **trng** command, you cannot use the"!=" operator.

Action: Use the “!r” range name.

1219 **!ERROR 1219! Range cannot be combined with any other qualifier**

Cause: For example, the following easy configuration command will result in this error: **tsto addr=400..4ff and data=40**.

Action: Do not attempt to combine labels when using range qualifiers.

1221 **!ERROR 1221! Range resource in use**

Cause: This error occurs when you attempt to use two different range expressions in the “easy” configuration trace specification or when you attempt to redefine the “complex” configuration range resource while it is currently being used as a qualifier in the trace specification.

Action: Do not attempt to use more than one range expression in the “easy” configuration trace specification. 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.

1224 **!ERROR 1224! Sequence term number out of range: <term>**

Cause: This error occurs when a sequencer qualification command (**tif**, **telif**, **tsq**, or **tsto**) 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.

1225 **!ERROR 1225! Sequence term not contiguous: <term>**

Cause: This error occurs when you attempt to insert a sequence term that is not between existing terms or after the last term. For example, the following easy configuration commands will result in this error: **tg any; tsq -i 4**.

Action: Be sure that the sequence term you enter is either between existing sequence terms or after the last sequence term.

1226 **!ERROR 1226! Too many sequence terms**

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.

1227 **!ERROR 1227! Sequence term not defined: <term>**

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.

1228 **!ERROR 1228! One sequence term required**

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.

1234 **!ERROR 1234! Invalid occurrence count: <number>**

Cause: Occurrence counts may be from 1 to 65535.

Action: Reenter the command with a valid occurrence count.

1239 **!ERROR 1239! Clock speed not available with current count qualifier**

Cause: This error occurs when you attempt to specify a fast (F) or very fast (VF) maximum qualified clock speed when counting time (**tcq time**). This error also occurs when you attempt to specify a very fast (VF) maximum qualified clock speed when counting states (for example, **tcq addr=400**).

Action: Change the count qualifier; then reenter the command. See Chapter 5 for more information.

1240 **!ERROR 1240! Count qualifier not available with current clock speed**

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

Action: Change the clock speed; then change the count qualifier. See Chapter 5 for more information.

1241 **!ERROR 1241! Invalid qualifier resource or operator: <expression>**

Cause: When specifying complex expressions, you have either specified an illegal pattern or used an illegal operator.

Action: See Chapter 11 for more information.

1245 **!ERROR 1245! Range qualifier not accessible in easy configuration**

Cause: This error occurs when you attempt to use the **trng** command in the easy configuration.

Action: Changing into the complex configuration will allow you to use the **trng** command; otherwise, specify the range in easy configuration command expressions.

1246 **!ERROR 1246! Pattern qualifiers not accessible in easy configuration**

Cause: This error occurs when you attempt to use the **tpat** command in the easy configuration.

Action: Changing into the complex configuration will allow you to use the **tpat** command; otherwise, specify the patterns in easy configuration command expressions.

1248 **!ERROR 1248! Range term used more than once**

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.

1249 **!ERROR 1249! Invalid qualifier expression: <expression>.**

Cause: This error message is shown with the errors that occur when patterns, the range, or the arm condition is used more than once within a set. This error message also occurs when intraset operators are not the same. For example, the following complex expression will result in this error: **p1 ~ p2 | p3**.

Action: See Chapter 11 for more information.

1250 **!ERROR 1250! Arm term used more than once**

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.

1251 **!ERROR 1251! Trigger term cannot be term 1**

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.

1253 **!ERROR 1253! Invalid pod number: <pod#>**

Cause: This error message occurs when you attempt to specify a slave clock for a non-existent analyzer pod.

Action: Use the trace activity command to display the valid pod numbers, and use only these numbers when entering commands.

1302 **!ERROR 1302! Trig1 signal cannot be driven and received**

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.

1303 **!ERROR 1303! Trig2 signal cannot be driven and received**

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.

1305 **!ERROR 1305! CMB execute; emulation trace started**

Cause: This status message informs you that an emulation trace measurement has started as a result of a CMB execute signal (as specified by the "**tx -e**" command).

13

Data File Formats

File formats for binary trace lists and symbol files

The HP 64700 Series Emulator defines two special file formats that allow compact representations of trace list information and symbol information. These file formats may be useful to you if you decide to build software tools that interact with the Terminal Interface. Both file formats are described in this chapter.

Binary/Hexadecimal Trace List Format

The **tl** command supports two options, **-b** (binary) and **-x** (hexadecimal) which allow you to dump the trace list to your host for post processing.

When you request a binary trace list dump from the HP 64700 Emulator (**-b** option), the emulator sends the data using the HP 64000 **transfer** protocol. You must use an 8-bit communications channel to successfully transfer the data (HP 64700 and the host device must both be configured to send and receive eight bits).

The hexadecimal trace list dump (**-x** option) also uses the HP 64000 **transfer** protocol, but does not require an 8-bit communications channel. However, twice as many characters will be transmitted as would be in the binary format.

Six primary trace list records may be transferred. These are:

- No Trigger Record
- Empty Trace Record
- New State Data Record
- More State Data Record
- New Timing Data Record
- More Timing Data Record

Each record has at least one byte. The first byte identifies the record type.

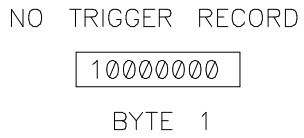
Other fields in the record, containing one or more bytes of information, provide additional information about the trace.

The Data Records contain secondary record structures which hold the actual trace information. For the State Data Records, the secondary record is the Trace State record; for the Timing Data Records, the secondary record is the Trace Sample record.

Each record structure is accompanied by a diagram. Note that line breaks in the diagram are not EOL characters in the record.

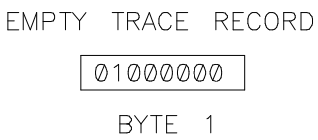
This section describes all record types, but the 68020 and 68030/EC030 emulators do not use the timing data record types, because they do not support an external analyzer.

No Trigger Record



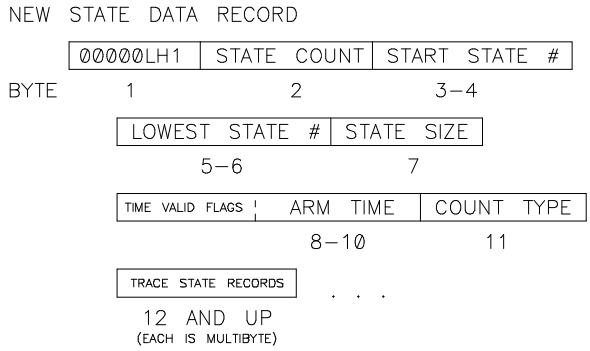
One byte indicating that the trigger condition of the current trace is not in memory. Trace data cannot be displayed until the trigger condition occurs and is placed in trace memory or until the trace is halted. Therefore, this is the only record that will be sent when the trace list is requested, because no others are available.

Empty Trace Record



One byte indicating that the most recent trace was halted before any states were stored. Therefore, this will be the only record sent.

New State Data Record



One byte indicating that this is the first trace list data displayed for the current or most recent trace.

If L=1, this is the only record being sent. Otherwise, one or more More Data Records follow.

If H=1, this record contains the highest numbered state this trace can have. Therefore, this is the end of the trace list. If the state count for this record is zero, the highest numbered state can be computed by subtracting 1 from the start state.

state count

One byte indicating how many trace states are contained in this record. This will be zero (0) if none of the requested states exist.

start state

Two bytes containing the starting state number (in the range -1024 through 1023), most significant byte first.

lowest state

Two bytes containing the lowest state number in the entire trace list, MSB first. Note that if the trace is halted after this record is sent, lower-numbered states may be valid.

state size

One byte indicating how many bytes of trace data will be in each trace state. This does not include the store cause or count data bytes.

arm time

Three bytes containing the time from arm to trigger, MSB first. The lower 20 bits contain the absolute value of the actual time, in 40 ns units.

The time alignment between HP 64700-Series emulators has a large margin of error (+/- 100 ns) due to delay variances in the trigger paths.

The correlation between the arm time counter value and the value displayed on screen should be as follows:

```
count       time
-----       ------------------------------------------------------
0000h       Arm occured an unknown amount of time after the trigger
0001h       Arm occured an unknown amount of time after the trigger
0002h     -40 ns - Arm input actually came after trigger was sampled but
                   still caused arm state to occur before trigger
                   internal to the elan chip.
0003h       0 ns
0004h      40 ns
0005h      80 ns
  .         .
  .         .
  .         .
FFFFh      2.621280 ms  This is now the maximum arm to trigger interval
                        that can be displayed.
```

The highest four bits contain status flags as follows:

high nibble = XVS0

If X = 1, the arm time is invalid, either because the arm signal was ignored (e.g., "tarm always"), or because the state analyzer clock speed was fast or very fast (e.g., "xtck -s F"). The 20 bits of time value will be 0.

If V = 1, the arm counter overflowed (if S = 0) or underflowed (if S = 1). For overflow, the 20 bits of time value contain the maximum time value, (1^20)-4, representing 41.94288 ms. For underflow, the S flag is set (see below), and the 20 bits of time value contain the absolute value of the minimum count, -1, representing -40 ns.

If S = 1, the arm time is negative. The 20 bits of time value contain the absolute value of the actual count.

count type

One ASCII character indicating the type of count data contained in each trace state.

"T" indicates each trace state contains a time count.

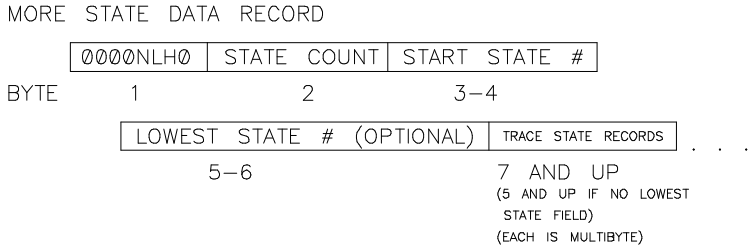
"S" indicates each trace state contains a state count.

"N" indicates that no count data is available.

first trace state..last trace state

Each of these records is in the trace state format described below. Each record is n bytes in length; n is the state size value (described above) plus one byte indicating the reason for storage of this state and an optional two bytes with count data information.

More State Data Record



One byte indicating this is more data from the same trace as the most recent New State Data Record.

If L=1, this is the last record sent. Otherwise, additional More Data Records follow.

If H=1, this record contains the highest numbered state in the trace; this is the end of the trace list. If the state count for this record is zero (0), the highest numbered state can be computed by subtracting one (1) from the start state.

If N=1, this record contains a new lowest state. The starting state number can change if the trace is halted; if it changes, it will always become more negative. It

can change a maximum of one time for a given trace list. N=1 will never occur unless L=1.

state count

One byte indicating the number of trace states contained in this record. This will be zero (0) if none of the requested states exist.

start state

Two bytes containing the starting state number (in the range -1024..1023), most significant byte (MSB) first.

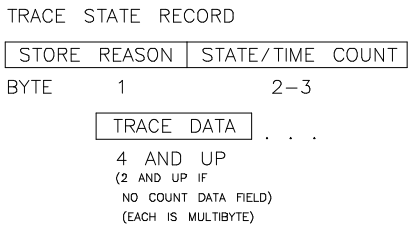
lowest state

Optional two bytes containing the lowest state number in the entire trace list, most significant byte (MSB) first. These bytes are only present if the record type has N=1.

first trace state..last trace state

Each of these records is in the trace state format described below. Each record is a variable number of bytes in length. The length is the state size value (described above) plus one byte indicating the reason for storage of this state and an optional two bytes with count data information.

Trace State Record



state type

One ASCII character indicating the reason this state was stored.

"Q" indicates this state satisfied a sequence branch qualifier (defined by **tif** or **telif**).

"S" indicates this state satisfied the store qualifier (defined by **tsto**).

"P" indicates this state satisfied the prestore qualifier. The count data field bytes below will be omitted for this state. Prestore states are marked as such only if a state or time count was specified for the trace (defined by **tcq**).

count data

Optional two bytes containing the state or time count for this state. The count value is relative to the previous non-prestore state. These bytes are omitted if the count type field in the New State Data Record was "N", or if this state is a prestore state (state type field in this record is "P"). The count data is encoded as follows (first byte is on the left):

eeeeemmm mmmmmmmm

e represents five bits of exponent.

m represents 11 bits of mantissa.

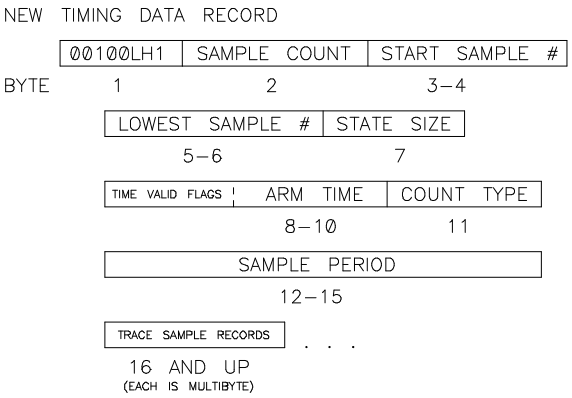
The value represented is (m*(2^e)) + (2^(11+e)) - (2^11)

Time counts are in 40-nanosecond units.

trace data

Trace data for this state, most significant byte (MSB) first. The length of this trace data is given by the state size field in the New State Data Record.

New Timing Data Record



record type = 00100LH1

One byte indicating this is the first trace list data displayed for the current or most recent trace.

If L=1, this is the only record sent. Otherwise, one (1) or more More Timing Data Records follow.

If H=1, this record contains the highest numbered sample in the trace; this is the end of the trace list. If the sample count field for this record is zero (0), the highest numbered sample can be computed by subtracting one (1) from the start sample field.

sample count

One byte indicating how many trace samples are contained in this record. This will be zero if no samples are present.

start sample

Two bytes containing the starting sample number (-1024..1023), most significant byte (MSB) first.

lowest sample

Two bytes containing the lowest sample number in the entire trace list, MSB first. Note that if the trace is halted after this record is sent, lower-numbered samples may become valid.

state size

One byte indicating the number of bytes of trace data in each trace sample. Note the relationship to the count type field.

arm time

Three bytes containing the time from arm to trigger, MSB first. The lower 20 bits contain the absolute value of the actual time, in 40 ns units.

The time alignment between HP 64700-Series emulators has a large margin of error (+/- 100 ns) due to delay variances in the trigger paths.

The correlation between the arm time counter value and the value displayed on screen should be as follows:

```
count       time
-----       ------------------------------------------------------
0000h       Arm occured an unknown amount of time after the trigger
0001h       Arm occured an unknown amount of time after the trigger
0002h     -40 ns - Arm input actually came after trigger was sampled but
                   still caused arm state to occur before trigger
                   internal to the elan chip.
0003h       0 ns
0004h      40 ns
0005h      80 ns
  .         .
  .         .
  .         .
FFFFh      2.621280 ms  This is now the maximum arm to trigger interval
                        that can be displayed.
```

The highest four bits contain status flags as follows:

high nibble = XVS0

If X = 1, the arm time is invalid, either because the arm signal was ignored (e.g., "tarm always"), or because the state analyzer clock speed was fast or very fast (e.g., "xtck -s F"). The 20 bits of time value will be 0.

If V = 1, the arm counter overflowed (if S = 0) or underflowed (if S = 1). For overflow, the 20 bits of time value contain the maximum time value, (1^20)-4, representing 41.94288 ms. For underflow, the S flag is set (see below), and the 20 bits of time value contain the absolute value of the minimum count, -1, representing -40 ns.

If S = 1, the arm time is negative. The 20 bits of time value contain the absolute value of the actual count.

count type

One ASCII character indicating the type of count data contained in each Trace Sample record.

"T" indicates the timing analyzer was set to transitional mode. Each Trace Sample record contains a six byte field which contains the delta time (in nanoseconds) since the last transition. A two-byte field containing the trace data taken at the delta time interval is also in the Trace Sample record.

"S" indicates the timing analyzer was set to standard mode. Each Trace Sample record contains only the two bytes of trace data.

"G" indicates the timing analyzer was set to glitch mode. Each trace sample consists of a two-byte trace data field and a two-byte glitch data field.

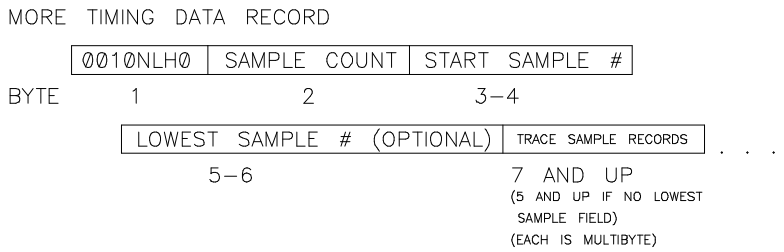
sample period

Four bytes containing the number of nanoseconds (ns) between samples.

first trace sample..last trace sample

Trace Sample records of the size defined in the sample size field (note relationship to the count type field).

More Timing Data Record



One byte indicating this is more data from the same trace as the most recent New Timing Data Record.

If L=1, this is the last record sent. Otherwise, additional More Timing Data Records follow.

If H=1, this record contains the highest-numbered sample in the trace; this is the end of the trace list. If the sample count field for this record is zero (0), the highest numbered sample can be computed by subtracting one (1) from the start sample field.

If N=1, this record contains a new lowest sample. The starting sample number can change if the trace is halted; if it changes, it will always become more negative. It can only change once for a given trace list. N=1 will only occur if L=1.

sample count

One byte indicating the number of Trace Sample records in this record. This will be zero (0) if no Trace Samples are present (the analyzer did not find the requested data in the last trace.)

start sample

Two bytes containing the starting sample number (in the range -1024..1023), most significant byte (MSB) first.

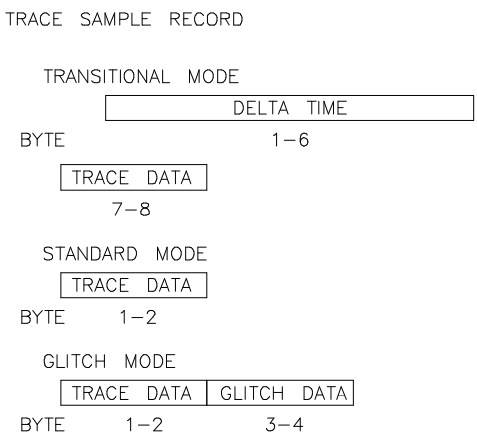
lowest sample

Optional two bytes containing the lowest sample number in the entire trace list, most significant byte (MSB) first. These two bytes are present only if the record type has N=1.

first trace sample..last trace sample

Trace Sample records of the size defined in the sample size field (note relationship to the count type field).

Trace Sample Records



Trace Sample records are variant records which are components of the New Timing Data Record and More Timing Data Record. The structure of the Trace Sample Record depends on the count type field in the Timing Data Records.

Transitional Mode (count type = "T")

delta time

Six bytes of data defining the delta time (elapsed time) since the last transition, in nanoseconds (ns).

trace data

Two bytes of trace data sampled at the delta time value given.

Standard Mode (count type = "S")

trace data

Two bytes of trace data sampled at the standard sampling period (see the **xtsp** command).

Glitch Mode (count type = "G")

trace data

Two bytes of trace data sampled at the standard sampling period (see the **xtsp** command).

glitch

Two bytes indicating the occurrences of glitches on any channel.

Symbol Files

The HP 64747/64748 emulators can load an ASCII text file containing symbol definitions.

Three types of symbols can be defined: local, global, and user. Only local and global symbols can be loaded from a symbol file; user symbols can only be created with the **sym** command.

Global symbols are general memory references. They represent the equivalent of "GLOBAL" or "PUBLIC" variables in compiled programs.

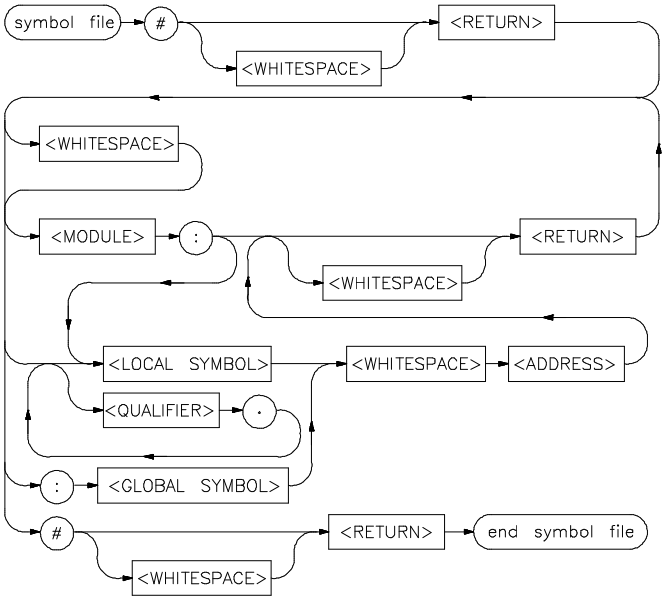
Local symbols are grouped by "module." The primary purpose of a module is to group local symbols, but can represent any arrangement of local symbols desired. Local symbols created by a higher level language processor are defined by implementation.

A module is usually a source file name, and symbols are function or procedure names. In a symbol file, any organizational scheme can be used to manage local symbols. While the module name can be equivalent to a source file name, or some other physical or logical entity, it is not necessary. Therefore, if memory is in short supply, you can organize the "local" symbols to allow for easy deletion of old symbols, and loading of new symbols that reference locations of interest.

Address references for all symbol types are absolute addresses.

Symbol file syntax

A symbol file is an ASCII text file. The format of this file is represented by:



<WHITESPACE>	This is one or more <SP> (space) or <HT> (horizontal tab) characters or a combination of these characters.
<RETURN>	This is a <LF> (line feed) or <CR><LF> (carriage return, line feed pair); a <CR> (carriage return) alone is not recognized.
<ADDRESS>	This is a valid address specification for the emulator being used.
<MODULE>	This defines a module name.
<LOCAL SYMBOL>	This is a local symbol reference. A local symbol definition line must include, or follow, a module name, or an error will occur when loading the file.

<GLOBAL SYMBOL>	This is a global symbol reference.
<QUALIFIER>	This allows you to specify label hierarchies. Its use is dependent on the implementation.
:	This is the literal colon (":").
.	This is the literal period (".")
#	This is the literal pound sign ("#").

Examples

Defining Local Symbols

Local symbols must include, or be preceded by, a module name reference. Therefore, the files

```
#
:main 0@p
GetAttrib:
Buffer 100@p
Pointer 120@p
#
```

and

```
#
:main 0@p
GetAttrib:Buffer 100@p
GetAttrib:Pointer 120@p
#
```

will produce the same result when loaded.

After loading either symbol file, enter:

```
M> sym
```

You will see:

```
sym main=00000@p
sym GetAttrib:Buffer=00100@p
sym GetAttrib:Pointer=00120@p
```

Naming Array Elements

You may wish to load symbols that name elements of an array to make referring to the array elements more explicit. If your array has four elements, each element is 10h bytes long, and begins at 2000h, the symbol file will contain the following:

```
#
ARRAY:
E1=2000@d
E2=2010@d
E3=2020@d
E4=2030@d
#
```

After loading the symbol file, enter:

```
M> sym
```

You will see, at least in part:

```
  sym ARRAY:E1=2000@d
  sym ARRAY:E2=2010@d
  sym ARRAY:E3=2020@d
  sym ARRAY:E3=2030@d
```

If you no longer need the references to ARRAY elements, you can remove the symbols with the command:

```
M> sym -dl ARRAY
```

14

Specifications and Characteristics

Electrical specifications and characteristics for the MC68020 and MC68030/MC68EC030 emulators

Processor Compatibility

HP 64747: Compatible with the Motorola MC68030 (HP64747A requires MMU be disabled/HP 64747B supports MMU enabled) and MC68EC030 processors, and with any processors that meet all specifications of the MC68030 or MC68EC030 processors.

HP 64748: Compatible with the Motorola MC68020 and MC68EC020 processors, and with any processors that meet all specifications of the MC68020 or MC68EC020 processors.

Electrical

Maximum clock speed

HP 64747: the maximum external clock speed is 40 MHz. No wait states are required for any memory accesses in asynchronous bus mode. In synchronous and burst modes, emulation memory requires one wait state for all clock speeds. Target system memory requires one wait state in synchronous and burst modes for clock frequencies above 25 MHz.

HP 64748: the maximum external clock speed is 33 MHz. No wait states are required for target system memory. Emulation memory requires one wait state at speeds greater than 25 MHz. (Dual-port emulation memory will run at full rated clock speed without wait states.)

HP 64747 Electrical Specifications

HP 64747 - DC ELECTRICAL SPECIFICATIONS ($V_{cc=5.0\ Vdc}$ ±5%; GND=0 Vdc; T_A=0 to 70°C					
Characteristic		**Symbol**	**Min**	**Max**	**Unit**
Input High Voltage		V_{IH}	2.0	V_{CC}	V
Input Low Voltage		V_{IL}	-0.5	0.8	V
Input Leakage Current $GND \leq V_{in} \leq V_{cc}$	$\overline{BR}$,$\overline{BGACK}$,$\overline{IPLx}$,$\overline{CDIS}$	I_{IN}	-2.5	2.5	µA
Input High Current	$\overline{CBACK}$,$\overline{CIIN}$,$\overline{STERM}$,$\overline{BERR}$,$\overline{AVEC}$, $\overline{DSACKx}$,$\overline{HALT}$, $\overline{MMUDIS}$ CLK,$\overline{RESET}$	I_{IH}	- - -	0 25 50	µA
Input Low Current	$\overline{RESET}$,$\overline{CBACK}$,$\overline{CIIN}$,$\overline{STERM}$ CLK,$\overline{BERR}$,$\overline{AVEC}$,$\overline{DSACKx}$,$\overline{HALT}$,$\overline{MMUDIS}$	I_{IL}	- -	-1.4 -0.25	mA
Output High Voltage I_{OH}=-400µA	$\overline{A0}$-A31,$\overline{AS}$,$\overline{BG}$,D0-D31,$\overline{DBEN}$,$\overline{DS}$, $\overline{ECS}$,R/$\overline{W}$,STATUS,REFILL,IPEND, $\overline{OCS}$,$\overline{RMC}$,SIZ0-SIZ1,FC0-FC2, $\overline{CBREQ}$,C1OUT	V_{OH}	2.4	-	V
Output Low Voltage I_{OL}= 2.5 mA I_{OL}= 3.2 mA I_{OL}= 4.5 mA I_{OL}= 5.3 mA I_{OL}= 2.0 mA I_{OL}= 9.3 mA	 $\overline{A0}$-A31,FC0-FC2,SIZ0-SIZ1 $\overline{BG}$,D0-D31 R/$\overline{W}$,$\overline{RMC}$ $\overline{AS}$,$\overline{DS}$,$\overline{DBEN}$,$\overline{IPEND}$ $\overline{STATUS}$,REFILL,$\overline{CBREQ}$,C1OUT,$\overline{ECS}$,$\overline{OCS}$ $\overline{RESET}$	V_{OL}	 - - - - - -	 0.5 0.5 0.5 0.5 0.5 0.5	V
Power Dissipation[1]	T_A=0°C	P_D	-	3.4	W
Capacitance V_{in}=0V,T_A=25°C, f=1 MHz		C_{in}	-	20	pF
Load Capacitance	$\overline{A0}$-A31,FC0-FC2,SIZ0-SIZ1,R/$\overline{W}$,$\overline{CBREQ}$, C1OUT All Other	C_L	- -	100 50	pF

Notes:

1 The power dissipation is an indication of how much power is drawn from the target system by the emulator probe, not the true power dissipation of the emulator probe.

HP 64747 - AC ELECTRICAL SPECIFICATIONS - CLOCK INPUT						
Num	**Characteristic**	**HP 64747 25MHz**		**HP 64747 40MHz**		**Unit**
		Min	**Max**	**Min**	**Max**	
	Frequency of Operation	12.5	25	25	40	MHz
1	Cycle Time	40	80	25	40	ns
2,3	Clock Pulse Width	19	61	11.5	29	ns
4,5	Rise and Fall Times	-	4	-	2	ns

HP64747 -AC ELECTRICAL SPECIFICATIONS — READ AND WRITE CYCLES (V_{CC}=5.0 Vdc ± 5%; GND=0 Vdc; T_A=0 to 70°C)						
Num	**Characteristic**	**HP 64747 25MHz**		**HP 64747 40MHz**		**Unit**
		Min	**Max**	**Min**	**Max**	
6	Clock High to FC,Size,$\overline{\text{RMC,C1OUT}}$,Address Valid Clock High to IPEND Valid	0 0	20 24*	0 0	14 24*	ns ns
6A	Clock High to $\overline{\text{ECS,OCS}}$ Asserted	0	15	0	10	ns
6B	FC,Size,$\overline{\text{RMC,C1OUT}}$,Address Valid to Negating $\overline{\text{ECS}}$ IPEND Valid to Negating $\overline{\text{ECS}}$	3 1*	- -	3 -7*	- -	ns ns
7	Clock High to FC,Size,$\overline{\text{RMC,C1OUT}}$,Address, Data High Impedance	0	40	0	25	ns
8	Clock High to FC,Size,$\overline{\text{RMC,IPEND,C1OUT}}$, Address Invalid	0	—-	0	—-	ns

HP64747 -AC ELECTRICAL SPECIFICATIONS — READ AND WRITE CYCLES
(V_{CC}=5.0 Vdc ± 5%; GND=0 Vdc; T_A=0 to 70°C)

Num	Characteristic	HP 64747 25MHz		HP 64747 40MHz		Unit
		Min	Max	Min	Max	
9	Clock Low to $\overline{AS,DS,CBREQ}$ Asserted	3	18	2	15*	ns
9A[1]	$\overline{AS}$ to $\overline{DS}$ Assertion Skew (Read)	-10	10	-8*	8*	ns
9B[14]	$\overline{AS}$ Asserted to $\overline{DS}$ Asserted (Write)	27	—-	14*	—-	ns
10	$\overline{ECS}$ Width Asserted	10	—-	5	—-	ns
10A	$\overline{OCS}$ Width Asserted	10	—-	5	—-	ns
10B[7]	$\overline{ECS}$,$\overline{OCS}$ Width Negated	10	—-	5	—-	ns
11	FC,Size,$\overline{RMC,C1OUT}$,Address Valid to $\overline{AS}$ Asserted (and $\overline{DS}$ Asserted,Read)	7	—-	5	—-	ns
12	Clock Low to $\overline{AS,DS,CBREQ}$ Negated	0	18	0	15*	ns
12A	Clock Low to $\overline{ECS}$/$\overline{OCS}$ Negated	0	18	0	12	ns
13	$\overline{AS}$, $\overline{DS}$ Negated to FC,Size,$\overline{RMC}$,C1OUT,Address Invalid	6*	—-	-2*	—-	ns
14	$\overline{AS}$ (and $\overline{DS}$, Read) Width Asserted (Asynchronous Cycle)	70	—-	30	—-	ns
14A[11]	$\overline{DS}$ Width Asserted, Write	30	—-	18	—-	ns
14B	$\overline{AS}$ (and $\overline{DS}$ Read) Width Asserted (Synchronous Cycle)	30	—-	18	—-	ns
15	$\overline{AS}$, $\overline{DS}$ Width Negated	30	—-	18	—-	ns
15A[8]	$\overline{DS}$ Negated to $\overline{AS}$ Asserted	25	—-	16	—-	ns
16	Clock High to $\overline{AS,DS,R/W,DBEN,CBREQ}$ High Impedance	—-	40	—-	25	ns
17	$\overline{AS}$, $\overline{DS}$ Negated to R/$\overline{W}$ Invalid	6*	—-	-2*	—-	ns
18	Clock High to R/$\overline{W}$ High	0	20	0	14	ns
20	Clock High to R/$\overline{W}$ Low	0	20	0	15	ns

HP64747 -AC ELECTRICAL SPECIFICATIONS — READ AND WRITE CYCLES (V_{CC}=5.0 Vdc ± 5%; GND=0 Vdc; T_A=0 to 70°C)						
Num	**Characteristic**	**HP 64747 25MHz**		**HP 64747 40MHz**		**Unit**
		Min	**Max**	**Min**	**Max**	
21[6]	R/$\overline{W}$ High to $\overline{AS}$ Asserted	7	—-	5	—-	ns
22[6]	R/$\overline{W}$ Low to $\overline{DS}$ Asserted (Write)	47	—-	24	—-	ns
23	Clock High to Data Out Valid	—-	20	—-	19*	ns
24	Data Out Valid to Negating Edge of $\overline{AS}$	5	—-	1*	—-	ns
25[6,11]	$\overline{AS}$, $\overline{DS}$ Negated to Data Out Invalid	7	—-	0*	—-	ns
25A[9,11]	$\overline{DS}$ Negated to $\overline{DBEN}$ Negated (Write)	7	—-	1*	—-	ns
26[6,11]	Data Out Valid to $\overline{DS}$ Asserted (Write)	7	—-	1*	—-	ns
27	Data-In Valid to Clock Low (Synchronous Setup)	6*	—-	6*	—-	ns
27A	Late $\overline{BERR}$, $\overline{HALT}$ Asserted to Clock Low (Setup)	8*	—-	8*	—-	ns
28[12]	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{DSACKx}$,$\overline{BERR}$,$\overline{HALT}$,$\overline{AVEC}$ Negated (Asynchronous Hold)	0	40	0	20	ns
28A[12]	Clock Low to $\overline{DSACKx}$,BERR,HALT,$\overline{AVEC}$ Negated (Synchronous Hold)	8	70	6	40	ns
29[12]	$\overline{DS}$ Negated to Data-In Invalid (Asynchronous Hold)	0	—-	0	—-	ns
29A[12]	$\overline{DS}$ Negated to Data-In High-Impedance	—-	40	—-	25	ns
30[12]	Clock Low to Data-In Invalid (Synchronous Hold)	8	—-	6	—-	ns
30A[12]	Clock Low to Data-In High-Impedance (Read followed by Write)	—-	60	—-	30	ns
31[2]	$\overline{DSACKx}$ Asserted to Data-In Valid (Asynchronous Data Setup)	—-	26	—-	12*	ns
31A[3]	$\overline{DSACKx}$ Asserted to $\overline{DSACKx}$ Valid (Skew)	—-	7	—-	1*	ns
32	$\overline{RESET}$ Input Transition Time	—-	1.5	—-	1.5	Clks

HP64747 -AC ELECTRICAL SPECIFICATIONS — READ AND WRITE CYCLES
(V_{CC}=5.0 Vdc ± 5%; GND=0 Vdc; T_A=0 to 70°C)

Num	Characteristic	HP 64747 25MHz		HP 64747 40MHz		Unit
		Min	Max	Min	Max	
33	Clock Low to $\overline{BG}$ Asserted	0	20	0	14	ns
34	Clock Low to $\overline{BG}$ Negated	0	20	0	14	ns
35	$\overline{BR}$ Asserted to $\overline{BG}$ Asserted ($\overline{RMC}$ Not Asserted)	1.5	3.5	1.5	3.5	Clks
37	$\overline{BGACK}$ Asserted to $\overline{BG}$ Negated	1.5	3.5	1.5	3.5	Clks
37A	$\overline{BGACK}$ Asserted to $\overline{BR}$ Negated	0	1.5	0	1.5	Clks
39[6]	$\overline{BG}$ Width Negated	60	—-	30	—-	ns
39A	$\overline{BG}$ Width Asserted	60	—-	30	—-	ns
40	Clock Low to $\overline{DBEN}$ Negated (Read)	0	20	0	21*	ns
41	Clock Low to $\overline{DBEN}$ Negated (Read)	0	20	0	21*	ns
42	Clock Low to $\overline{DBEN}$ Asserted (Write)	0	20	0	21*	ns
43	Clock High to $\overline{DBEN}$ Negated (Write)	0	20	0	21*	ns
44	R/$\overline{W}$ Low to $\overline{DBEN}$ Asserted (Write)	7	—-	5	—-	ns
45[5]	$\overline{DBEN}$ Width Asserted (Asynchronous Read) DBEN Width Asserted (Asynchronous Write)	40 80	—-	22 45	—-	ns
45A[9]	DBEN Width Asserted (Synchronous Read) DBEN Width Asserted (Synchronous Write)	5 40	—- —-	5 22	—- —-	ns ns
46	R/$\overline{W}$ Width Asserted (Asynchronous Write or Read)	100	—-	50	—-	ns
46A	R/$\overline{W}$ Width Asserted (Synchronous Write or Read)	60	—-	30	—-	ns
47A	Asynchronous Input Setup Time ($\overline{HALT}$,$\overline{BERR}$,$\overline{DSACKx}$) Asynchronous Input Setup Time (IPLx)	7* 12*	—-	7* 12*	—-	ns

HP64747 -AC ELECTRICAL SPECIFICATIONS — READ AND WRITE CYCLES
(V_{CC}=5.0 Vdc ± 5%; GND=0 Vdc; T_A=0 to 70°C)

Num	Characteristic	HP 64747 25MHz		HP 64747 40MHz		Unit
		Min	Max	Min	Max	
47B	Asynchronous Input Hold Time from Clock Low	8	—-	6	—-	ns
48[4]	$\overline{\text{DSACKx}}$ Asserted to $\overline{\text{BERR}}$,$\overline{\text{HALT}}$ Asserted		25		14	ns
53	Data Out Hold from Clock High	3	—-	2	—-	ns
55	R/$\overline{\text{W}}$ Asserted to Data Bus Impedance Change	20	—-	11	—-	ns
56	$\overline{\text{RESET}}$ Pulse Width (Reset Instruction)	512	—-	512	—-	Clks
57	$\overline{\text{BERR}}$ Negated to $\overline{\text{HALT}}$ Negated (Rerun)	0	—-	0	—-	ns
58[10]	$\overline{\text{BGACK}}$ Negated to Bus Driven	1	—-	1	—-	Clks
59[10]	$\overline{\text{BG}}$ Negated to Bus Driven	1	—-	1	—-	Clks
60[13]	Synchronous Input Valid to Clock High (Setup Time)	4*	—-	4*	—-	ns
61[13]	Clock High to Synchronous Input Invalid (Hold Time)	8	—-	6	—-	ns
62	Clock Low to $\overline{\text{STATUS,REFILL}}$ Asserted	0	20	0	15	ns
63	Clock Low to $\overline{\text{STATUS,REFILL}}$ Negated	0	20	0	15	ns

Notes:

1 This number can be reduced to 5 nanoseconds if strobes have equal loads.

2 If the asynchronous setup time (#47) requirements are satisfied, the DSACKx low to data setup time (#31) and DSACKx low to BERR low setup time (#48) can be ignored. The data must only satisfy the data-in to clock low setup time (#27) for the following clock cycle, BERR must only satisfy the late BERR low to clock low setup time (#27A) for the following clock cycle.

3 This parameter specifies the maximum allowable skew between DSACK0 to DSACK1 asserted or DSACK1 to DSACK0 asserted, specification #47 must be met by DSACK0 or DSACK1.

4 This specification applies to the first DSACKx signal asserted. In the absence of DSACKx, BERR is an asynchronous input using the asynchronous input setup time (#47).
5 DBEN may stay asserted on consecutive write cycles.
6 The minimum values must be met to guarantee proper operation. If this maximum value is exceeded, BG may be reasserted.
7 This specification indicates the minimum high time for ESC and OCS in the event of an internal cache hit followed immediately by a cache miss or operand cycle.
8 This specification guarantees operation with the MC68881 or MC68882, which specifies a minimum time for DS negated to AS asserted (specification #13A in the MC68881/MC68882 User's Manual). Without this specification, incorrect interpretation of specifications #9A and #15 would indicate that the MC68030 does not meet the MC68881 requirements.
9 This specification allows a system designed to guarantee data hold times on the output side of data buffers that have output enable signals gererated with DBEN. The timing on DBEN precludes its use for synchronous read cycles with no wait states.
10 These specifications allow system designers to guarantee that an alternate bus master has stopped driving the bus when the MC68030/EC030 regains control of the bus after an arbitration sequence.
11 DS will not be asserted for synchronous write cycles with no wait states.
12 These hold times are specified with respect to strobes (asynchronous) and with respect to the clock (synchronous). The designer is free to use either hold time.
13 Synchronous inputs must meet specifications #60 and #61 with stable logic levels for all rising edges of the clock. These values are specified relative to the high level of the rising clock edge.
14 This specification allows system designers to qualify the CS of an MC68881/MC68882 with AS (allowing 7 ns for a gate delay) and still meet the CS to DS setup time requirement (spec 8B) of the MC68881/MC68882.
15 The asterisk "*" means there is some difference between the specification for the emulator and the specification for the MC68030/EC030 for this entry.

HP 64748 Electrical Specifications

HP 64748 - DC ELECTRICAL SPECIFICATIONS (V_{cc}=5.0 Vdc ± 5%; GND=0 Vdc; T_A=0 to 70°C)					
Characteristic		**Symbol**	**Min**	**Max**	**Unit**
Input High Voltage		V_{IH}	2.0	V_{CC}	V
Input Low Voltage		V_{IL}	-0.5	0.8	V
Input Leakage Current GND≤V_{in}≤V_{cc}	$\overline{BR}$, $\overline{BGACK}$,$\overline{IPLx}$	I_{IN}	-2.5	2.5	µA
Input High Current	$\overline{BERR}$,$\overline{AVEC}$,$\overline{DSACK}$x CLK,$\overline{RESET}$,HALT	I_{IH}	—- —-	25 50	µA
Input Low Current	$\overline{RESET}$,$\overline{HALT}$ CLK,$\overline{BERR}$,AVEC,$\overline{DSACK}$x	I_{IL}	—- —-	-1.4 -0.25	mA
Output High Voltage I_{OH}=-400µA	A0-A31,$\overline{AS}$,$\overline{BG}$,D0-D31,$\overline{DBEN}$,$\overline{DS}$, $\overline{ECS}$,R/$\overline{W}$,$\overline{IPEND}$,$\overline{OCS}$,RMC, SIZ0-SIZ1,FC0-FC2	V_{OH}	2.4	—-	V
Output Low Voltage I_{OL} = 2.5 mA I_{OL} = 3.2 mA I_{OL} = 4.5 mA I_{OL} = 5.3 mA I_{OL} = 2.0 mA I_{OL} = 9.3 mA	 A0-A31,$\overline{FC0-FC2}$,$\overline{SIZ0-SIZ1}$ $\overline{BG}$,D0-D31 R/$\overline{W}$,$\overline{RMC}$ $\overline{AS}$,$\overline{DS}$,$\overline{DBEN}$,$\overline{IPEND}$ $\overline{ECS}$,$\overline{OCS}$ $\overline{RESET}$,$\overline{HALT}$	V_{OL}	 —- —- —- —- —- —-	 0.5 0.5 0.5 0.5 0.5 0.5	V
Power Dissipation	T_A = 0°C T_A = 70°C	P_D	—-	200 200	mW
Capacitance V_{in}=0V, T_A =25°C, f=1MHz		C_{in}	—-	20	pF
Load Capacitance	A0-A31,$\overline{FC0-FC2}$,$\overline{SIZ0-SIZ1}$,R/$\overline{W}$ All Other	C_L		100 50	pF

Notes:

1 The power dissipation is an indication of how much power is drawn from the target system by the emulator probe, not the true power dissipation of the emulator probe.

HP 64748 - AC ELECTRICAL SPECIFICATIONS - CLOCK INPUT						
Num	**Characteristic**	**MC68020 33 MHz**		**HP 64748**		**Unit**
		Min	**Max**	**Min**	**Max**	
	Frequency of Operation	12.5	33	12.5	33	MHz
1	Cycle Time	30	80	30	80	ns
2,3	Clock Pulse Width	14	66	14	66	ns
4,5	Rise and Fall Times	—-	3	—-	3	ns

HP 64748 - AC ELECTRICAL SPECIFICATIONS — READ AND WRITE CYCLES (V_{CC}=5.0 Vdc ± 5%; GND=0 Vdc; T_A=0 to 70°C)						
Num	**Characteristic**	**MC68020 33MHz**		**HP 64748**		**Unit**
		Min	**Max**	**Min**	**Max**	
6	Clock High to FC,Size,RMC,Address Valid	0	21	0	21	ns
6A	Clock High to ECS,OCS Asserted	0	10	0	10	ns
7	Clock FC,Size,RMC,Address,Data High Impedance	0	30	0	30	ns
8	Clock High to FC,Size,RMC,Address Invalid	0	—-	0	—-	ns
9	Clock Low to AS,DS Asserted	3	15	3	20*	ns
9A[1]	$\overline{AS}$ to $\overline{DS}$ Assertion Skew (Read)	-10	10	-12	12	ns
9B[11]	$\overline{AS}$ Asserted to $\overline{DS}$ Asserted (Write)	22	—-	20	—-	ns

HP 64748 - AC ELECTRICAL SPECIFICATIONS — READ AND WRITE CYCLES
(V_{CC}=5.0 Vdc ± 5%; GND=0 Vdc; T_A=0 to 70°C)

Num	Characteristic	MC68020 33MHz		HP 64748		Unit
		Min	Max	Min	Max	
10	$\overline{ECS}$ Width Asserted	10	—-	10	—-	ns
10A	$\overline{OCS}$ Width Asserted	10	—-	10	—-	ns
10B[7]	$\overline{ECS}$,$\overline{OCS}$ Width Negated	5	—-	5	—-	ns
11	FC,Size,$\overline{RMC}$,Address Valid to $\overline{AS}$ Asserted (and $\overline{DS}$ Asserted,Read)	5	—-	5	—-	ns
12	Clock Low to $\overline{AS}$,$\overline{DS}$ Negated	0	15	0	20*	ns
12A	Clock Low to $\overline{ECS}$/$\overline{OCS}$ Negated	0	15	0	15	ns
13	$\overline{AS}$, $\overline{DS}$ Negated to FC,Size,$\overline{RMC}$,Address Invalid	5	—-	0*	—-	ns
14	$\overline{AS}$ (and $\overline{DS}$, Read) Width Asserted (Asynchronous Cycle)	50	—-	48	—-	ns
14A	$\overline{DS}$ Width Asserted, Write	25	—-	23	—-	ns
15	$\overline{AS}$, $\overline{DS}$ Width Negated	25	—-	23	—-	ns
15A[8]	$\overline{DS}$ Negated to $\overline{AS}$ Asserted	18	—-	16	—-	ns
16	Clock High to $\overline{AS}$,$\overline{DS}$,R/$\overline{W}$,$\overline{DBEN}$ High Impedance	—-	30	—-	30	ns
17	$\overline{AS}$, $\overline{DS}$ Negated to R/$\overline{W}$ Invalid	5	—-	0*	—-	ns
18	Clock High to R/$\overline{W}$ High	0	15	0	15	ns
20	Clock High to R/$\overline{W}$ Low	0	15	0	15	ns
21	R/$\overline{W}$ High to $\overline{AS}$ Asserted	5	—-	5	—-	ns
22	R/$\overline{W}$ Low to $\overline{DS}$ Asserted (Write)	35	—-	35	—-	ns
23	Clock High to Data Out Valid	—-	18	—-	23*	ns
25	$\overline{AS}$, $\overline{DS}$ Negated to Data Out Invalid	5	—-	2	—-	ns

HP 64748 - AC ELECTRICAL SPECIFICATIONS — READ AND WRITE CYCLES
(V_{CC}=5.0 Vdc ± 5%; GND=0 Vdc; T_A=0 to 70°C)

Num	Characteristic	MC68020 33MHz		HP 64748		Unit
		Min	Max	Min	Max	
25A[9]	$\overline{DS}$ Negated to $\overline{DBEN}$ Negated (Write)	5	—	0	—	ns
26	Data Out Valid to $\overline{DS}$ Asserted (Write)	5	—	5	—	ns
27	Data-In Valid to Clock Low (Synchronous Setup)	5	—	10*	—	ns
27A	Late $\overline{BERR}$, $\overline{HALT}$ Asserted to Clock Low (Setup)	5	—	10*	—	ns
28	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{DSACKx}$,$\overline{BERR}$,$\overline{HALT}$,$\overline{AVEC}$ Negated (Asynchronous Hold)	0	40	0	30*	ns
29	$\overline{DS}$ Negated to Data-In Invalid (Asynchronous Hold)	0	—	0	—	ns
29A	$\overline{DS}$ Negated to Data-In High-Impedance	—	30	—	30	ns
30	Clock Low to Data-In Invalid (Synchronous Hold)	6	—	6	—	ns
31[2]	$\overline{DSACKx}$ Asserted to Data-In Valid (Asynchronous Data Setup)	—	17	—	13*	ns
31A[3]	$\overline{DSACKx}$ Asserted to $\overline{DSACKx}$ Valid (Skew)	—	10	—	7	ns
32	$\overline{RESET}$ Input Transition Time	—	1.5	—	1.5	Clks
33	Clock Low to $\overline{BG}$ Asserted	0	20	0	30	ns
34	Clock Low to $\overline{BG}$ Negated	0	20	0	30	ns
35	$\overline{BR}$ Asserted to $\overline{BG}$ Asserted ($\overline{RMC}$ Not Asserted)	1.5	3.5	1.5	3.5	Clks
37	$\overline{BGACK}$ Asserted to $\overline{BG}$ Negated	1.5	3.5	1.5	3.5	Clks
37A[6]	$\overline{BGACK}$ Asserted to $\overline{BR}$ Negated	0	1.5	0	1.5	Clks
39	$\overline{BG}$ Width Negated	50	—	50	—	ns
39A	$\overline{BG}$ Width Asserted	50	—	50	—	ns
40	Clock Low to $\overline{DBEN}$ Negated (Read)	0	15	0	15	ns

HP 64748 - AC ELECTRICAL SPECIFICATIONS — READ AND WRITE CYCLES (V_{CC}=5.0 Vdc ± 5%; GND=0 Vdc; T_A=0 to 70°C)						
Num	**Characteristic**	**MC68020 33MHz**		**HP 64748**		**Unit**
		Min	**Max**	**Min**	**Max**	
41	Clock Low to $\overline{\text{DBEN}}$ Negated (Read)	0	15	0	15	ns
42	Clock Low to $\overline{\text{DBEN}}$ Asserted (Write)	0	15	0	15	ns
43	Clock High to $\overline{\text{DBEN}}$ Negated (Write)	0	15	0	15	ns
44	R/$\overline{\text{W}}$ Low to $\overline{\text{DBEN}}$ Asserted (Write)	5	—	5	—	ns
45[5]	$\overline{\text{DBEN}}$ Width Asserted (Asynchronous Read) $\overline{\text{DBEN}}$ Width Asserted (Asynchronous Write)	30 60	—	30 60	—	ns
46	R/$\overline{\text{W}}$ Width Asserted (Asynchronous Write or Read)	75	—	75	—	ns
47A	Asynchronous Input Setup Time ($\overline{\text{HALT}}$,$\overline{\text{BERR}}$,$\overline{\text{DSACKx}}$) Asynchronous Input Setup Time (IPLx)	5 5	—	10* 10*	—	ns
47B	Asynchronous Input Hold Time from Clock Low	10	—	10	—	ns
48[4]	$\overline{\text{DSACKx}}$ Asserted to $\overline{\text{BERR}}$,$\overline{\text{HALT}}$ Asserted		15		11	ns
53	Data Out Hold from Clock High	0	—	0	—	ns
55	R/$\overline{\text{W}}$ Asserted to Data Bus Impedance Change	20	—	10	—	ns
56	$\overline{\text{RESET}}$ Pulse Width (Reset Instruction)	512	—	512	—	Clks
57	$\overline{\text{BERR}}$ Negated to $\overline{\text{HALT}}$ Negated (Rerun)	0	—	2	—	ns
58[10]	$\overline{\text{BGACK}}$ Negated to Bus Driven	1	—	1	—	Clks
59[10]	$\overline{\text{BG}}$ Negated to Bus Driven	1	—	1	—	Clks

Notes:

1 This number can be reduced to 5 nanoseconds if strobes have equal loads.

2 If the asynchronous setup time (#47) requirements are satisfied, the DSACKx low to data setup time (#31) and DSACKx low to BERR low setup time (#48) can be ignored. The data must only satisfy the data-in to clock low setup time (#27) for the following clock cycle, BERR must only satisfy the late BERR low to clock low setup time (#27) for the following clock cycle.

3 This parameter specifies the maximum allowable skew from DSACK0 to DSACK1 asserted or DSACK1 to DSACK0 asserted, specification #47 must be met by DSACK0 or DSACK1.

4 This specification applies to the first DSACKx signal asserted. In the absence of DSACKx, BERR is an asynchronous input using the asynchronous input setup time (#47).

5 DBEN may stay asserted on consecutive write cycles.

6 The minimum values must be met to guarantee proper operation. If this maximum value is exceeded, BG may be reasserted.

7 This specification indicates the minimum high time for ECS and OCS in the event of an internal cache hit followed immediately by a cache miss or operand cycle.

8 This specification guarantees operation with the MC68881, which specifies a minimum time for DS negated to AS asserted (specification #13A in the MC68881 User's Manual). Without this specification, incorrect interpretation of specifications #9A and #15 would indicate that the MC68020 does not meet the MC68881 requirements.

9 This specification allows a system designed to guarantee data hold times on the output side of data buffers that have output enable signals generated with DBEN. The timing on DBEN precludes its use for synchronous read cycles with no wait states.

10 These specifications allow system designers to guarantee that an alternate bus master has stopped driving the bus when the MC68020 regains control of the bus after an arbitration sequence.

11 This specification allows system designers to qualify the CS of an MC68881/MC68882 with AS (allowing 7 ns for a gate delay) and still meet the CS to DS setup time requirement (spec 8B) of the MC68881/MC68882.

12 The asterisk "*" means there is some difference between the specification for the emulator and the specification for the MC68020 for that entry.

Physical (HP 64747 and HP 64748)

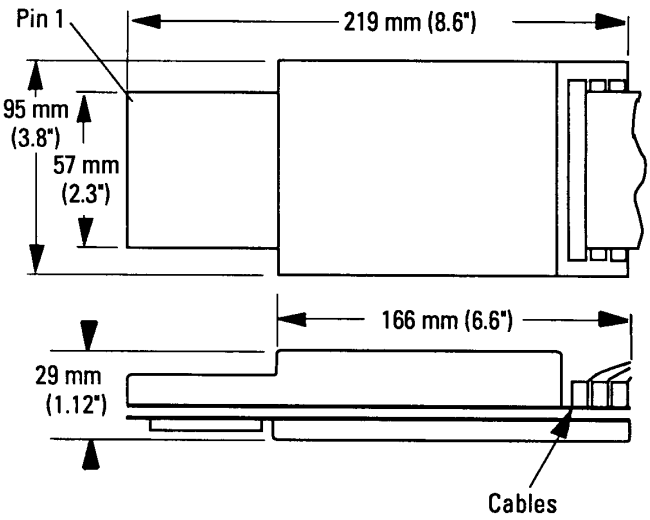
Emulator Dimensions

173 mm height x 325 mm width x 389 mm depth (6.8 in. x 12.8 in. x 15.3 in.)

Cable Length

Emulator to target system, approximately 914 mm (36 inches).

Probe dimensions



Environmental (HP 64747 and HP 64748)

Temperature

Operating, 0° to +40° C (+32° to +104° F); nonoperating, -40° C to +60° C (-40° F to +140° F).

For the HP 64747 (MC68030/EC030), 100 lfm of airflow over the probe is recommended to ensure optimum performance.

Altitude

Operating, 4600 m (15 000 ft), nonoperating, 15 300 m (50 000 ft).

Relative Humidity

15% to 95%.

BNC, labeled TRIGGER IN/OUT (HP 64747 and HP 64748)

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 (HP 64747 and HP 64748)

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

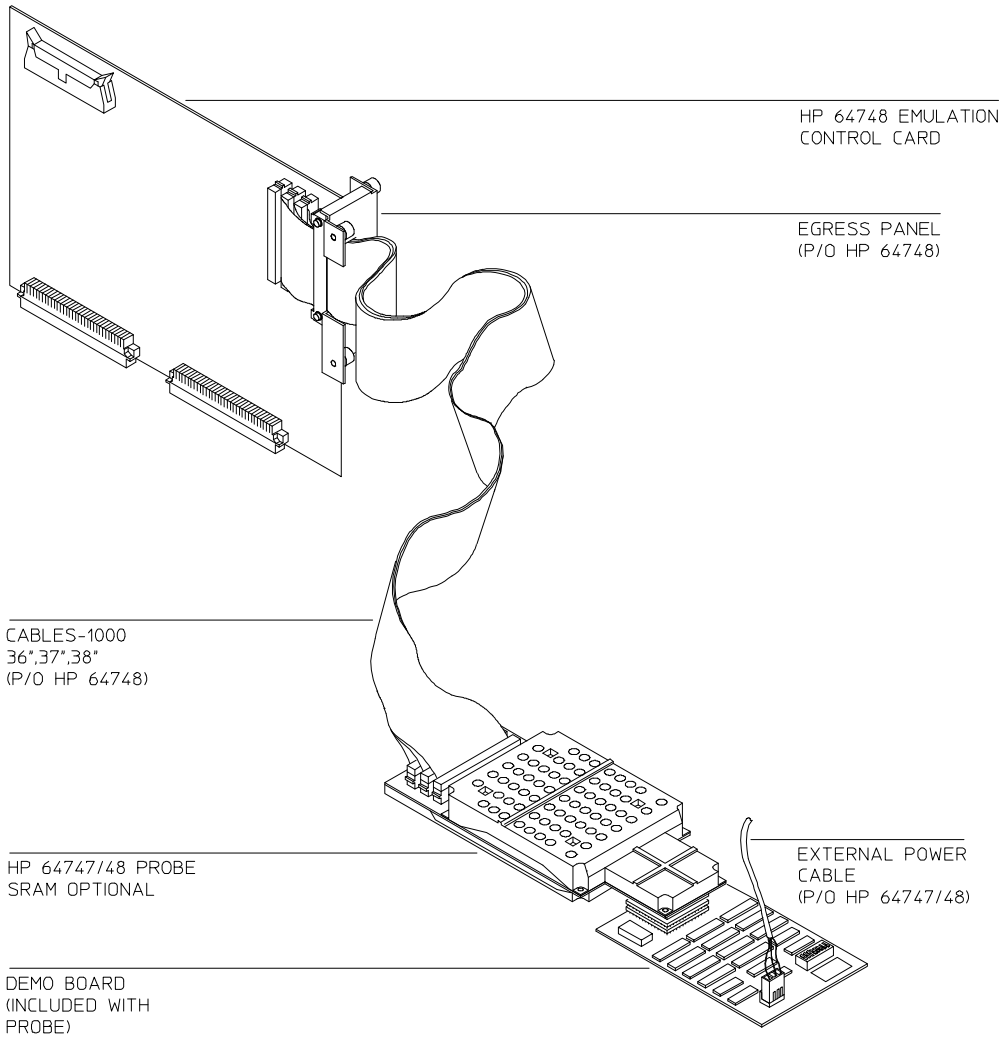
In This Part

Chapter 15, “Installation and Service,” tells you how to set up the emulator and verify performance of the emulator. It also tells you what to do when you suspect that there is a problem with the operation of the emulator.

When you finish installation of the emulator, go to part 1 of this manual and perform the Quick Start procedure.

15

Installation and Service


HP 64748 EMULATION
CONTROL CARD
EGRESS PANEL
(P/O HP 64748)
CABLES-1000
36",37",38"
(P/O HP 64748)
HP 64747/48 PROBE
SRAM OPTIONAL
EXTERNAL POWER
CABLE
(P/O HP 64747/48)
DEMO BOARD
(INCLUDED WITH
PROBE)

The HP 64747 and HP 64748 emulator subsystems plug into the HP 64700 Card Cage.

Equipment supplied

The HP 64747/48 package contains

- The HP 64747/48 Active Probe emulator
- This emulator terminal interface user's guide

Support Services Information is at the back of each binder.

Tools needed

- Flat blade screw driver

Antistatic precautions

Circuit boards contain electrical components that are easily damaged by small amounts of static electricity. To avoid damage to the emulator cards, follow these guidelines:

- If possible, work at a static-free workstation.
- Handle the parts only by the edges; do not touch components or traces.
- Use a grounding wrist strap that is connected to the card cage chassis. There is a ground recepticle near the serial number tag on the rear panel of the HP 64700 card cage.

Installation Overview

When you order a complete system (an HP 64747/48 emulator in an HP 64700 Card Cage), the HP 64747/48 emulator is already installed in the card cage. The installation procedure in this manual is provided for users that already have an HP 64700 Card Cage and want to install the HP 64747/48 emulator in the card cage, or would like to replace parts of the emulator.

Caution

If you already have a modular HP 64700 Series Card Cage and want to remove the existing emulator and insert an HP 64747 or HP 64748 emulator in its place, the HP 64700 Series generic firmware and analyzer firmware may NOT be compatible, and the software will indicate incompatibility. Instructions for updating firmware from a PC or HP 9000 are provided in the *HP 64700 Card Cage Installation/Service* manual.

After hardware installation, run a performance test to verify that the emulator is working properly. After you verify performance of the emulator, you are ready to update software, and use the emulator.

Use this chapter to accomplish hardware removal and installation procedures, running performance verification, and ordering parts. See the *HP 64700 Series Card Cage Installation/Service* manual for information on system configurations, installing product software, software updates, and ordering parts related to the card cage. Make sure that power to the HP 64700 Card Cage is off before removing or installing any hardware.

To install the emulator into the HP 64700 Card Cage

1 Turn power off.

2 Remove the HP 64700 Card Cage top cover.

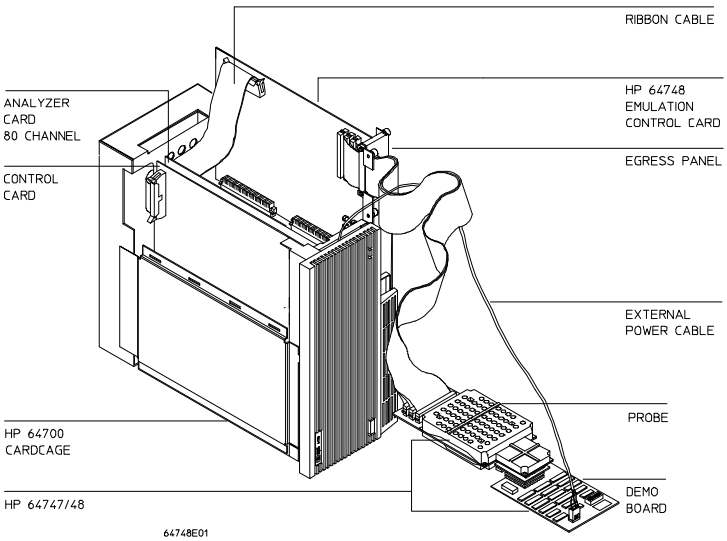
3 Remove the HP 64700 Card Cage side panel.

4 Install the HP 64747/48 Probe and 64748 Emulation Control Card and secure in place with the four Egress Panel screws.

5 Connect ribbon cable from the Emulation Control Card to the analyzer card.

6 Replace the HP 64700 Card Cage side panel and top cover.

Instructions for removing and installing cards into and out of the HP 64700 Series Card Cage are provided in the *HP 64700 Installation/Service* manual and on the label on the bottom of the HP 64700 Card Cage. The only difference is the firmware flashed in the Emulation Control Card.



To install the demo board external power cable

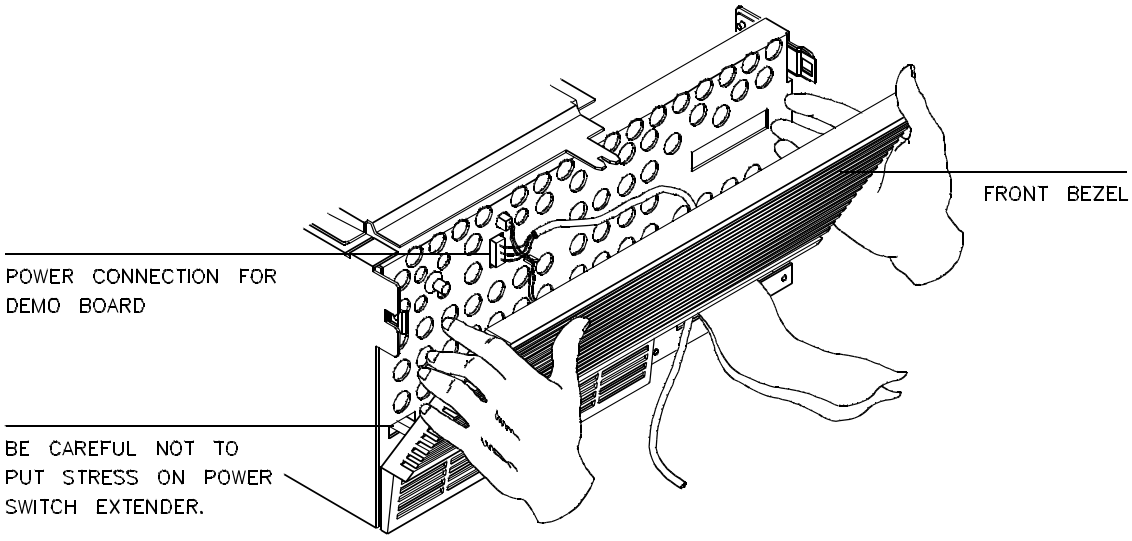
1 Remove the front bezel of the HP 64700 Card Cage. Be careful because the plastic ears are easily broken on the front bezel.

2 Connect the demo board external power cable to the front panel.

3 Route the demo board external power cable to the lower right of the HP 64700 Card Cage front panel and out of the front panel egress opening as shown in the figure.

4 Replace the front bezel.

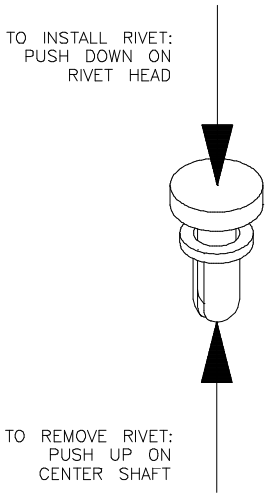
Instructions for removing and installing the HP 64700 Card Cage Front Bezel are provided in the *HP 64700 Card Cage Installation/Service* manual.



To remove and install plastic rivets

- Press the top of the plastic rivet head to secure in place.
- Push the center shaft to remove plastic rivet.

The HP 64747/48 probe has top and bottom plastic covers that are secured in place with plastic rivets. A package of extra plastic rivets are provided with the HP 64747/48 emulator. The plastic rivets are secured in place by pressing the flat part of the rivet. The plastic rivets are removed by pushing on the center shaft of the rivet. Discard rivets that are removed and use new rivets from the package of extra rivets. Removed rivets are difficult to reuse.



To remove and install plastic covers to access SIMM sockets on the probe

Top Cover

1 Remove six plastic rivets to remove the top plastic cover.

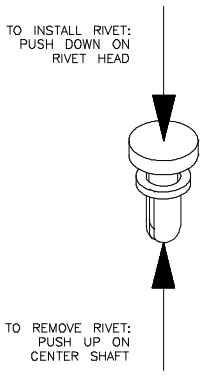
2 Install in reverse order, making sure to add washers to two of the rivets shown in the figure below.

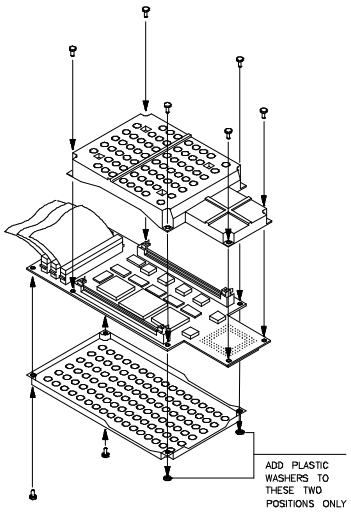
Bottom Cover

3 Remove four plastic rivets to remove the bottom cover. Both top and bottom covers can be removed.

4 To reinstall, secure covers in place by inserting six plastic rivets from the top (plastic washers are installed on two of the rivets) and two plastic rivets from the bottom as shown in the figure.

Remove top plastic cover to access the SIMM memory sockets. Remove bottom cover only to replace a defective active probe on the exchange program.

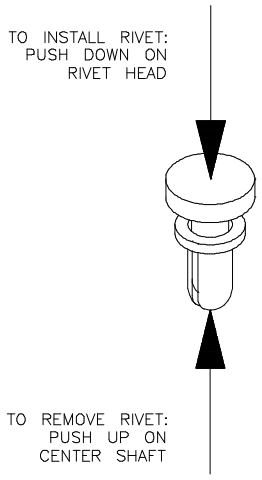




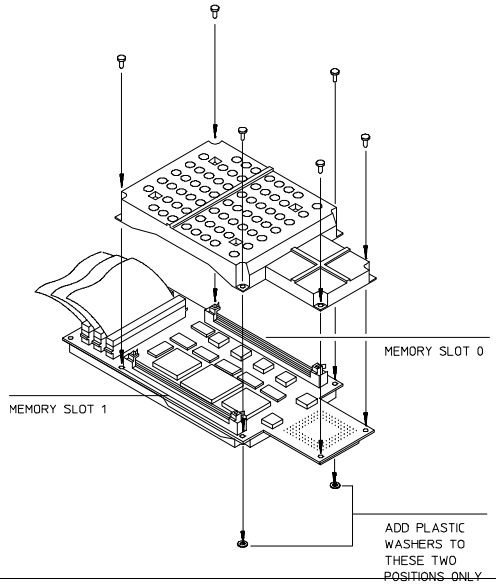
To install emulation memory modules

(Observe antistatic precautions listed on the third page of this chapter)

1. Remove six rivets from the probe top plastic cover by pushing the center shaft of the plastic rivets.



2. Remove the top plastic cover. The bottom cover will still be attached to the probe with two plastic rivets.

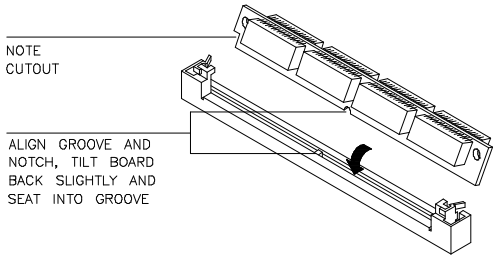


There are three types of emulation memory modules that can be inserted into sockets on the probe. You can use any combination of memory modules. Determine the total memory you are using and which sockets you will place the memory into. The HP 64171A memory module is the 256-Kbyte, 35-ns SRAM; the HP 64171B is the 1-Mbyte 35-ns SRAM; and the HP 64173A is the 4-Mbyte, 25-ns SRAM.

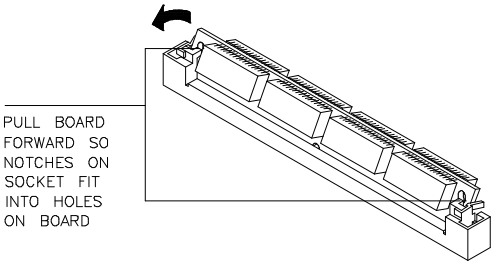
3. Determine placement of memory modules. See "To assign memory map terms" in Chapter 7, for more information.

To install emulation memory modules

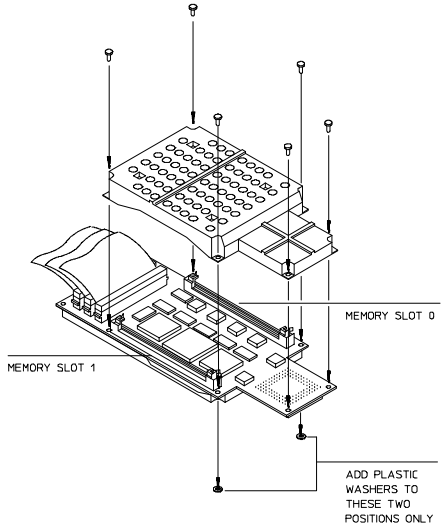
4. Insert the emulation memory module into the appropriate probe socket. The SRAM module is keyed and will go in only one way.



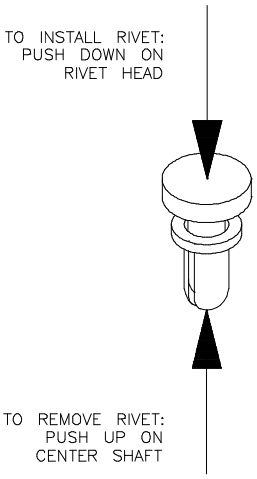
5. Secure the emulation memory module in place by pushing forward so notches on the socket fit into the holes on the memory module.



6. Install the top plastic cover with six plastic rivets. Install washers on two of the rivets as shown in the figure.



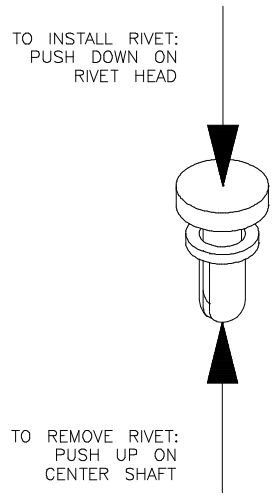
7. Secure the plastic rivets in place by pushing on the head of each plastic rivet.



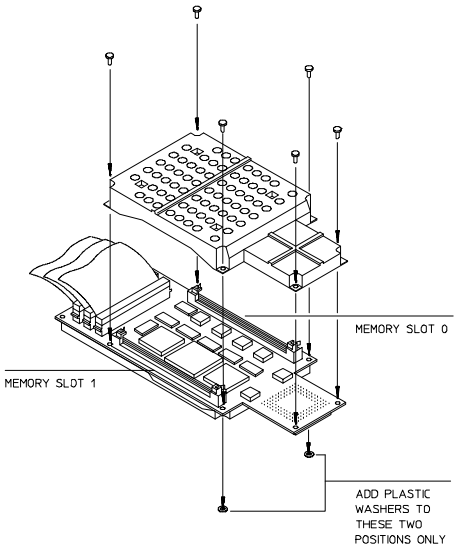
To remove emulation memory modules

(Observe antistatic precautions)

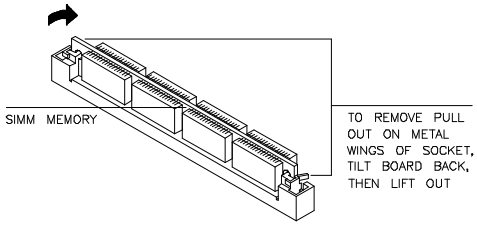
1. Remove six rivets from the probe top plastic cover by pushing the center shaft of the plastic rivet.



2. Remove the top plastic cover. The bottom cover will still be attached to the probe with two plastic rivets.



3. Remove the memory module by pulling the wings of the socket and tilting the module back. Lift the memory module out.

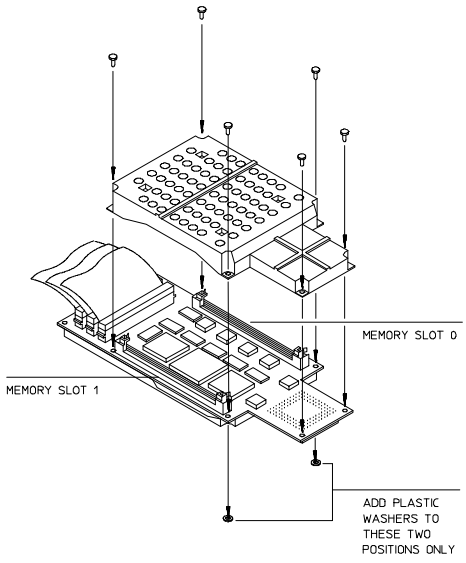


After removal of the memory module, you may want to install another memory module in its place. Refer to the section “To Install Static Random Access Memory Modules” earlier in this chapter. Then come back here to complete the installation process.

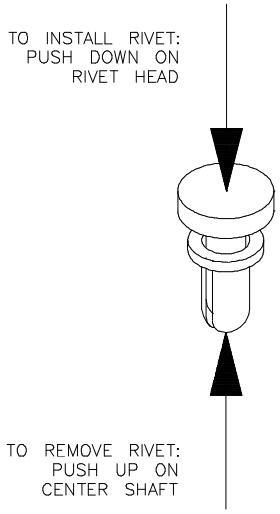
If you do not want to install a memory module, proceed to the next step.

To remove emulation memory modules

4. Install the top plastic cover with six plastic rivets. Install washers on two of the rivets as shown in the figure.



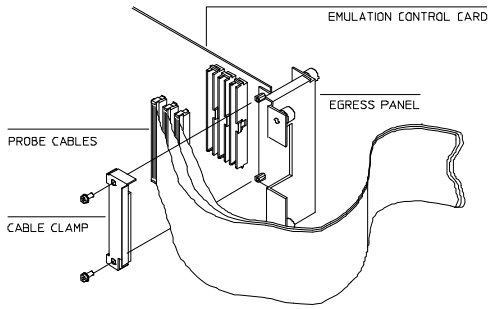
5. Secure the plastic rivets in place by pushing on the head of the plastic rivet.



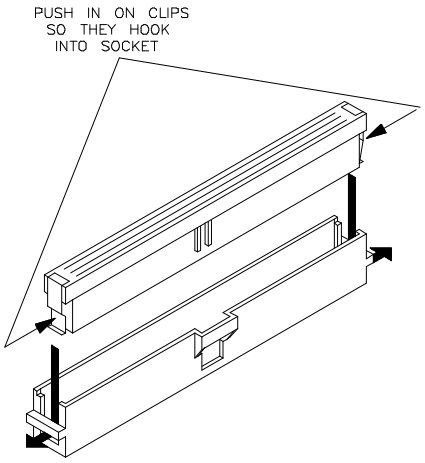
To install the emulator probe cable

The Probe Cables consist of three ribbon cables. The longest cable connects to J3 of the Emulation Control Card, and to J1 of the Probe. The shortest cable connects to J1 of the Emulation Control Card and J3 of the Probe. The ribbon cables are held in place on the Emulation Control Card by a cable clamp attached with two screws. No clamp holds the ribbon cables on the Probe.

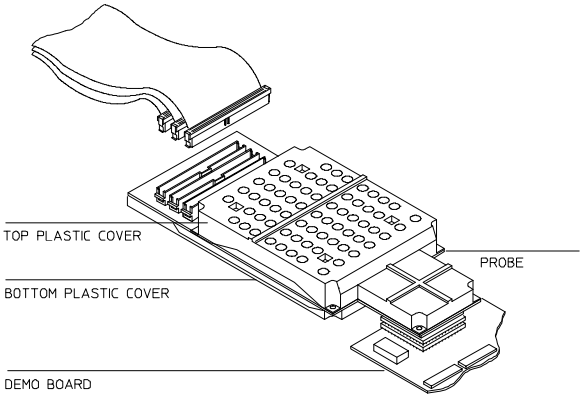
1. Secure the ribbon cable on the Emulation Control Card with cable clamp and two screws.



2. Insert the ribbon cables into the appropriate sockets, making sure that the clips hook into the sockets as shown.



3. Insert the cables into the appropriate sockets on the Probe.



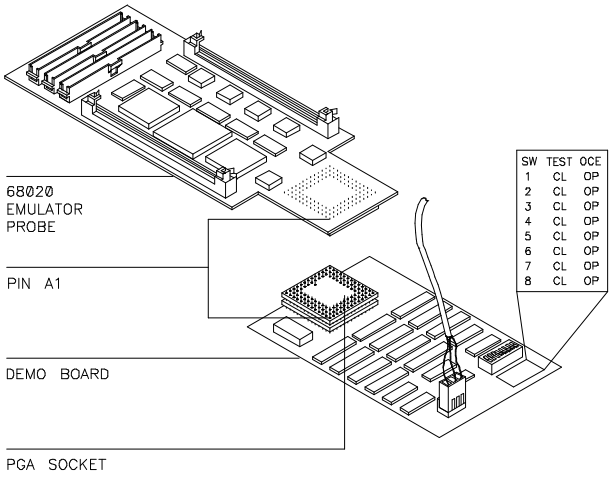
To connect the probe to the demo board

- Turn off emulator power.
- Carefully install the probe into the demo board and connect the external power cable to J1.
- Set Demo Board switches for TEST or OCE.
- Turn emulator power on.

The HP 64747/48 probe and demo board must be connected together when running the emulator in “out of circuit” mode (not plugged into a target system), or when making performance verification tests.

When you install the probe into the demo board, be very careful not to bend any of the pins. The socket of the MC68020 is keyed to prevent inserting the probe into the demo board socket incorrectly. There is no keying on the MC68030/EC030, so use extreme care.

After connecting the probe to the demo board, set the TEST/OCE switches. Press each switch rocker down in direction of desired setting. Use the closed (CL) position for all switches when running performance verification tests (TEST), and the open (OP) position for out of circuit emulation (OCE).



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 "To Install the Demo Board Power Cable".)

5 Set all switches on the Demo Board to the TEST (CL - closed) position.

6 Turn on power to the HP 64700 Card Cage.

7 Establish communication with the emulator from your host or ASCII terminal and obtain a prompt (such as **R>**).

8 Enter: **pv <N>** <return>

where **<N>** is an integer that specifies the number of times to run the set of performance verification tests. It is sometimes necessary to enter the command **tcf -e** before running performance verification.

There are different hardware system configurations for the HP 64700 Series system. For information on hardware configurations, refer to the HP 64700 Installation/Service manual.

Examples

If you are using a LAN, you can use the telnet capability with the Terminal Interface:

1 From your host computer enter the command: **telnet <emulator_name>**.

2 Now enter the command: **pv 1**

A message similar to the following should appear:

```
Testing:  HP 64748 Motorola 68020 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
HP64748 Motorola 68020 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.

When your performance verification test is complete, use the keyboard **<CTRL>d** keys to end the emulation session.

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 64747/48 Emulator consists of the following modules.

```
Test #          Function Test Name
   0      Host Tests
   1      Reset Release Tests
   2      Transition Tests
   3      Dual Port Access Tests
   4      Emulation Memory Banks Test
   5      Analysis Interface
   6      Demo Board
```

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, including interrupt lines BR, BERR, etc.

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 among the products you have installed. You can determine which versions of firmware you have by entering the terminal interface **ver** command.

There are at least three assemblies that have separate firmware in the HP 64700 Card Cage. These assemblies are:

- 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 host controller card has Flash EPROMs that can be updated with current versions of emulator and analyzer firmware.

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 *HP 64700 Series Card Cage Installation/Service manual*.

Parts List

Main Assembly **Component Part**	**New**	**Exchange**
HP 64747A Probe and Demo Board		
FG Monitor Floppy	64747-18003	
MC68030 Probe assembly	64747-62107	64747-69507
HP 64747A/B Demo Board	64747-66506	
HP 64747B Probe and Demo Board		
FG Monitor Floppy	64747-18004	
MC68030 Probe assembly	64747-66509	64747-69509
HP 64747A/B Demo Board	64747-66506	
Top Plastic Cover	64747-04101	
Bottom Plastic Cover	64747-04102	
HP 64748A/D Probe and Demo Board		
FG Monitor Floppy	64748-18002	
MC68020 Probe assembly	64748-62101	64748-69101
HP 64748 Demo Board	64748-66510	
Top Plastic Cover	64748-04101	
Bottom Plastic Cover	64748-04102	
Probe and Demo Board Generic Parts		
Washer	3050-0300	
Plastic Rivet	0361-1215	
External Power Cable	5181-0201	
Plastic Rivets Kit (rivets and washers)	64748-68700	

Main Assembly **Component Part**	**New**	**Exchange**
HP 64748B Emulation Control Card Subassembly		
Plastic Rivet	0361-1215	
Egress Panel	64748-00201	
Bracket (used with Egress Panel)	64748-01201	
Strain Relief Strap	64748-01204	
Cable-100 36"	64748-61601	
Cable-100 37"	64748-61602	
Cable-100 38"	64748-61603	
Emulation Control Card (without external cable or egress panel)	64748-66507	64748-69507
Wrist strap	9300-1405	
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 64171A 256 Kbyte SRAM Module	64171-66503	64171-69503
HP 64172A 256 Kbyte SRAM Module	64172A	64172-69501
HP 64171B 1 Mbyte SRAM Module	64171-66502	64171-69502
HP 64172B 1 Mbyte SRAM Module	64172B	64172-69502
HP 64173A 4 Mbyte SRAM Module (do not use on 64748A)	64173A	64173-69501
Analyzer Card	64740-66526	64740-69526
34-pin ribbon cable	64708-61601	

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)

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.

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.

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 MC68020 and MC68030/EC030 emulators 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 Software 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.

Foreground

the directly addressable memory range of the emulation processor.

Foreground Monitor

a monitor program that executes in the foreground address space. When the monitor exists in foreground, it is directly accessible by, and can interact with, your target program.

Guarded Memory

an address range that is to be inaccessible to the emulation processor. The emulator will generate a break and display an error message if an access to guarded memory occurs.

Handshaking

a process that involves receiving and/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.

Logical Address Space

the addresses assigned to code during the process of compiling, assembling and linking to generate absolute files. Refer to Chapter 9 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 target 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 the 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 the 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).

Pass Through Mode

see Transparent Mode.

PC Interface

a program that runs on the HP Vectra and IMB PC/AT compatible computers. This is a friendly interface used to operate an HP 64700 Series emulator.

Performance Verification

a program that tests the emulator to determine whether the emulation and analysis hardware is functioning properly.

Physical Address Space

the address space in hardware memory and hardware I/O that is accessed by the microprocessor during normal program execution. Refer to Chapter 9 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

an address at which a particular opcode causes program execution to break into the monitor. The details of the software breakpoint instruction varie among emulators. Software breakpoints are used to stop target program execution at a particular point so that you can view the state of the processor or target system, or you can begin stepping through program instructions from a known location.

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.

Transparent Configuration

the configuration in which the HP 64700 Series emulator is connected between a data terminal and a host computer. When the emulator is in the transparent (pass through) mode, the data terminal acts like a normal terminal connected to the computer. In this configuration, you can develop code on the host computer and download absolute code into the emulator for debugging and testing.

Transparent Mode

the emulator mode in which all characters received on one port will be copied to the other port. This mode allows a data terminal (connected to one emulator port) to access a host computer (connected to the other emulator port) through the emulator.

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 high level (Graphical or Softkey) Interface and return to the host computer operating system. An unlocked exit command allows you to exit the high level interface and re-enter later with the default configuration. (See also Locked Exit.) This is not available in the Terminal Interface.

User Program

Another name for your target program (the program you are developing for your product.

Viewport

see Window.

Wait States

extra microprocessor clock cycles that increase the total time of a bus cycle. Wait states are typically used when slower memory is implemented.

Window

a specified rectangular area of virtual space shown on the display in which data can be observed.

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.

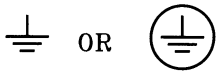
Safety Symbols Used In Manuals

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.

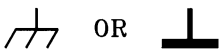
Hot Surface. This symbol means the part or surface is hot and should not be touched.

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.