



User's Guide for the Terminal Interface

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

Notice

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

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

© Copyright 1993, 1994, Hewlett-Packard Company.

This document contains proprietary information, which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another language without the prior written consent of Hewlett-Packard Company. The information contained in this document is subject to change without notice.

HP is a trademark of Hewlett-Packard Company.

UNIX is a registered trademark of UNIX System Laboratories Inc. in the U.S.A. and other countries.

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

Microtec is a registered trademark of Microtec Research, Inc.

Hewlett-Packard Company

P.O. Box 2197

1900 Garden of the Gods Road

Colorado Springs, CO 80901-2197, U.S.A.

RESTRICTED RIGHTS LEGEND. Use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in subparagraph (C) (1) (ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013. Hewlett-Packard Company, 3000 Hanover Street, Palo Alto, CA 94304 U.S.A. Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Printing History

New editions are complete revisions of the manual. Many product updates and fixes do not require manual changes, and manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual revisions.

Edition 1	64783-97000, March 1993
Edition 2	64783-97002, October 1993
Edition 3	64783-97004, January 1994

Safety and Certification and Warranty

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

The HP 64783A/B Emulator

Description

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

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

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

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

The emulators plug into the modular HP 64700 instrumentation card cage and offer 80 channels of processor bus analysis with the HP 64794A or HP 64704A emulation-bus analyzer. Flexible memory configurations are offered from zero through two megabytes of emulation memory. High performance download is achieved through the use of a LAN or RS-422 interface. An RS-232 port and a firmware-resident interface allow debugging of a target system at remote locations.

For software development the HP AxCASE environment is available on SUN SPARCsystems and HP workstations. This environment includes an ANSI standard C compiler, assembler/linker, a debugger that uses either a software simulator or the emulator for instruction execution, the HP Software Performance Analyzer that allows you to optimize your product software, and the HP Branch Validator for test suite verification.

If your software development platform is a personal computer, support is available from several third party vendors. This capability is provided through the HP 64700's ability to consume several industry standard output file formats.

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

Features

HP 64783A/B Emulator

- HP 64783A: 16 to 33 MHz active probe emulator
- HP 64783B: 20 to 40 MHz active probe emulator
- Supports MC68040, MC68EC040, and MC68LC040
- Supports burst and synchronous bus modes
- Symbolic support
- Number of breakpoints available:
 - If specified at RAM addresses: unlimited;
 - If specified at ROM addresses: eight.
- 36 inch cable and 219 mm (8.8") x 102 mm (4") probe, terminating in PGA package
- Background and foreground monitors
- Simulated I/O with workstation interfaces
- Consumes IEEE-695, HP-OMF, Motorola S-Records, and Extended Tek Hex File formats directly. (Symbols are available with IEEE-695 and HP-OMF formats.)
- Multiprocessor emulation
 - synchronous start of 32 emulation sessions
 - cross triggerable from another emulator, logic analyzer, or oscilloscope
- Demo board and self test module included

Emulation-bus analyzer

- 80-channel emulation-bus analyzer, which uses the static deMMUer of the MC68040 emulator
- Post-processed dequeued trace with symbols
- Eight events, each consisting of address, status, and data comparators
- Events may be sequenced eight levels deep and can be used for complex trigger qualification and selective store

Emulation memory

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

In This Book

This manual covers the HP 64783A/B emulator for the MC68040, MC68EC040, and MC68LC040 microprocessors. It is divided into the following parts:

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 MC68040 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

Part 1 Quick Start Guide

The Emulation Process	2
Develop Your Programs	2
Configure the Emulator	2
Use the Emulator	2
In This Part	3

1 Getting Started

The Getting Started Procedure	6
Step 1. Log in to the emulator	6
Step 2. Initialize the emulator	7
Step 3. Load the demo program	7
Step 4. See the emulation memory map	8
Step 5. Display the symbols of the demo program	9
Step 6. Display the demo program in memory	10
Step 7. Start the demo program	11
Step 8. Take a trace of the demo program	12
Step 9. View the status of the trace	12
Step 10. View the trace list	13
Step 11. Use the demo program	14
Step 12. Stop (break from) program execution	15
Step 13. Display processor registers	15
Step 14. Step through program execution	16
Step 15. Reset the emulator	17
Step 16. If the emulator status character is unfamiliar	17

Contents

The MMU demonstration	18
Step 17. Again, run the demo program from reset	19
Step 18. See the setup of the MMU	20
Step 19. Look at the translation table details for a single logical address	21
Step 20. Look at details of an MMU Table	22
Step 21. Take another trace of emulation activity	23
Step 22. Prepare the deMMUer so you can see symbolic addresses in the trace list	24
Step 23. Enable the deMMUer	25
Step 24. Take a new trace	26
Step 25. Inverse assemble the trace list	27
Step 26. See the demo help screen	28
Step 27. Reset the emulator	28

2 Troubleshooting

If the demo program won't work	30
If you don't see a prompt	30
If you see an unfamiliar prompt	31
If the emulator displays a prompt, but doesn't respond to commands	32
If you can't load the demo program	32
If you can't load a program	33
If the emulator won't run the program	33
If you can't break to the monitor	34
If the emulator won't reset	34

Part 2 Using the MC68040 Emulator/Analyzer

Making Measurements 36

In This Part 36

3 Using the Terminal Interface

Using the Interface 39

To apply power 40

To initialize the emulator 41

To enter commands 42

To recall commands 43

To repeat commands 44

To enable or disable command line editing 45

To edit a command 45

To get on-line help 46

To display the emulator status 47

To set the date and time 47

To change the prompt 48

To check the version of the Terminal Interface software 49

To print strings on the output device 49

To insert delays in command processing 50

Building and Using Macros 51

To create macros 51

To execute a macro 52

To delete macros 53

Using Command Files 54

Building Command Files 54

Editing Command Files 55

Comments in Command Files 55

To create a command file with a text editor 55

To log a command file from a PC host 56

To log a command file on a UNIX host (emulator on different port) 57

To use a command file on a PC host 58

To use a command file on a UNIX host (emulator on different port) 59

4 Using the Emulator

To configure the emulator 62

To build programs 62

Loading and Storing Programs 64

To load a program from a PC host (PC controls emulator) 64

To load programs over the LAN 65

To load a program from a UNIX host (emulator on different port) 67

Symbols 68

To load program symbols over the LAN 68

To add user symbols 70

To remove symbols 70

To display symbols 71

Accessing Processor Memory Resources 72

To display memory 72

To modify memory 73

To search memory 75

To copy memory blocks 77

To initialize display and set access modes 78

Using Processor Run Controls 79

To run a program 79

To break to monitor 80

To step the processor 81

To reset the processor 84

Viewing and Modifying Processor Registers 85

To display registers 85

To modify registers 86

Using Execution Breakpoints	88
Setting execution breakpoints in RAM	88
Setting execution breakpoints in ROM	89
Execution breakpoints in ROM when the MMUs manage memory	89
Using temporary and permanent breakpoints	90
To enable or disable the execution breakpoints feature	91
To insert an execution breakpoint	92
To enable a temporary execution breakpoint	93
To set a ROM breakpoint in RAM	94
To disable an execution breakpoint	95
To remove an execution breakpoint	96
To display execution breakpoints	96
Using the Emulator In-Circuit	97
To install the emulation probe	98
To power-on the emulator and your system	99
To probe target system sockets	99
Using MC68040 With MMU Enabled	100
To enable the processor memory management unit	100
To view the present logical-to-physical mappings	101
To see translation details for a single logical address	103
To see details of a translation table used to map a logical address	103
Using an FPU with an MC68EC040 or MC68LC040 Target System	104

5 Using the Analyzer

Making Basic Analyzer Measurements	108
To create an expression	108
To start a trace measurement	109
To stop a trace measurement	109
To display the trace status	110
To display the trace list	110
To define a simple trigger qualifier	110
To define a simple storage qualifier	111
To set the trigger position	112

Contents

Displaying the Trace List	113
To define analyzer labels	113
To delete analyzer labels	114
To display the analyzer labels	114
To change the trace format	115
To display the trace list	117
To prevent trace list header display	119
To control symbol and address display in the trace list	120
To control trace list disassembly and dequeuing	122
To obtain a time or state count in the trace list	125
Analyzing Program Execution when the MMU is Enabled	126
To program the deMMUer in a static memory system	126
To trace program execution in physical address space	127
Using the Trace Sequencer	128
To change the trace configuration	128
Using Easy Configuration	129
To create a simple expression	129
To insert a sequence term	130
To remove a sequence term	131
To reset the sequencer	131
To define a primary branch	132
To define a global restart term	133
To display the current sequencer settings	134
To specify trace start with a sequencer term other than term one active	134
Using Complex Configuration	135
To assign the trigger term	135
To reset the sequencer	136
To display the current sequencer settings	137
To define trace patterns	138
To define a range qualifier	139
To create a complex expression	140
To define a primary branch term	142
To define a secondary branch term	144
To define complex storage qualifiers	145
To prevent storage of sequencer-advance states in the trace memory	147
To specify trace start with a sequencer term other than term one active	148

Setting Analyzer Clocks 149
 To trace target/background code execution 149
 To configure the analyzer clock 150

Using Other Analyzer Features 152
 To define a prestore qualifier 152
 To count states or time 153
 To check trace signal activity 155
 To define equates 155
 To display equates 156
 To delete equates 156
 To set trace memory depth in the deep analyzer 157

6 Making Coordinated Measurements

Basic Elements of Coordinated Measurements 160
 To start a simultaneous program run on two emulators 162
 To trigger one emulation-bus analyzer from another emulation-bus analyzer 163
 To break to the monitor on an analyzer trigger signal 166
 To break the emulator to its monitor after the deep emulation-bus analyzer completes a trace 167
 To set up the deep emulation-bus analyzer so its counts are enabled by an external instrument 168

7 Configuring the Emulator

Memory 170
 Emulation Monitor 170
 Other Configuration Items 171

Mapping and Configuring Memory 172
 To assign memory map terms 172
 To assign the memory map default 177
 To check the memory map 177
 To delete memory map terms 178
 To enable one wait state 178
 To enable the memory management unit 179

Contents

To select the emulation monitor	180
To set the monitor base address	182
To interlock monitor and target system cycle termination signals	183
To set foreground monitor interrupt priority	184
To set the background monitor keep-alive address	185
To preset the interrupt stack pointer and PC	186
Setting Other Configuration Items	188
To restrict to real-time runs	188
To disable the processor cache memories	189
To disable target system interrupts	190
To enable breakpoint acknowledge cycle termination interlocking	191
Providing MMU Address Translations for the Foreground Monitor	192
Locating the Foreground Monitor using the MMU Address Translation Tables	194

8 Solving Problems

If the emulator appears to be malfunctioning	196
If the trace listing states column contains "dma long write (retry)" repeatedly	196
If the analyzer fails to trigger on a program address	197
If the analyzer triggers on a program address when it should not	197
If trace disassembly appears to be partially incorrect	198
If you see unexplained states in the trace list	198
If the analyzer won't trigger	199
If the target processor remains in a wait state	199
If you suspect that the emulator is broken	200
If you have trouble mapping memory	200
If emulation memory behavior is erratic	201
If you're having problems with DMA	201
If you're having problems with emulation reset	202
If the deMMUer runs out of resources during the loading process	202
If "out of deMMUer resources" with less than eight mappings	203
If only physical memory addresses in analyzer measurement results	203
If the deMMUer is loaded but you still get physical addresses	204
If you can't break into the monitor after you enable the MMU	205
If you see exclamation marks "!" in count columns of the trace lists	205
If you see negative time or state counts in trace lists	206

If you do not see the counter overflow indication "!" where you expected to see it in a trace list	206
If your target system loses sync when emulation breakpoints are executed	206

Part 3 Reference

Commands and Expressions	208
In This Part	208

9 Using Memory Management

Understanding Emulation And Analysis Of The Memory Management Unit	210
Terms And Conditions You Need To Understand	210
Logical vs Physical	210
What are logical addresses?	211
What are physical addresses?	211
Static and dynamic system architectures	211
Static system example	211
Non-paged dynamic system example	212
Paged dynamic system example	212
Where Is The MMU?	213
Using supervisor and user privilege modes	214
How the MMU is enabled	214
Hardware enable	214
Software enable	215
Restrictions when using the emulator with the MMU turned on	215
How the MMU affects the way you compose your emulation commands	216
Seeing details of the MMU Translations	217
How the emulator helps you see the details of the MMU mappings	217
Supervisor/user address mappings	219
Translation details for a single logical address	220
Address mapping details	220
Status information	221
Table details for a selected logical address	222

Contents

Using the DeMMUer	223
What part of the emulator needs a deMMUer?	223
What would happen if the analyzer didn't get help from the deMMUer?	223
How does the deMMUer serve the analyzer?	223
Reverse translations are made in real time	224
DeMMUer options	224
What the emulator does when it loads the deMMUer	225
Restrictions when using the deMMUer	226
Keep the deMMUer up to date	226
The target program is interrupted while the deMMUer is being loaded	226
The analyzer must be off	226
Expect strange addresses if you analyze physical memory with multiple logical mappings	226
Resource limitations	228
Example to show resource limitations	229
The Emulation Memory Map Can Help	229
Dividing the deMMUer table between user and supervisor address space	231
Solving Problems	232
Using the "mmu" command to overcome plug-in problems	232
Use the analyzer with the deMMUer to find MMU mapping problems	233
Failure caused by access to guarded memory	233
Failure due to system halt	234
Execution breakpoint problems	234
A "can't break into monitor" example	235

10 Emulator Commands

The Command Set	240
b	241
bc	242
bnct	245
Defaults	246
bp	248
cf	253
cl	257
cmb	259
cmbt	261
cp	264
dmmu	266

dt	268
dump	269
echo	271
equ	274
es	279
help,?	280
init	282
load	284
m	287
mac	291
map	295
mmu	299
mo	302
po	305
pv	306
r	309
reg	310
rep	312
rst	313
rx	314
s	316
ser	319
stty	322
sym	325
t	330
ta	331
tarm	333
tcf	335
Easy Configuration	337
Complex Configuration	339
Resetting the Analyzer Configuration	341
tck	342
tcq	346
telif	349
tf	353
tg	
356	
tgout	359
th	364
tif	366

Contents

tinit	370
tl	372
tlb	376
tp	378
tpat	381
tpq	384
trng	386
ts	389
tsck	393
tsq	395
tsto	399
tx	402
ver	404
w	405
x	407

11 Expressions

ADDRESS	411
ANALYZER_EXPR	413
COMPLEX_EXPR	415
EXPR	420
SIMPLE_EXPR	427
Easy Configuration	427

12 Emulator Error Messages

Emulator error messages	432
Analyzer Error Messages	472
##IL# in trace list Mnemonic column	484

13 Data File Formats

Binary/Hexadecimal Trace List Format	486
No Trigger Record	487
Empty Trace Record	487
New State Data Record	488
More State Data Record	490
Trace State Record	492
New Timing Data Record	493
More Timing Data Record	496
Trace Sample Records	497
Symbol Files	499
Symbol file syntax	500

14 Specifications and Characteristics

Processor Compatibility	504
Electrical	504
Motorola JTAG	504
HP 64783A/B Maximum Ratings	505
HP 64783A/B Electrical Specifications	506
HP 64783A/B Clock AC Timing Specifications	508
HP 64783A/B Output AC Timing Specifications	509
HP 64783A/B Input AC Timing Specifications	511
Physical	514
Environmental	515
BNC, labeled TRIGGER IN/OUT	515
Communications	516

Part 4 Installation and Service

In This Part 518

15 Connecting the Emulator to a Target System

Plugging The Emulator Into A Target System 520

Understanding an emulator 520

Equivalent circuits 522

Connecting the emulator to the target system 524

Verifying Operation Of The Emulator In Your Target System 526

Running the emulator configured like the processor 527

To verify operation of the target system 528

Interpreting the trace list 537

Fixing timing problems 539

Installing the emulator in a target system without known good software 540

Installing Emulator Features 542

Evaluating the reset facilities 542

Installing the background monitor 544

Resetting into the background monitor 544

Dealing with keep-alive circuitry while using the background monitor 546

Testing memory accesses with the background monitor 547

Running a program from the background monitor 548

Breaking into the background monitor 551

Exiting the background monitor 552

Software breakpoint entry into the background monitor 553

Stepping with the background monitor 555

Installing the foreground monitor 558

Resetting into the foreground monitor 559

Dealing with keep-alive circuitry by using the custom foreground monitor 561

Testing memory access with the foreground monitor 562

Running a program from the foreground monitor 563

Breaking into the foreground monitor 565

Exiting the foreground monitor 567

Software breakpoint entry into the foreground monitor 567

Stepping with the foreground monitor 570

Installing emulation memory 572

16 Installation and Service

Installation 576

Installing Hardware 577

Step 1. Install optional memory modules on Deep Analyzer card, if desired 579

Observe antistatic precautions 579

Step 2. Connect the Emulator Probe Cables 581

Step 3. Install Boards into the HP 64700 Card Cage 584

Step 4. Install emulation memory modules on emulator probe 596

Step 5. Connect the emulator probe to the demo target system 600

Step 6. Apply power to the HP 64700 602

To verify the performance of the emulator 606

What is pv doing to the Emulator? 608

Troubleshooting 608

To ensure software compatibility 609

Parts List 611

What is an Exchange Part? 611

17 Installing/Updating Emulator Firmware

Step 1: Install the firmware update utility 617

Step 2: Run "progflash" to update emulator firmware 618

Glossary

Index

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

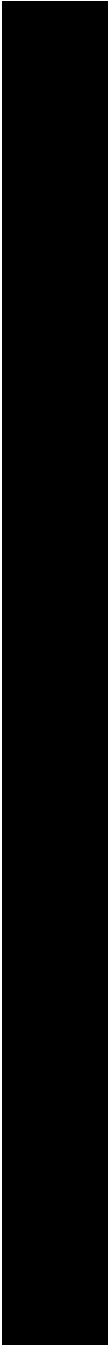
After you configure the emulator, you can load the programs you want to test, run them, and make various measurements to verify their functionality. The emulator allows you to control program runs, display and modify memory and registers, and record program execution.

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.



1



Getting Started

The Getting Started Procedure

The first steps of this procedure introduce you to some of the basic features of the MC68040 emulator. The last steps of this procedure show you how the MC68040 emulator helps you develop your target system and program code in a virtual memory system managed by the MC68040 memory management unit (MMU).

Step 1. Log in to the emulator

- Type the following command:

```
$telnet hostname
```

Where <hostname> is the name of the emulator. You could use the Internet Protocol (IP) address (or internet address) in place of the emulator name, if desired. For example:

```
$telnet 15.35.226.210
```

Note

The "telnet" capability of the HP64700A is unsupported. It is provided at no cost. Hewlett-Packard makes no warranty on its quality or fitness for a particular purpose.

You should see messages similar to:

```
Trying...  
Connected to 15.35.226.210  
Escape character is '^]'
```

After you connect to the emulator, you should see a prompt similar to:

```
R>
```

Step 2. Initialize the emulator

- Make sure you begin an emulation session with the emulator in its default, power-up state by initializing the emulator. Initialize the emulator by entering the **init** command.

```
R>init  
# Limited initialization completed
```

Step 3. Load the demo program

- Type in the **demo** command.

```
R>demo
```

The command "demo" initializes the configuration of the emulator and loads the demo program along with its symbols into emulation memory.

Step 4. See the emulation memory map

- See how the emulation memory map was set up to store the demo program. Use the **map** command:

```
R>map
# remaining number of terms      : 5
# remaining emulation memory     : 20000h bytes
map 00000000..00000fff@a   eram lock      # term 1
map 0ff00000..0ff000fff@a  eram dp,lock,tci # term 2
map 0ffff0000..0fffffff@a  eram lock      # term 3
map other tram
R>
```

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

In an emulation memory map, you can characterize memory ranges as emulation RAM, emulation ROM, target system RAM, target system ROM, or as guarded memory.

The "lock" attribute requires cycle-termination signals from the target system to end emulation-memory cycles. The "dp" attribute assigns the associated 4-Kbyte address space to the dual-port memory. Term 2 is occupied by the emulation foreground monitor, which always uses dual-port memory. The "tci" attribute prevents data read by the processor from being stored in the instruction and data caches when execution is in the emulation monitor.

Step 5. Display the symbols of the demo program

- Type in the `sym` command:

```
R>sym
sym __ctype=0000081a6
sym __initcopy=00000809e
sym _alarm=00001c004
sym _clocktic=00001c000
sym _demodisp=000008094
sym _duration=000018000
sym _main=0000100a2
sym _memcpy=0000082e0
sym _memset=0000082a8
sym _outchar=000010000
sym _setalarm=00000808a
sym _sys_demodisp=0000007c8
sym _sys_demointr=0000007e0
sym _sys_intrhdlr=000000812
sym _sys_setalarm=000000874
sym _sys_startup=0000008cc
sym _sysbuf=00001c010
sym _tolower=0000082c8
sym _userbuf=000018004
sym _usermode=000008000
sym dtoui=00000816c
sym dtoui=000008188
sym entry=000000700
R>
```

Step 6. Display the demo program in memory

- Display a partial listing of the demo program by using the following memory command:

```
U>m -dm 8000..8050
00008000 _usermode      LINK.W    A6,#$FFFC
00008004 -              MOVEM.L   D2-D4/A2-A4,-(A7)
00008008 -              MOVEA.L   ($0008,A6),A2
0000800c -              MOVE.L    ($000C,A6),D2
00008010 -              MOVEA.L   ($0010,A6),A3
00008014 -              TST.L     D2
00008016 -              BLE.L     $00008074
0000801c -              MOVEC     USP,D0
00008020 -              MOVE.L   D0,($FFFC,A6)
00008024 -              MOVE.L   D2,D0
00008026 -              LSL.L    #2,D0
00008028 -              SUB.L    D0,($FFFC,A6)
0000802c -              MOVEA.L   ($FFFC,A6),A4
00008030 -              MOVEQ    #$00000000,D3
00008032 -              BRA.B    $00008066
00008034 -              MOVEA.L   ($00,A3,D3.L*4),A0
00008038 -              MOVEQ    #$FFFFFFFF,D0
0000803a -              ADDQ.L   #1,D0
0000803c -              TST.B    (A0)+
0000803e -              BNE.B    $0000803A
00008040 -              ADDQ.L   #1,D0
00008042 -              MOVE.L   D0,D4
00008044 -              MOVEQ    #$00000002,D0
00008046 -              MOVE.L   D4,D1
00008048 -              DIVSL.L D0,D0:D1
0000804c -              ADD.L    D4,D0
0000804e -              SUB.L    D0,($FFFC,A6)
M>
```

The **m** command lets you display and modify memory locations. When displaying memory, the **-dm** option causes the contents of memory locations to be disassembled (displayed in assembly language mnemonic format).

Note that the prompt changed from **R>** to **M>**. This indicates that the emulation processor changed from the reset state to the state of running in the monitor. The emulator uses one of the routines in the monitor to read the content of memory.

Step 7. Start the demo program

- Type in the command to run the emulator from reset.

```
M>r rst  
U>
```

The "U>" prompt indicates that the user program (in this case, the demo program) is running.

Step 8. Take a trace of the demo program

- Use the trace to see where the demo program is running:

```
U>t
  Emulation trace started
```

The **t** (trace) command tells the analyzer to look at activity on the emulation processor buses and control signals at each bus cycle. The information available during a bus cycle is called a state.

When a state matches the "trigger state", which you specify, the analyzer captures it into its trace memory and identifies it as line 0. The analyzer fills the remainder of its trace memory with states that occur after the trigger state.

The default trigger state is "any state". With the default specification, the **t** command will cause the analyzer to trigger on the first state it finds and store each new state that occurs until its trace memory is filled.

Step 9. View the status of the trace

- Type in the **ts** command:

```
U>ts
--- Emulation Trace Status ---
NEW User trace complete
Arm ignored
Trigger in memory
Arm to trigger ?
States 1024 (1024) 0..1023
Sequence term 2
Occurrence left 1
U>
```

Step 10. View the trace list

- List the first twenty states stored in the trace list (-t 20), with instructions disassembled (-d), and symbols and addresses shown in the addr column (-e), by entering the following **tl** command:

```
U>tl -d -e -t 20
  Line  addr,H      68040 Mnemonic
-----
  0     000008f8  BLE.B      $000008E4
      =000008fa  MOVEQ     #$00000050,D0
  1     000008fc  CMP.L      D2,D0
      =000008fe  BLS.B      $000008E4
  2     000008e8  TST.B      ($00,A2,D2.L)
  3     000008ec  BEQ.B      $000008F6
      =000008ee  ADDQ.L    #1,D2
  4     000008f0  MOVEQ     #$00000050,D0
      =000008f2  CMP.L      D0,D2
  5     fffffc010  $00----- phy sdata byte read
  6     000008f4  BCS.B      $000008E8
      =000008f6  TST.L      D2
  7     000008f0  MOVEQ     #$00000050,D0
      =000008f2  CMP.L      D0,D2
  8     000008f4  BCS.B      $000008E8
      =000008f6  TST.L      D2
  9     000008f8  BLE.B      $000008E4
      =000008fa  MOVEQ     #$00000050,D0
 10     000008fc  CMP.L      D2,D0
      =000008fe  BLS.B      $000008E4
 11     00000900  MOVE.L    #$0000092A,($FFF8,A6)
 12     00000904  $092AFF8  phy sprog long read
 13     000008e0  Unimplemented F-Line Opcode: $FF00
      =000008e2  ADDQ.L    #4,A7
 14     000008e4  MOVEQ     #$00000000,D2
      =000008e6  BRA.B      $000008F0
 15     000008e8  TST.B      ($00,A2,D2.L)
 16     000008f0  MOVEQ     #$00000050,D0
      =000008f2  CMP.L      D0,D2
 17     000008f4  BCS.B      $000008E8
      =000008f6  TST.L      D2
 18     000008f8  BLE.B      $000008E4
      =000008fa  MOVEQ     #$00000050,D0
 19     000008fc  CMP.L      D2,D0
      =000008fe  BLS.B      $000008E4
U>
```

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

The second column contains the address information for each state. Addresses may be locations of instruction opcodes on fetch cycles, or they may be sources or destinations of operand cycles. The **-e** option in the **tl** command causes symbols to be shown in this column (when available) instead of hexadecimal values.

Step 11. Use the demo program

The third column shows mnemonic information about the emulation bus cycle. The **-d** option in the **tl** command causes instructions to be disassembled.

Note the equals signs "=" in the **addr** column. These are equivalent addresses. These addresses never appeared on the emulation bus. These addresses were emitted by the disassembler to show actual locations of the associated instructions.

Step 11. Use the demo program

- The demo program is designed to write numbers to the seven-segment LED on the demo board. Have the following numbers written to the LED: 1234567890ABCDEF. Use the following command to modify the "_sysbuf" symbol:

```
U>m _sysbuf="1234567890ABCDEF"
```

The string you entered in the "m" command should write one time on the LED of the demo board.

Step 12. Stop (break from) program execution

The **b** command causes the emulator to break away from running the target program to run the emulation monitor program.

The monitor program contains the routines that provide most of the features of the emulator, such as display of internal registers and target system resources. The MC68040 emulator has both a background monitor and a foreground monitor (the foreground monitor is used during this demo procedure).

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

- Break emulator execution away from the demo program and begin execution in the monitor by entering the **b** command:

```
U>b  
M>
```

Notice that the emulation status character becomes "M", which indicates that the emulator is executing in the monitor.

Step 13. Display processor registers

- Type in the **reg** command:

```
M>reg  
reg pc=0000094e st=2009  
reg d0=ffffffdfff d1=fffffff d2=00000000 d3=00000000  
reg d4=00000000 d5=00000000 d6=00000000 d7=00000000  
reg a0=00000002 a1=00018068 a2=0001c010 a3=00000000  
reg a4=00000000 a5=00000000 a6=0001fff6 a7=0001ffee  
reg usp=0001bffe msp=00000001 isp=0001ffee vbr=00000000  
reg cacr=00000000 sfc=00 dfc=04  
M>
```

Step 14. Step through program execution

Each time the **reg** command is used, the new values of the processor registers are shown. In the case where the emulator is running your target program, some of the processor registers will usually have changed each time you enter the **reg** command. In this case (the emulator is running the monitor), the register values are not changing.

Step 14. Step through program execution

- Step one instruction in the target program by entering the **s** command.

```
M>s
0000008e4@s -                MOVEQ    #$00000000,D2
PC = 0000008e6@s
!STATUS 155! Vector table modified for single stepping
M>
```

- Step eight instructions in the target program by entering the **s 8** command

```
M>s 8
0000008e6@s -                BRA.B    $000008F0
0000008f0@s -                MOVEQ    #$00000050,D0
0000008f2@s -                CMP.L   D0,D2
0000008f4@s -                BCS.B   $000008E8
0000008e8@s -                TST.B   ($00,A2,D2.L)
0000008ec@s -                BEQ.B   $000008F6
0000008f6@s -                TST.L   D2
0000008f8@s -                BLE.B   $000008E4
PC = 0000008e4@s
M>
```

The **s** command lets you step through execution of your target program. You can step single instructions, or several instructions at a time.

Step 15. Reset the emulator

- Reset the emulator by entering the `rst` command.

```
M>rst  
R>
```

Notice that the emulation status character changes to "R", which shows that the emulator is being held in the reset state.

Step 16. If the emulator status character is unfamiliar

The "R", "U", and "M" emulation prompt status characters are described in this chapter. If you see other emulation status characters, enter the `es` command for more information about the emulator status.

- Display the emulator status information by entering the `es` command.

```
R>es  
M68040--Emulation reset
```

The MMU demonstration

The remainder of this demonstration shows how the M68040 emulator helps you develop and analyze your target program within a memory system that is managed by the MMU of the MC68040 processor.

The demo program simulates a real target system that displays hexadecimal characters on the seven-segment display of the HP64783A demo board (used with HP 64783A and HP 64783B emulators). A simple operating system uses interrupts to maintain a system clock and to provide an alarm capability for user mode tasks. The MMU of the 68040 is used for translating addresses for separate ROM and RAM areas and to provide memory access protections.

The demo program requires 128 Kbytes of emulation memory. One 64-Kbyte block of emulation RAM is mapped to lower memory to correspond to system ROM. The other 64-Kbyte block is mapped to upper memory to correspond to system RAM. The operating system boot code sets up the MMU with the following address translations and access protections:

Logical Address	Physical Address	Attributes
00000000..00007fff	00000000..00007fff@a	S W (32K super prog/tables)
00008000..0000ffff	00008000..0000ffff@a	W (32K lib/libc prog)
00010000..00017fff	ffff0000..ffff7fff@a	W (32K user prog)
00010000..0001bfff	ffff8000..ffffBfff@a	(16K user data/stack)
0001c000..0001ffff	ffffC000..ffffffffff@a	S (16K super data/stack)
80000000..80ffffff	80000000..80ffffff@a	TT (monitor)

Where:

S = Supervisor access only.

W = Write protected.

TT = Controlled by a transparent translation register.

ROM space is write protected and is divided into two blocks. The first half contains privileged operating system code and static tables that are only accessible in supervisor mode. The second half contains shared library functions accessible in both supervisor and user mode. RAM space is divided into three blocks and is initialized by the operating system at bootup. The first half is write protected and contains target program code. The second half is divided into two blocks for user data/stack space and supervisor data/stack space, respectively.

To run the demo program, run from reset and modify "_sysbuf" to a NUL-terminated hexascii string (ie: **m -db _sysbuf="0123456789ABCDEF"**). The operating system monitors the "_sysbuf" variable and invokes the user mode task to display each character on the seven-segment display. The user mode task makes calls to the operating system interface to access the I/O port for the seven-segment display and to set up an alarm to display each character for one-half

Step 17. Again, run the demo program from reset

second. After all characters in the string have been displayed, control returns to the operating system, at which time, "_sysbuf" can be modified again. If the hexascii string ends with "@", the user mode task will display the hexascii string repetitively and will not return to the operating system. In this case, the emulation processor must be reset to stop the repetition.



Step 17. Again, run the demo program from reset

- Type in the command:

```
M>r rst
U>
```

Step 18. See the setup of the MMU

- The **mmu** command lets you see the present setup of the MMU. The MMU was set up by the demo program when you first started it. Type in the **mmu** command:

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

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

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

No part of the demo program was placed in the 16-Megabyte range controlled by the transparent translation register. Transparent translation registers are easy to use when reserving 1:1 address space for the emulation monitor, which is essential because the emulator must be able to access the monitor whether or not the MMU is enabled.

The demo program could have created an appropriate entry in the MMU tables to reserve memory space for the emulation monitor. The transparent translation registers were used to store the monitor because they are much easier to use. Simply define the address space to be translated 1:1 by the transparent translation registers and load the emulation monitor into it.

Step 19. Look at the translation table details for a single logical address**Step 19. Look at the translation table details for a single logical address**

- To see how logical address 18000h is translated by the MMU to its corresponding physical address, type the following command: **mmu -t 18000**.

```

U>mmu -t 18000
Logical Address (hex)      0    0    0    1    8    0    0    0
Logical Address (bin)    0000 0000 0000 0001 1000 0000 0000 0000
Table Level              AAAA AAAB BBBB BBCC CCCC PPPP PPPP PPPP

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

Physical Address (hex) = ffff8000
U>

```

Logical address 18000h is translated to physical address ffff8000h by the MC68040 MMU, using the MMU tables.

When you are developing a virtual memory system, you will need to check the translations of selected addresses. The **-t** option of the **mmu** command lets you do this.

Step 20. Look at details of an MMU Table

- Type the following command to see the details of Table C where it is used to translate logical address 18000h: **mmu -t -c 18000**.

The **mmu** command with the appropriate options lets you see a listing of the details of one of the MMU tables where it is used in translating a selected logical address.

```
U>mmu -t -c 18000
Logical Address (hex)      0 0 0 1 8 0 0 0
Logical Address (bin)    0000 0000 0000 0001 1000 0000 0000 0000
Table Level              AAAA AAAB BBBB BBCC CCCC PPPP PPPP PPPP

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

Occasionally you will need to examine the content of one of the MMU translation tables at the point where it is used to translate a particular logical address. The **mmu -t -c** command and options let you do this.

Step 21. Take another trace of emulation activity

- Use the **t** and **tl** commands to take a new trace as follows:

```
U>t
  Emulation trace started
U>tl
  Line   addr,H   68040 Mnemonic
-----
    0  0000099c  $4A826FEA phy sprog long read
    1  00000998  $B48065F2 phy sprog long read
    2  0000099c  $4A826FEA phy sprog long read
    3  000009a0  $7050B082 phy sprog long read
    4  000009a4  $63E42F0A phy sprog long read
    5  000009a8  $48790001 phy sprog long read
    6  000009ac  $00964EB9 phy sprog long read
    7  00000988  $588F7400 phy sprog long read
    8  0000098c  $60084A32 phy sprog long read
    9  00000990  $28006708 phy sprog long read
U>
```

Note that all of the addresses displayed are physical addresses (denoted by "phy" in the "68040 Mnemonic" column of the trace list).

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

To see logical addresses in the tracelist, you must use the deMMUer, demonstrated next in this procedure.

Step 22. Prepare the deMMUer so you can see symbolic addresses in the trace list

- Load the deMMUer so it can supply logical address information to the analyzer (derived by reverse translating the physical addresses on the emulation bus). Type the command: **dmmu -lv**.

```
U>dmmu -lv
All physical addresses within the following 32-Mbyte range(s) will be
reverse translated into logical addresses for the analyzer:
00000000..001ffffff@a
0fe00000..0fffffff@a

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

The above command loaded the deMMUer with information to reverse translate two ranges of physical addresses obtained from the MMU. The verbose "v" mode of this display was selected so we could see which ranges of physical addresses would be reverse translated by the deMMUer.

Note the message below the list of physical address ranges that will be reverse translated. This tells you that any physical addresses that might have been derived from two or more logical addresses will be reverse translated to the lowest logical address by the deMMUer.

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

```
U>mmu
Logical Address      Physical Address      Attributes
00000000..000007fff 00000000..000007fff@a S W
00000800..00000ffff 00000800..00000ffff@a W
00001000..000017fff 0ffff0000..0ffff7fff@a W
00001800..00001bfff 0ffff8000..0ffffbfff@a
00001c00..00001ffff 0ffffc000..0fffffff@a S
0ff000000..0fffffff 0ff000000..0fffffff@a TT
U>
```

Physical address 1ffff, for example, might appear when the MMU translates either logical address 1ffff or logical address fffff. The deMMUer will send 1ffff to the analyzer because it is the lowest logical address that might have caused physical address fffff to appear on the emulation bus.

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



Step 23. Enable the deMMUer

- Enable the deMMUer to supply logical addresses (obtained by reverse translating physical addresses) to the analyzer. Type the command: **dmmu -e**.

```
U>dmmu -e
```

Step 24. Take a new trace

- The purpose of this trace is to see if the analyzer is now capturing logical address information for each state on the emulation bus. Type in the **t** and **tl** commands shown below:

```
U>t
Emulation trace started
U>tl -e
  Line   addr,H   68040 Mnemonic
-----
   0    000008fc $B08263E4 log sprog long read
   1    000008e8 $4A322800 log sprog long read
   2    000008ec $67085282 log sprog long read
   3    000008f0 $7050B480 log sprog long read
   4    _sysbuf  $00----- log sdata byte read
   5    000008f4 $65F24A82 log sprog long read
   6    000008f0 $7050B480 log sprog long read
   7    000008f4 $65F24A82 log sprog long read
   8    000008f8 $6FEA7050 log sprog long read
   9    000008fc $B08263E4 log sprog long read
  10    00000900 $2D7C0000 log sprog long read
  11    00000904 $092AFF8  log sprog long read
  12    000008e0 $FF00588F log sprog long read
  13    000008e4 $74006008 log sprog long read
  14    000008e8 $4A322800 log sprog long read
  15    000008f0 $7050B480 log sprog long read
  16    000008f4 $65F24A82 log sprog long read
  17    000008f8 $6FEA7050 log sprog long read
  18    000008fc $B08263E4 log sprog long read
  19    000008e8 $4A322800 log sprog long read
U>
```

Note that "log" is now shown in the "68040 Mnemonic" column. Because the deMMUer is supplying logical addresses to the analyzer, the analyzer is able to replace the address for "_sysbuf" with the symbol name in the trace list. You told the analyzer to use symbol names where possible when you included the **-e** option with the **tl** command.

Step 25. Inverse assemble the trace list

- Now show the trace list inverse assembled into assembly language mnemonics. Use the following **tl** command and the options shown:

```
U>tl -d -e -t 20
  Line  addr,H  68040 Mnemonic
-----
  0  000008fc  CMP.L    D2,D0
    =000008fe  BLS.B    $000008E4
  1  000008e8  TST.B    ($00,A2,D2.L)
  2  000008ec  BEQ.B    $000008F6
    =000008ee  ADDQ.L   #1,D2
  3  000008f0  MOVEQ    #$00000050,D0
    =000008f2  CMP.L    D0,D2
  4  _sysbuf   $00----- log sdata byte read
  5  000008f4  BCS.B    $000008E8
    =000008f6  TST.L    D2
  6  000008f0  MOVEQ    #$00000050,D0
    =000008f2  CMP.L    D0,D2
  7  000008f4  BCS.B    $000008E8
    =000008f6  TST.L    D2
  8  000008f8  BLE.B    $000008E4
    =000008fa  MOVEQ    #$00000050,D0
  9  000008fc  CMP.L    D2,D0
    =000008fe  BLS.B    $000008E4
 10  00000900  MOVE.L   #$0000092A,($FFF8,A6)
 11  00000904  $092AFF8 log sprog long read
 12  000008e0  Unimplemented F-Line Opcode: $FF00
    =000008e2  ADDQ.L   #4,A7
 13  000008e4  MOVEQ    #$00000000,D2
    =000008e6  BRA.B    $000008F0
 14  000008e8  TST.B    ($00,A2,D2.L)
 15  000008f0  MOVEQ    #$00000050,D0
    =000008f2  CMP.L    D0,D2
 16  000008f4  BCS.B    $000008E8
    =000008f6  TST.L    D2
 17  000008f8  BLE.B    $000008E4
    =000008fa  MOVEQ    #$00000050,D0
 18  000008fc  CMP.L    D2,D0
    =000008fe  BLS.B    $000008E4
 19  000008e8  TST.B    ($00,A2,D2.L)
U>
```

Note that the symbol "_sysbuf" is shown in the trace list instead of the hexadecimal address value that it represents. You requested that symbols be shown in place of hexadecimal address values when you included the "-e" option to the **tl** command above.

Step 26. See the demo help screen

- Enter the command `? demo`

The display contains a description of the demo program used in this chapter. Other help screens are available in the emulator. Simply enter `? or help`, followed by the name of the command or topic for which you would like to receive some help. If you enter `? or help` alone, the emulator will show you a list of topics on which you can receive helpful information.

Step 27. Reset the emulator

- Sometimes you may want to reset the emulation processor. This may be done from the emulator or the target system. To reset the emulation processor from the emulator, type: `rst`

The prompt will change to `R>`.

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



Troubleshooting

Finding out what's wrong and fixing it

Chapter 2: Troubleshooting

If the demo program won't work

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 reset flying lead is connected from the probe to the demo board.
- Make sure the green power indicator LED on the demo board is on.
- 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 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 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 state, or has not arbitrated the bus away from the emulation processor. The demo board will not cause these conditions.
- Make sure that the emulator is properly configured for your system requirements. See Chapter 7, “Configuring the Emulator.” The demo command automatically configures the emulator for the demo board.
- 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 monaddr=<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. The demo program requires use of the foreground monitor.
- 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.

- Make sure that you haven't reassigned port actions with the datacomm switch settings.

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 monaddr=<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. 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.

Chapter 2: Troubleshooting
If you can't break to the monitor

- 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 monaddr=<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.
- Make sure the reset flying lead is connected from the probe to the demo board.
- 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 MC68040 Emulator/Analyzer

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 record 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

Chapter 3: Using 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 not been <u>g</u> rant <u>ed</u> the bus by the external arbiter (<u>B</u> G is not asserted).
h>	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 a cycle termination from the target system.

Chapter 3: Using the Terminal Interface

Using the Interface

?>

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 MC68040 emulator and do a complete initialization. You will see a display similar to the following 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
```

```
HP64783 Motorola 68040 Emulator
HP64740 Emulation Analyzer
R>
```

The emulator executes the 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><b>demo
```

To enter a run command, enter:

```
R><b>r
```

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

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

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

```
R><b>es;ts
```

To display various analyzer settings, enter:

```
R><b>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:

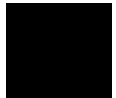
```
R>demo;rep 3 {s 1;reg}
```

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.
/ <code><string></code>	Find previous command matching <code><string></code> .
n	Fetch previous command matching <code><string></code> .
N	Fetch next command matching <code><string></code> .

- 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

Chapter 3: Using the Terminal Interface

Using the Interface

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 MC68040 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"
```

Chapter 3: Using the Terminal Interface

Using the Interface

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
```

```
R>r rst
```

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;t1;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 `_sysbuf` variable:

```
R>mac setup={rst;demo;r rst;m -db _sysbuf="1234"}
```

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 rst               # runs program
# Now give the program a command
m _sysbuf="1234"    # display 1234
# execute a trace
t                  # start a trace
tl -e              # list trace with symbols and addresses
```

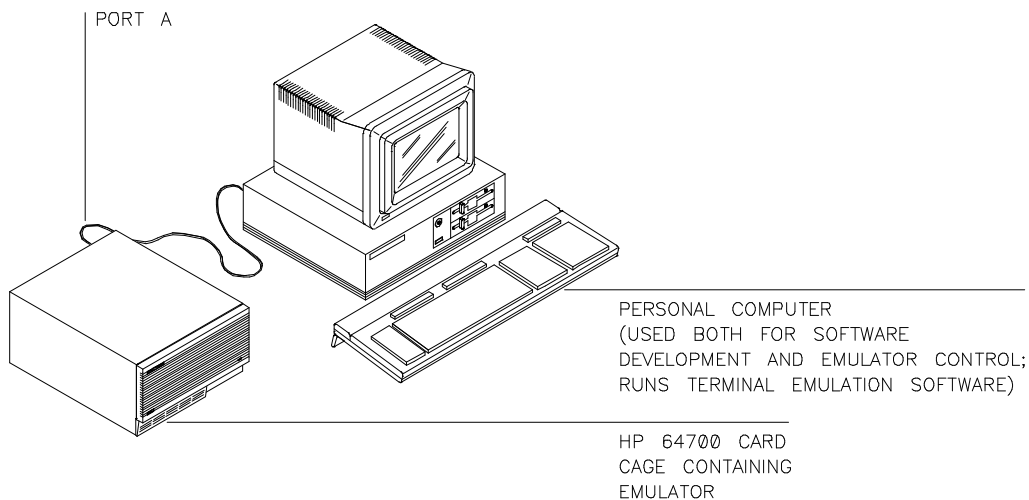
Chapter 3: Using the Terminal Interface Using Command Files

```
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.

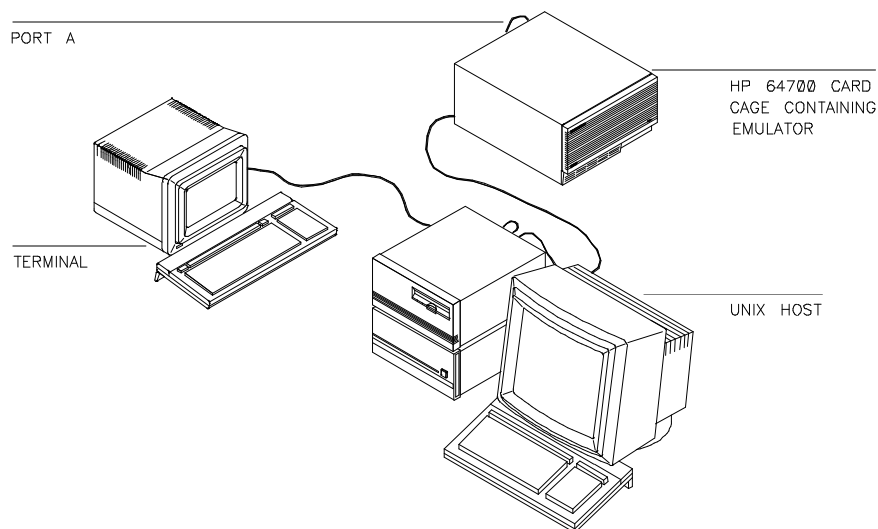
The above procedure allows you to save a series of Terminal Interface commands in a file that you can later edit into a form suitable for use as a command file.



To simplify editing the file, use the **po** command to change the standard output device while logging the commands. Only the prompts and commands will be logged to the file. The emulator responses will be sent to the standard output device.

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: `~.`



Chapter 3: Using the Terminal Interface

Using Command Files

- 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. Also, you can reassign the emulator's standard output (with **po -o <port>** to eliminate the emulator command responses (memory and trace displays and the like) that would clutter the command file. However, if you want to log the responses to configuration commands that you enter without parameters (for example, **cf** and **tsq**), don't reassign the standard output port.

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>~%>cf file
```

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

```
R>~%>
```

Edit the command file **cf file** 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
```

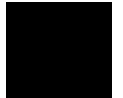
Download the command file named **cfile**:

```
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 control the processor and view system resources

Chapter 4: Using the Emulator

To configure the emulator

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 MC68040 assembly language using a text editor.
- 2 Translate the “C” source files to relocatable object code using a compatible C cross compiler.

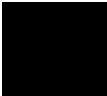
Translate the assembly source files to relocatable object files using a compatible MC68040 cross assembler.

- 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
68040	HP B1463	HP B1465

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.)



Loading and Storing Programs

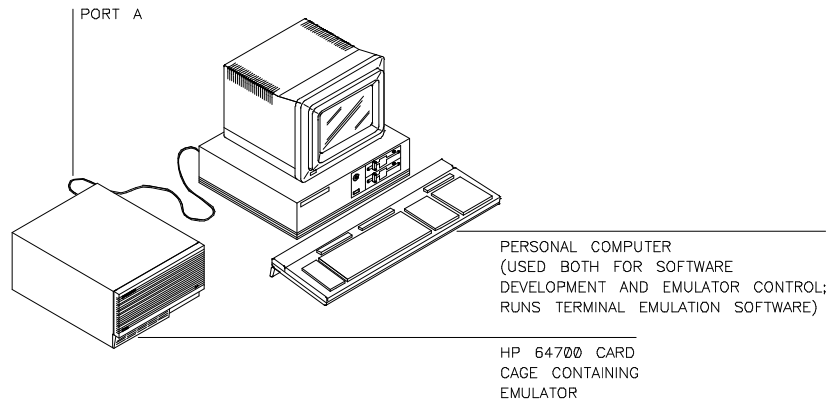
The Terminal Interface provides commands that allow you to move files into emulation or target memory from a host computer through the serial ports of the HP 64700 Card Cage. You can also save a range of memory in an absolute file for later reuse. (You might do this if you patch a section of code and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

Many different absolute file formats are supported. The primary one discussed in this section is Motorola S-record format. If you have a host computer with the HP 64000 **transfer** utility, 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 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 performed as follows:

Chapter 4: Using the Emulator

To load programs over the LAN

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
$
```

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).

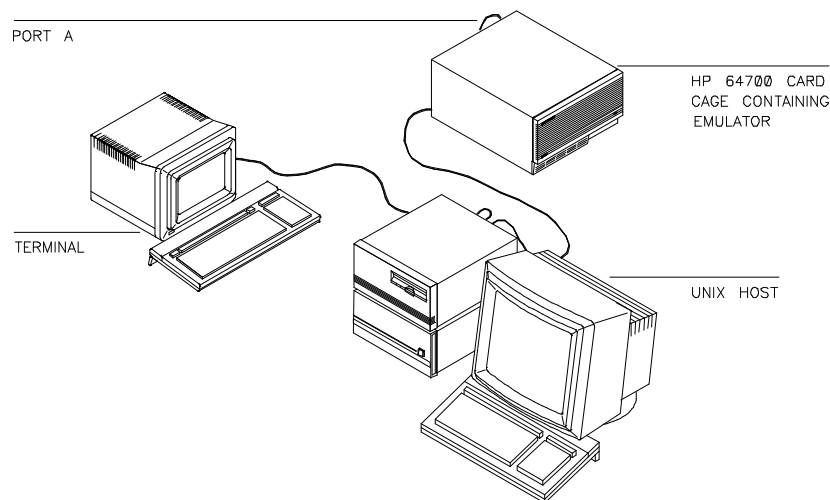
2 To connect to the emulator, type the command `cu -l /dev/ttyXX` at the HP-UX prompt .

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.



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 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 symbols for the demo program, enter:

```
R>>demo  
R>sym
```

To display the value of all global symbols in a program that has both local and 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 MC68040 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 without interrupting it. Dual-port memory may also be displayed while the emulator is reset. For other memory, the emulator must use the monitor to access it.

Examples

Before using the following examples, reload the demo program:

```
R>demo
```

To display program memory for the part of the demo program from the setalarm address to the demodisp address in the current display mode, enter:

```
R>m _setalarm.._demodisp
```

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

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

To display a portion of the demo program's sysbuf variable in byte format, enter:

```
R>m -db _sysbuf
```

To display the usermode portion of the demo program in mnemonic format, enter:

```
R>m -dm _usermode.._demodisp
```

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.

Chapter 4: Using the Emulator

To modify memory

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 `_sysbuf` variable of the demo program to 1234 ascii, enter:

```
R>m -db _sysbuf="1234"
```

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 MC68040 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.

Chapter 4: Using the Emulator

To search memory

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 a message area in a program for the string "ommand," enter:

```
R>ser -db <beginning symbol>..<ending symbol>="ommand"
```

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

```
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 eram # Emulation RAM (1K block for
exception vector table)
```

Copy the exception table from target ROM to emulation RAM:

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

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

```
M>reg 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, **l** for long word, or **x** for letting the emulator select the optimum access size. The default is **x**.
- 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.

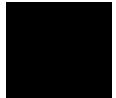
If set to **x**, the size you include in your "display memory" or "modify memory" command will be used for the access. It will temporarily override the **x** designation for the access. If set to **b**, **w**, or **l**, the size selected in your "display memory" or "modify memory" commands will have no effect on the actual memory access; it will be what you specified for the memory access size.

If you choose **b**, **w**, or **l** for your access size, only the selected size will be used. In cases where access is made to misaligned addresses or memory content having an insufficient number of bytes, the emulator will perform read-modify-write transactions to complete the access using only the size you specified.

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 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 the `@u` or `@s` function code to specify the privilege level for the run command.

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.

Chapter 4: Using the Emulator

To break to monitor

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 use the values found at memory locations 0 and 4.

Examples

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

```
R>r rst
```

or to run a program from a known address, such as 400h, enter:

```
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.

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

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.

Chapter 4: Using the Emulator

To step the processor

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.

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

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

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

The background monitor only requires that the trace exception vector be an even value and point to readable memory. This allows the processor to complete trace exception processing, including initial prefetches from the trace exception handler, during transition into the background monitor. After reading the trace exception vector, the emulator attempts to read from the address it points to. If the read succeeds, the emulator single steps without modifying the vector table. Otherwise, the emulator attempts to write the current value of VBR into the trace exception vector (because the vector table is readable).

There are some limitations when single stepping. A step may fail when single stepping an instruction that changes the address of the vector table (modifies the

VBR register). With the background monitor, instructions that can be interrupted (ie: floating-point operations) may not complete because the emulator generates an interrupt after a finite amount of time after the single step is initiated.

Examples

To step the processor one instruction, enter:

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 `_usermode` symbol in the demo program, enter:

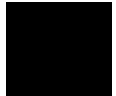
M>s 5 _usermode

To step once and disable step display, enter:

M>s -q

To step twice and display only the resulting program counter value, enter:

M>s -w 2



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.

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

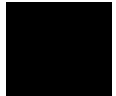
If the emulator is running a user program when the target system reset occurs, it will behave as if a **r rst** command were issued.

If the MC68040 emulator is in the monitor when the target reset occurs, it will reenter the monitor when the reset is released. (As if a **rst -m** command were given.)

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>**

Register Class	Register Names
* (basic)	pc, st, usp, isp, msp, cacr,d0..d7, a0..a7, vbr, dfc, sfc
fpu	fpcr, fpsr, fpiar, fp0..fp7
mmu	dt0, dt1, itt0, itt1, mmusr, tc, srp, urp

- where **<reg_class_name>** is the name of a class of registers. The available registers and register classes are shown in the following table:

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**

Chapter 4: Using the Emulator

To modify registers

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.

The mmu register class of the MC68EC040 is different from the mmu register class of the MC68040 and MC68LC040. The MC68EC040 uses registers `dacr0/iacr0`, `dacr1/iacr1`, which are nearly identical to `dt0/itt0`, `dt1/itt1`. These MC68EC040 registers are displayed in the `dt0/itt0`, `dt1/itt1` registers, respectively

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.

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 `_usermode` address of the demo program, enter:

```
M>reg pc=8000
```

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:

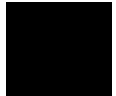
```
M>reg d3=0 a6=80a5
```

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

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

To modify the itt0 register of the MC68040 to 00000110, enter:

```
M>reg itt0=110
```



Using Execution Breakpoints

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

Execution breakpoints are implemented using the BKPT instruction of the MC68040. Before you use execution breakpoints, you must enable the execution breakpoints feature (with the **bc -e bp** command). Then you can insert, enable, disable, or remove execution breakpoints.

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

Setting execution breakpoints in RAM

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

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

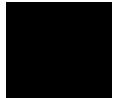
The emulator allows an unlimited number of breakpoints to be set in RAM.

Setting execution breakpoints in ROM

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

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

To determine if an active breakpoint uses one of the eight hardware resources, display the address in memory. Breakpoints implemented in software will show a BKPT #7 instruction at the breakpoint address. Breakpoints implemented using one of the eight hardware resources will show the original instruction at the breakpoint address.



Execution breakpoints in ROM when the MMUs manage memory

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

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

Using temporary and permanent breakpoints

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

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

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

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

To enable or disable the execution breakpoints feature

- To enable a type of break condition, type: **bc -e <cond>**
- To disable a type of break condition, type: **bc -d <cond>**
- To check the states of the break conditions, type: **bc**
- Where **<cond>** can be any of the following: **rom**, **bp**, **bnct**, **cmbt**, **trig1**, and **trig2**. All break conditions are enabled when the emulator is initialized.

You can choose to enable or disable individual types of breakpoints. For example, you might want to temporarily disable breaks on writes to ROM. (Perhaps you need to see the next few bus cycles after the write to ROM in the trace list.) Or you might want to enable a break when the analyzer finds its trigger condition.

Note that **bc -e rom** will not protect data at addresses mapped as ROM. Emulation memory or RAM memory in the target system will be changed by processor writes even if that memory has been mapped as ROM.

Examples

To enable execution breakpoints, enter:

```
R>bc -e bp
```

To disable execution breakpoints and breaks on writes to ROM:

```
R>bc -d bp rom
```

To generate an analyzer output trigger signal when the analyzer finds its trigger event, and cause the emulator to break to the monitor when the trigger is generated:

```
R>tgout trig1  
R>bc -e trig1
```

To insert an execution breakpoint

- Insert an execution breakpoint at a location given by `<address>` by typing:
bp <address>

When you set an execution breakpoint, the emulator uses the monitor to insert the breakpoint instruction. Therefore, you can't enable a breakpoint when the emulator is reset. Use the **b** command to begin running in the monitor.

Examples

To insert a breakpoint at address 42a, enter:

```
M>bp 42a
```

To insert a permanent breakpoint at the symbol `_sys_intrhdlr`, enter:

```
M>bp -p _sys_intrhdlr
```

To insert a temporary breakpoint at `_main`, enter:

```
M>bp -t _main
```

To enable a temporary execution breakpoint

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

When a temporary breakpoint is executed, it is disabled. If you want to reenable the temporary breakpoint, 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 will be marked “enabled”. If the breakpoint is in RAM, the BKPT instruction will be written to memory at that location.

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

Examples

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

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

To enable existing breakpoints at `_sys_intrhdlr` and `_main`, enter:

```
M>bp -e _sys_intrhdlr _main
```

To set a ROM breakpoint in RAM

- To have the emulator treat a particular breakpoint address as if it were in ROM hardware, **type: bp -h <address>**

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

Examples

Force a hardware breakpoint to be executed on address 1000h:

```
M>bp -h 1000h
```

Force a temporary hardware breakpoint to be executed at address 2000h:

```
M>bp -t -h 2000h
```

To disable an execution breakpoint

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

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

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

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

Examples

To disable an existing execution breakpoint at `_sys_intrhdlr`, enter:

```
M>bp -d _sys_intrhdlr
```

To disable existing breakpoints at `_sys_intrhdlr` and `_main`, enter:

```
M>bp -d _sys_intrhdlr _main
```

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

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

To remove an execution breakpoint

- To remove an existing execution breakpoint at a location given by **<address>**, type: **bp -r <address>**
- To remove all existing execution 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. In RAM, the original instruction is restored to memory, and the breakpoint table entry is removed.

The emulator uses the monitor to remove the breakpoint.

Examples

To remove an existing execution breakpoint at `_sys_intrhdlr`, enter:

```
M>bp -r _sys_intrhdlr
```

To remove existing breakpoints at `_sys_intrhdlr` and `_main`, enter:

```
M>bp -r _sys_intrhdlr _main
```

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

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

To display execution breakpoints

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

Using the Emulator In-Circuit

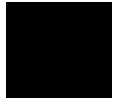
Out-of-circuit emulation is useful for debugging your program code. You can use the emulation-bus analyzer and other emulator features to test and evaluate your code.

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

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

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

- 1 Study the chapter titled "Connecting the Emulator to a Target System" later in this manual. It discusses things you need to know to successfully connect the emulator to a target system and overcome problems you may encounter.
- 2 Study your system design, especially its memory configuration.
- 3 Study the chapter titled "Configuring the Emulator" in this manual to determine how to set configuration items for the best results with your target system.
- 4 Install the emulation probe in your target system by following the results of step 1.
- 5 Set configuration items as determined by the results of step 3.



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.

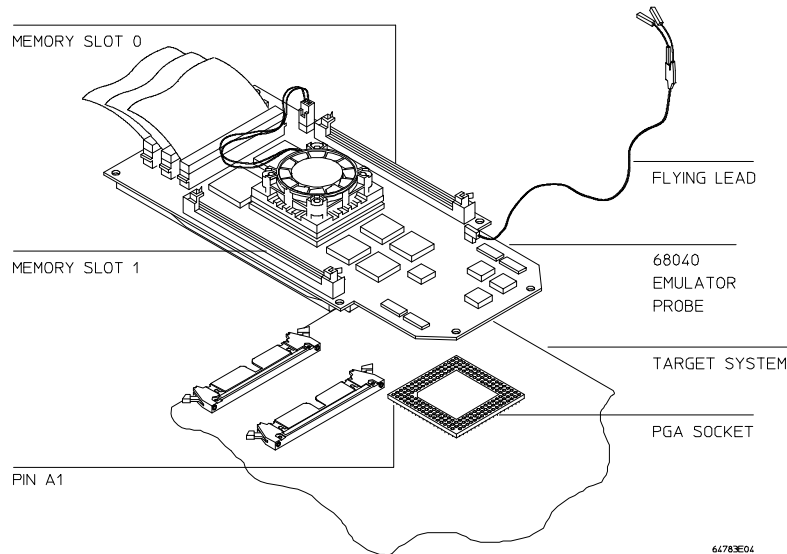
Caution

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 target system sockets

- A flexible adapter is available from Hewlett-Packard for special target system probing needs. It is listed in the following table:

Probe type	HP part number
68040 PGA to PGA flexible adapter	E3429A

Using MC68040 With MMU Enabled

When you enable memory management in the MC68040 emulator, many capabilities and features become available that are not otherwise offered. Also, some of the features of the emulator behave differently. The following paragraphs will help you when you are using the MC68040 emulator with the MMU enabled. Chapter 9 will help you use the MC68040 MMU efficiently.

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

To enable the processor memory management unit

- To turn on the MMU in the MC68040 emulation processor, enter: **cf mmu=en**

Once enabled, the MMU of the MC68040 can be set up by the operating system to manage logical (virtual) memory in physical address space. The selection of a root pointer and the value in the translation control register determine how the MMU of the MC68040 will manage memory. The MMU of the MC68040 must be enabled by this configuration question before the operating system can establish those control values.

The target system may control the MMU during program execution by using the MDIS signal. The target system can disable the MMU even if it is enabled via the configuration question.

You must use a foreground monitor when the MMU is enabled. If the background monitor is selected when you type **cf mmu=en**, the emulator will give you an error message telling you to select a foreground monitor.

Note

Make sure the foreground monitor is mapped to memory space that has a 1:1 translation and is not write protected. Refer to Chapter 7 for instructions on how to map the foreground monitor to 1:1 address space in the MMU.

Examples

To enable the MC68040 MMU so that the operating system can set it up to manage memory, enter the command:

M>cf mmu=en

To disable the MC68040 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.

Chapter 4: Using the Emulator

To view the present logical-to-physical mappings

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

```
U>mmu 0..0fff
```

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

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 0fff, enter the command:

```
U>mmu 0..0fff@s
```

To see only the mappings under user space in the address range from 0 through 0fff, enter the command:

```
U>mmu 0..0fff@u
```

Note that the valid address spaces are **u** and **s**.

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

```
U>mmu tc=0C000 srp=80006000 urp=80002000
```

To see translation details for a single logical address

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

Examples

To see how logical address 40F0 (in supervisor 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 in user space is mapped through the translation tables, enter the command:

```
U>mmu -t 1000@u
```

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

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

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

Table a may be accessed at several base addresses, depending on which logical address is being translated. This command ensures you see Table a where you want to see it.

Examples

To see the details of Table a used to map logical address 1250, enter the command:

```
M>mmu -ta 1250
```

Using an FPU with an MC68EC040 or MC68LC040 Target System

The MC68EC040 and MC68LC040 processors do not have an on-chip FPU. When floating-point functionality is required, all floating-point operations must be implemented in software using integer instructions. Language systems usually provide a floating-point software library for this purpose.

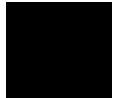
The HP 64783A/B emulator uses an MC68040 processor with an on-chip FPU. Because there is no way to disable the FPU, floating-point operations may execute differently, depending on the language system used. If your language system generates calls directly to the floating-point software library and does not emit any opcodes for floating-point instructions, then there should be no difference in floating-point operations whether you are using the emulator or the MC68EC040/LC040 processor plugged into your target system.

If your language system emits opcodes for floating-point instructions and relies on an F-Line exception handler to call the floating-point software library when the instruction is executed, then your target system will operate differently when the emulator is plugged in. When using the emulator, most floating-point instructions will be executed on the FPU in hardware instead of generating an F-Line exception and allowing the floating-point operations to be implemented in software. For this scenario, the following three points should be taken into consideration:

- Floating-point software libraries cannot be tested while the emulator is plugged in. Floating-point instructions are always executed on-chip, not by your floating-point libraries. This will definitely cause a problem for anyone trying to develop floating-point software libraries.
- Target programs containing FPU instructions will run faster when the emulator is plugged into the target system because they are executed in the hardware of the MC68040 instead of by the floating-point software libraries, as they will be when the MC68EC040/LC040 processor is plugged in. This will cause performance measurements to show much better results when using the emulator than you will actually obtain when you use the MC68EC040/LC040 processor.
- If you are unaware that your language tools use floating-point instructions (and you do not actively provide floating-point libraries and F-Line exception

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

handling), you may find that your target system does not work when you unplug the emulator and plug in your MC68EC040/LC040 target processor.





5



Using the Analyzer

How to view program execution in real-time

Making Basic Analyzer Measurements

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.

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**

You can selectively display portions of the buffer using the **tl** command.

If you are using the deep analyzer, the depth of the trace list buffer depends on whether or not you installed memory modules on the analyzer card, and the capacity of the memory modules installed. Refer to Chapter 16, "Installation and Service", for details. If you are using the 1K analyzer, the trace list buffer is 512 or 1024 states deep (depending on whether or not you turn on the state/time count).

To define a simple trigger qualifier

- Define a simple trigger on an address value, by typing: **tg addr=<value>**
- Define a simple trigger when a particular type of cycle is executing, by typing: **tg status=<execution status>**

Many times, you will want to specify a trigger for the analyzer (define a reference state) within the trace. You may want to start the trace when the trigger location is

reached. You may want to end the trace when the trigger state occurs so that you can see activity leading up to the trigger event.

You can use either a simple address expression, or one that includes symbols. You can specify data events and/or bus status types. Refer to the **equ** page in the commands chapter to see all of the status equates that are predefined for use in trigger specifications that include status types.

Example

To trigger the analyzer when a program performs a write cycle at address 1000h.

```
M>tg addr=1000h and stat=write
```

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

To see only accesses to the system clock counter in the demo program:

```
M>demo  
M>tsto addr=_clocktic  
M>t  
M>r rst  
M>t1
```

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. Labels can be used in trace specifications or you can use the **tf** command to add those columns to the trace list. See “To change the trace format.”

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
```

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
```


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>demo  
R>t  
R>r
```

Display the trace starting at state 11 by entering:

```
U>tl 11
```

Chapter 5: Using the Analyzer
To display the trace list

You will see:

Line	addr,H	68040 Mnemonic
11	00000714	\$0298FFFF log sprog long read
12	00000718	\$FFFB51C8 log sprog long read
13	00000708	\$00064E7B log sprog long read
14	0000070c	\$000441F8 log sprog long read
15	00000710	\$06407007 log sprog long read
16	00000714	\$0298FFFF log sprog long read
17	00000718	\$FFFB51C8 log sprog long read
18	0000071c	\$FFF841F8 log sprog long read
19	00000720	\$02004E7B log sprog long read
20	00000640	\$FFFF0007 log sdata long read

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

U>**tl -n 3**

You will see:

Line	addr,H	68040 Mnemonic
21	00000724	\$88074E7B log sprog long read
22	00000640	\$FFFF0003 log sdata long write
23	00000710	\$06407007 log sprog long read

Display the trace from the top by entering the command:

U>**tl -t**

You will see:

Line	addr,H	68040 Mnemonic
0	00000000	\$FFFFFFFE log sdata long read
1	00000004	\$00000700 log sdata long read
2	00000700	\$203CFF00 log sprog long read

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	68040 Mnemonic
27	00000720	\$02004E7B log sprog long read
28	00000644	\$FFFF1007 log sdata long read
29	00000724	\$88074E7B log sprog long read
30	00000644	\$FFFF1003 log sdata long write
31	00000710	\$06407007 log sprog long read
32	00000714	\$0298FFFF log sprog long read
33	00000718	\$FFFB51C8 log sprog long read
34	0000071c	\$FFF841F8 log sprog long read
35	00000720	\$02004E7B log sprog long read

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

You will see:

```
0 00000000 $FFFFFFFE log sdata long read
1 00000004 $00000700 log sdata long read
2 00000700 $203CFF00 log sprog long read
3 00000704 $C0004E7B log sprog long read
4 00000708 $00064E7B log sprog long read
5 0000070c $000441F8 log sprog long read
```

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>demo
R>t
R>r rst
```

Display only symbols:

```
U>tl -s 160..170
```


Chapter 5: Using the Analyzer
To control symbol and address display in the trace list

You will see:

Line	addr,H	68040 Mnemonic
160		\$0000---- phy sdata word write
161		\$4CDF040C phy sprog long read
162		\$----0000 phy sdata word write
163		\$4E5E4E75 phy sprog long read
164		\$0000---- phy sdata word write
165	dtoI	\$F200B000 phy sprog long read
166		\$----0000 phy sdata word write
167		\$0195---- phy sdata word write
168		\$----0000 phy sdata word read
169		\$0195---- phy sdata word read
170		\$246EFFFC phy sprog long read

Display only hexadecimal values for addresses:

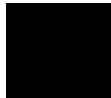
M>t1 -a 160..170

You will see:

Line	addr,H	68040 Mnemonic
160	ffffffe4	\$0000---- phy sdata word write
161	00008164	\$4CDF040C phy sprog long read
162	ffffffde	\$----0000 phy sdata word write
163	00008168	\$4E5E4E75 phy sprog long read
164	ffffffe0	\$0000---- phy sdata word write
165	0000816c	\$F200B000 phy sprog long read
166	ffffffea	\$----0000 phy sdata word write
167	fffffec	\$0195---- phy sdata word write
168	ffffffea	\$----0000 phy sdata word read
169	fffffec	\$0195---- phy sdata word read
170	000080c0	\$246EFFFC phy sprog long read

Display both symbols and hexadecimal values for addresses:

M>t1 -e 160..170



To control trace list disassembly and dequeuing

You will see:

Line	addr,H	68040 Mnemonic
160	ffffffe4	\$0000---- phy sdata word write
161	00008164	\$4CDF040C phy sprog long read
162	ffffffde	\$----0000 phy sdata word write
163	00008168	\$4E5E4E75 phy sprog long read
164	ffffffe0	\$0000---- phy sdata word write
165	dtoi	\$F200B000 phy sprog long read
166	ffffffea	\$----0000 phy sdata word write
167	ffffffec	\$0195---- phy sdata word write
168	ffffffea	\$----0000 phy sdata word read
169	ffffffec	\$0195---- phy sdata word read
170	000080c0	\$246EFFF C phy sprog long read

To control trace list disassembly and dequeuing

- 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.

Chapter 5: Using the Analyzer
To control trace list disassembly and dequeuing

The MC68040 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>demo
R>t
R>r rst
```

Display disassembled:

```
U>t1 -d 160..170
```

Chapter 5: Using the Analyzer
To control trace list disassembly and dequeuing

You will see:

Line	addr,H	68040 Mnemonic
160	ffffffe4	\$0000---- phy sdata word write
161	00008164	MOVEM.L (A7)+,D2-D3/A2
162	ffffffde	\$----0000 phy sdata word write
163	00008168	UNLK A6
	=0000816a	RTS
164	ffffffe0	\$0000---- phy sdata word write
165	dt0i	FMOVE.L FPCR,D0
166	ffffffea	\$----0000 phy sdata word write
167	fffffec	\$0195---- phy sdata word write
168	ffffffea	\$----0000 phy sdata word read
169	fffffec	\$0195---- phy sdata word read
170	000080c0	MOVEA.L (\$FFFC,A6),A2

The default is all cycles displayed (-oa).

Display only instructions:

U>t1 -oi 160..170

Line	addr,H	68040 Mnemonic
161	00008164	MOVEM.L (A7)+,D2-D3/A2
163	00008168	UNLK A6
	=0000816a	RTS
165	dt0i	FMOVE.L FPCR,D0
170	000080c0	MOVEA.L (\$FFFC,A6),A2

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

U>t1 -oda 165..175

Line	addr,H	68040 Mnemonic
165	dt0i	FMOVE.L FPCR,D0
166	ffffffea	\$----0000 phy sdata word write
167	fffffec	\$0195---- phy sdata word write
168	ffffffea	\$----0000 phy sdata word read
169	fffffec	\$0195---- phy sdata word read
170	000080c0	MOVEA.L (\$FFFC,A6),A2
	=ffffffea	src sdata read: \$00000930
171	000080c4	CMPI.B #\$53,(A2)
172	000080c8	BEQ.B \$000080D0 (branch taken)
173	ffffffea	\$----0000 phy sdata word write
175	fffffec	\$0930---- phy sdata word write

?branch 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

To obtain a time or state count in the trace list

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. The "branch taken" notation would have been shown beside the branch if the dequeuer had determined that the branch was taken. The "branch not taken" notation would have been shown if the dequeuer had definitely determined that the branch was not taken.

To obtain a time or state count in the trace list

- You can have the analyzer show you a count of time between each of the states in the trace list by typing: **tcq time**.
- You can have the analyzer show you a count of the occurrences of a selected state in the trace list by typing: **tcq <EXPRESSION>**.
- You can turn off the analyzer's count of time or states by typing: **tcq none**

If you are using the deep analyzer, counts of states or counts of time can be made at full clock speeds of the system under test. No tradeoff of analyzer state memory resources are required when the deep analyzer makes state or time counts.

If you are using the 1K analyzer, counts of states or counts of time are only available with reduced clock speeds of the system under test. Refer to the explanation of the **tck** command to understand how the clock speed affects the ability of the 1K analyzer to count.

The 1K analyzer's state/time counter uses half of the analyzer's state memory resources. Therefore, when you require a time count (**tcq time**) or state count (**tcq <EXPRESSION>**, etc.) during the trace, the 1K analyzer's trace depth is reduced to 512 states. You can obtain the full the trace depth (1024 states) by disabling state/time counting.

Analyzing Program Execution when the MMU is Enabled

Most emulation and analysis commands that require an address as part of the command use logical addresses. When the MC68040 MMU is enabled, physical addresses are placed on the emulation bus. The physical addresses may not be the same as the logical addresses. The deMMUer reverse translates the physical addresses back to logical addresses and supplies these to the analyzer so that the analyzer can:

- accept commands expressed in source file symbols.
- display trace lists with addresses expressed in source file symbols.
- display appropriate portions of source code preceding lists of trace data.

Refer to the chapter titled "Using Memory Management" 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 trace program execution in physical address space

To pick the place to load the deMMUer, you might set an execution breakpoint in your code at a point where you are sure your MMU will be set up to translate the address space you want to analyze. After the breakpoint has executed (emulator running in foreground monitor), you can load the deMMUer.

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

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

Address ranges will be reverse translated correctly if they are translated by the setup of the MMU that existed when you issued the **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.

Note that you can add a reverse-translation range of addresses to the present content of the deMMUer by entering a command, such as:
dmmu -t <logical address> <physical_address>

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 **init**), the analyzer is reset to easy configuration.

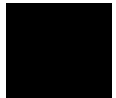
Change the trace configuration to complex if you need to specify a trigger sequence that includes more than four sequence states with multiway branches, or if you need different storage qualifiers at each level of the sequence. Change the trace configuration to easy if you need only one storage qualifier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

Chapter 5: Using the Analyzer

To insert a sequence term

Example

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

```
addr=_sys_demodisp
data=41
addr=_sys_demointr and stat=read
addr!=_sys_demointr or stat!=ack
```

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!=_sys_demointr 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 renumbered.

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.

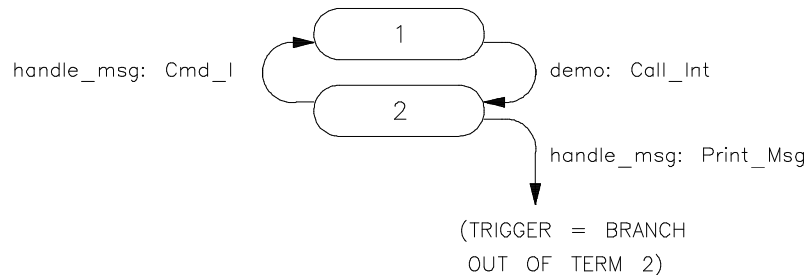
- To display the primary branch qualifier for a term given by `<TERM#>`, type: **tif**
`<TERM#>`
- To display all primary branch qualifiers, type: **tif**

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 would require three sequence terms.

Example

Suppose you want to trace the sequence of code from `Call_Int` through `Print_Msg` in a program that sends messages in response to commands it receives. Normally, a command that causes the message handler to call a routine called `Cmd_I` is found, and you don't want the analyzer to store data if `Cmd_I` is found. You only want the analyzer to capture a trace if some other symbol (other than `Cmd_I`) causes the `Print_Msg` routine to be called.

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



Set up the measurement:

```
R>tif 1 addr=Call_Int
R>tif 2 addr=Print_Msg
R>telif addr=Cmd_I
R>tp -b 20
R>t
R>r
U>t1
```

The symbol Call_Int will be at the top of the trace list because each event that causes the sequencer to advance to the next stage will be captured in trace memory. The trace will show up to 20 states of activity that occurred before the trigger state, then the trigger, and then the activity that occurred after the trigger was recognized.

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=Call_Int  
tif 2 addr=Print_Msg  
tsto all  
telif addr=Cmd_I
```

To specify trace start with a sequencer term other than term one active

- Specify any sequence term to be the first active term at trace start by typing: **tsq -init <TERM#>** (trace sequencer initial term = <TERM#>)

Where <TERM#> is the sequence term you want to be the first active term at trace start.

This feature is only available in the deep analyzer (HP 64794). There may be times when you want an elaborate sequencer setup to begin with a term other than term 1 as the first active term.

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.

Chapter 5: Using the Analyzer

To reset the sequencer

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 rst
U>t1 -5..5
```

You will see:

Line	addr,H	68040 Mnemonic	seq
-5			
-4	00000000	\$FFFFFFFE log sdata long read	+
-3	00000004	\$00000700 log sdata long read	+
-2	00000700	\$203CFF00 log sprog long read	+
-1	00000704	\$C0004E7B log sprog long read	+
0	00000708	\$00064E7B log sprog long read	+
1	0000070c	\$000441F8 log sprog long read	+
2	00000710	\$06407007 log sprog long read	+
3	00000714	\$0298FFFF log sprog long read	.
4	00000708	\$00064E7B log sprog long read	.
5	0000070c	\$000441F8 log sprog long read	.

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:

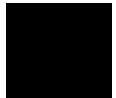
```
M>tsq -r
```

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 you are testing a program with three locations that should never be reached under conditions you are testing. You want to trace and trigger on any access to any of the three locations (called location1, location2, and location3). Enter the commands:

```
R>tcf -c
R>tpat p1 addr=location1
R>tpat p2 addr=location2
R>tpat p3 addr=location3
R>tg p1|p2|p3
R>t
```

Now you can run your tests. If there is ever an access to location1, location2, or location3 during your tests, the analyzer will capture the access and all activity related to the access so you can examine the details of the access.

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 a trace on any read cycle from a range of addresses in a message-storage area (called `Msg_A` through `End_Msgs`). You want the analyzer to store only the message read cycles. Enter the following commands:

```
R>tcf -c
R>trng addr=Msg_A..End_Msgs
R>tpat p5 stat=read
R>tg r and p5
R>tsto r and p5
R>t
R>r
```

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`

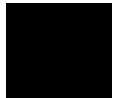
The last expression is invalid because you can't repeat different sets to extend the expression.

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.

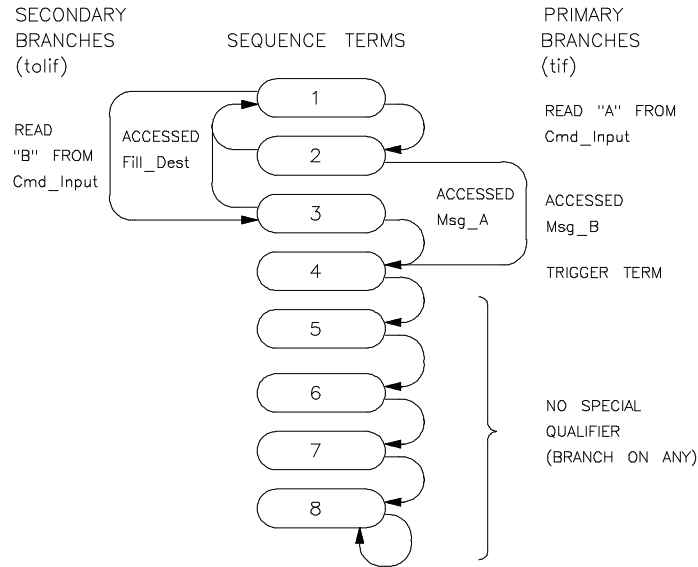
Example

Suppose the following intermittent problem occurs in a program you are developing: the message for command A is sometimes output when command B is entered, and vice versa. The state diagram shows how the sequencer can be set up to trigger a trace on the first occurrence of this intermittent problem.

If a state is found that is a command input of "A", the sequencer transitions to term 2. If a state is found that is a command input of "B", the sequencer transitions to term 3. If any other state is found, the sequencer simply waits for the next state.

If sequence term 2 is active (command A was recognized by sequence term 1), then sequence term 2 waits for an access to the message area. If message B appears, term 2 transitions to term 4, triggering a trace. Sequence term 3 makes the same sort of test for command B and message A.

Chapter 5: Using the Analyzer To define a primary branch term



Here is how the analyzer can be set up to match this state diagram:

```

R>tcf -c
R>tp e
R>tsq -t 4
R>tpat p1 addr=Cmd_Input and stat=read
R>tlb byte3 56..63
R>tpat p5 byte 3
R>tpat p6 byte3
R>tpat p2 addr=Msg_B
R>tpat p3 addr=Msg_A
R>tpat p4 addr=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

```

To define a secondary branch term

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.

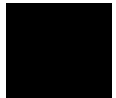
Since the secondary branch qualifier is unique for each sequence term, it is more flexible than the global restart qualifier in easy configuration. You can use it as a global restart by making all secondary branch qualifiers identical, and having them restart the trace sequence, or you can use the secondary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

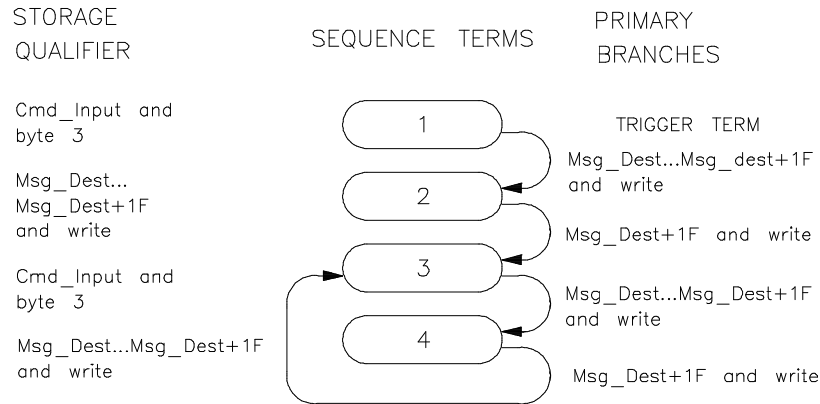
In complex mode, there are eight storage qualifiers, one for each sequencer term. This allows you to store only the states of interest at each level of the sequence, which uses the trace memory more efficiently and makes the trace display easier to read.

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



Chapter 5: Using the Analyzer
To define complex storage qualifiers

Example Suppose you wanted to capture a trace that shows only the ASCII data writes to a message destination address range, and the commands that caused them to occur. A state diagram for the analyzer might look like the following:



You might implement the above diagram by using the following commands:

```
R>tcf -c
R>tpat p1 addr=Cmd_Input
R>tpat p2 addr=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 message destination 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 message destination 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

To prevent storage of sequencer-advance states in the trace memory

activity (a “+” is shown in the **seq** column in the trace list when the sequencer changes states).

To prevent storage of sequencer-advance states in the trace memory

- Prevent storage of the states that cause the sequencer to advance from one sequence state to another by typing: **tsq -inc dis** (trace sequencer "include" disabled).
- Restore the default specification that causes the analyzer to store all sequencer-advance states by typing: **tsq -inc en** (trace sequencer "include" enabled).

This feature is only available in the deep analyzer (HP 64794).

The default setup causes the deep analyzer to store all states that advance the sequencer from one sequence state to another. These sequence-advance states are stored regardless of whether they are qualified by your store-qualifier specification (**tsto**) or not. There may be times when you are using an elaborate sequence to identify one or two trace lines that you want to capture in memory. This selection allows you to prevent storage of the series of sequence-advance states.

To specify trace start with a sequencer term other than term one active

- Specify any sequence term to be the first active term at trace start by typing: **tsq -init <TERM#>** (trace sequencer initial term = <TERM#>)

Where <TERM#> is the sequence term you want to be the first active term at trace start.

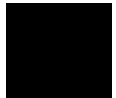
This feature is only available in the deep analyzer (HP 64794).

There may be times when you want an elaborate sequencer setup to begin with a term other than term 1 as the first active term.

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 64783 emulator generates the L clock to drive the emulation-bus 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 qualify capture of either target program execution (which includes execution of the foreground monitor), or execution of the background monitor. You also use this command to specify the maximum data rate that the analyzer will see, which affects the state/time counter.



To trace target/background code execution

- To trace only target program (user) code execution, type: **tck -u**
- To trace only background monitor code execution, type: **tck -b**
- To trace both target and background code execution, type: **tck -ub**

The emulation-bus analyzer has built-in qualifiers that allow you to select whether the analyzer captures execution of the target program (which includes execution of the foreground monitor), background monitor code, or both. Usually, you'll want to trace only your target program. If you're trying to solve a problem with emulator and target system interaction, you may want to trace both the target program and background monitor code.

To configure the analyzer clock

- Ignore the procedure on this page and the next page if you are using the deep analyzer (HP 64794). If you are using the 1K analyzer, you must configure the analyzer clock as described on this page and the next.
- To set the analyzer for a slow data rate (less than or equal to 16 MHz), type: **tck -s S**
- To set the analyzer for a fast data rate (between 16 and 20 MHz), type: **tck -s F**
- To set the analyzer for a very fast data rate (greater than 20 MHz), type: **tck -s VF**

The MC68040 analyzer clock is set to **tck -s VF** by default. The analyzer can capture all types of bus cycles correctly up to the maximum clock rate of 40 MHz, but cannot correctly count states or time at higher speeds for certain bus cycle types.

The worst-case situation is one where a zero-wait state burst cycle is performed. The analyzer clock rate for burst cycles is given by the equation:

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

To determine the correct setting for the **tck -s** command in the MC68040 emulator, calculate the maximum data rate by using the above equation. Remember that the emulator requires one wait state for all accesses when the external clock is greater than or equal to 25 MHz. (See the chapter titled “Configuring the Emulator” for more information.) Then choose the data rate option according to the data rate.

If no burst cycles are performed, the analyzer clock speed can be set slow (**tck -sS**).

Chapter 5: Using the Analyzer
To configure the analyzer clock

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

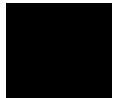
Analyzer clock rate	tck setting	Valid tcq options
clock ≤ 16 MHz	tck -s S	tcq <state> tcq time
clock ≤ 20 MHz	tck -s F	tcq <state>
clock ≥ 20 MHz	tck -s VF	tcq none

Example

Suppose you are running the MC68040 processor at 40 MHz. You have set **cf wait=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 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>`

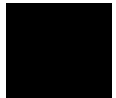
You use the prestore qualifier to save states that are related to other routines that you're tracing. For example, you might be tracing a subprogram, and want to see which program called it. You specify a set of possible callers as a prestore specification. If you don't know the set of possible callers, you can specify that the last read cycle be prestore before entry into the subroutine.

Example

Suppose you want to trigger a trace on a call to `Print_Msg` and prestore the command that called `Print_Msg`. The following example assumes you know the only possible callers to `Print_Msg` are `Cmd_A`, `Cmd_B`, and `Cmd_I`.

Enter the commands:

```
R>>tcf -c
R>&trng addr=Mst_Dest..Msg_Dest+1f
R>tpat p1 addr=Print_Msg
R>tpat p5 addr=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
```



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

Chapter 5: Using the Analyzer

To count states or time

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 MC68040 emulator defaults to **tcq never**. See “To configure the analyzer clock.”

Example

Suppose you want to count the number of write transactions to a message destination area in a program. Enter the commands:

```
R>tcf -c
R>trng addr=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 destination area.

Enter the commands:

```
R>t
R>r
```

These commands start a trace, and then run the program and provide input.

When you display the trace list, the count will show the total number of writes to the message destination area.

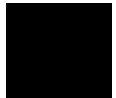
To check trace signal activity

- 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.

The trace signal display is not available in the deep analyzer (HP 64794). In the 1K analyzer, you can display trace signal activity as described above.



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
```

Chapter 5: Using the Analyzer

To display equates

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).

To set trace memory depth in the deep analyzer

- Set the depth of the analyzer trace memory by typing **tcf -deep <DEPTH>** (trace configure deep <DEPTH>).

<DEPTH> is the portion of the trace memory that the deep analyzer will use to store captured data during a trace. Unspecified depth will be ignored.

<DEPTH> can be any depth from 1 state to the maximum number of states that can be stored in the trace memory of your deep analyzer.

Maximum trace memory depth can be obtained by omitting the <DEPTH> parameter from your command, or by specifying <DEPTH> as 0.

The memory depth you specify will affect the analyzer response to a **w -m** (wait for measurement complete) command. The analyzer detects measurement complete after the trigger has been captured and the trace memory depth you specify has been filled.

Examples

To obtain a 4K memory depth in your deep analyzer, enter:

```
tcf -deep 4096
```

To obtain the full memory depth in your deep analyzer, enter:

```
tcf -deep
```

or

```
tcf -deep 0
```

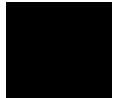
or

```
tcf -deep <full memory depth available in your analyzer>
```

Entering **tcf -deep** or **tcf -deep 0** allows you to achieve maximum trace memory depth even if you are unsure of exactly how much depth is available on the analyzer card.



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 MC68040 processor and another processor. You use HP 64700 Series emulators to replace both target system processors, and connect the emulators using the CMB. You can run a program simultaneously on both emulators, or you can start a trace on one emulation analyzer when the other emulator reaches a certain program address. These measurements are possible with the CMB.

Three signal lines are used to control interaction over the CMB.

TRIGGER The CMB TRIGGER line is low true. This signal can be driven or received by any HP 64700 connected to the CMB. This signal can be used to trigger an analyzer. It can be used as a break source for the emulator.

READY The CMB READY line is high true. It is an open-collector circuit and performs an ANDing of the ready state of enabled emulators on the CMB. Each emulator on the CMB releases this line when it is ready to run. This line goes true when all enabled emulators are ready to run, providing for a synchronized start.

When CMB is enabled, each emulator is required to break to background when CMB READY goes false, and will wait for CMB READY to go true before returning to the run state. When an enabled emulator breaks, it will drive the CMB READY false and will hold it false until it is ready to resume running. When an emulator is reset, it also drives CMB READY false.

EXECUTE The CMB EXECUTE line is low true. Any HP 64700 on the CMB can drive this line. It serves as a global interrupt and is processed by both the emulator and the analyzer. This signal causes an emulator to run from a specified address when CMB READY returns true.

Chapter 6: Making Coordinated Measurements

Basic Elements of Coordinated Measurements

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
bnc	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 MC68040 emulator supports the use of the CMB when configured with either a background or foreground monitor.

To start a simultaneous program run on two emulators

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 from another emulation-bus analyzer

- 1 Connect a CMB cable (coordinated measurement bus) between the two instrumentation card cages.
- 2 In the first analyzer, specify the analyzer trigger by typing: **tg <state>** (trigger on <state>)

where <state> is a unique state to be recognized by the analyzer (such as, **addr=1000 and data=44 and stat=write**)
- 3 Set the first analyzer to drive trig1 when it captures a state that meets the trigger specification by typing: **tgout trig1 trigger** or **tgout trig1** (trigger output on trig1 when trigger specification satisfied). Note that **trigger** is the default. It is offered on the deep analyzer, but not on the 1K analyzer.
- 4 Set the second analyzer to trigger its trace when it receives trig1 from the first analyzer by typing: **tarm =trig1; tg arm** (trace arm signal is on trig1, trigger when arm switches to true)
- 5 Establish interaction through the rear panel CMB connection on each card cage, as follows: In the interface of the first analyzer, set the rear panel CMB connection to receive trigger by typing: **cmbt -r trig1** (CMB trigger line receives trig1). In the interface of the second analyzer, set the rear panel CMB connection to drive trigger to the second analyzer by typing: **cmbt -d trig1** (CMB trigger line drives trig1).
- 6 Make sure the first (driving) analyzer is not driving trig1 by issuing the **th** (trace halt) command.
- 7 Start the second (receiving) analyzer using the **t** (trace) command.
- 8 Start the first (driving) analyzer using the **t** (trace) command.

With this coordination, you can effectively widen your analysis bus to any number of channels, limited only by the number of analyzers available in your system.

Chapter 6: Making Coordinated Measurements

To trigger one emulation-bus analyzer from another emulation-bus analyzer

Trig1 and trig2 are used to coordinate measurements between instruments in the instrumentation card cage. An analyzer can drive or receive either or both of these lines. Also, the rear-panel BNC and the CMB trigger signal can drive or receive either of these signals.

When you use trig1 and/or trig2 to coordinate actions in associated equipment, there is delay in the coordination. In the example above, several states may be executed between the time the first analyzer recognizes its trigger and the time the second analyzer recognizes its trigger.

The 1K analyzer can be set up to generate a trigger output pulse on trig1 and/or trig2 when the analyzer recognizes its trigger. The deep analyzer can be set up to generate a trigger output pulse on trig1 and/or trig2 on any of the following events:

- Recognition of the analyzer trigger (**tgout trig1 trigger**).
- Completion of a trace, the trigger captured and trace memory filled (**tgout trig1, trig2 complete**).
- Capture of a state that is <DELAY> number of states after the trigger was captured (**tgout trig2 -t 20**).
- Capture of a state that occurs after the trigger state and is <DELAY> number of states before the end of trace memory (**tgout trig1, trig2 -c 10**).
- Recognition of a specific state (**tgout trig1 addr=100**).
- Recognition of a pattern when a particular sequence term is active in the complex mode (**tgout 1 p1**)

Example

Assume you have two MC68040 emulators, running out-of-circuit. The demo program is loaded in each emulator. The following example will trigger the analyzers in both emulators when `_sys_demodisp` 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=_sys_demodisp
R>tgout trig1
R>cmbt -r trig1
R>tp c
R>t
R>b
M>
```

To trigger one emulation-bus analyzer from another emulation-bus analyzer

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:

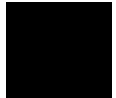
```
U>m -db _sysbuf="ABC"
```

Display the trace on the first emulator:

```
U>t1 -e -10..10
```

Display the trace on the second emulator:

```
U>t1 -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 the chapter titled “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 the chapter titled “Emulator Commands.”

To break the emulator to its monitor after the deep emulation-bus analyzer completes a trace**To break the emulator to its monitor after the deep emulation-bus analyzer completes a trace**

- 1 Set the analyzer to drive trig1 when the analyzer completes its trace (captures trigger plus enough states to fill its trace memory) by typing: **tgout trig1 complete** (trigger output on trig1 when measurement complete).
- 2 Set the emulator to break to its monitor program on receipt of the trig1 signal by typing: **bc -e trig1** (break condition enabled on trig1).

This feature is available on the deep analyzer, but not on the 1K analyzer.

You may use this setup to stop your program at some point in its execution so you can single step through a portion of your target program after a complex set of conditions have been established in your target system.

Trig1 and trig2 are used to coordinate measurements between instruments in the instrumentation card cage. The analyzer can drive or receive either or both of these lines. Also, the rear-panel BNC and the CMB trigger signal can drive or receive either of these signals.

The above steps cause the emulator to break to its monitor program when the emulation-bus analyzer completes its trace. When you use trig1 and/or trig2 to coordinate actions in associated equipment there is delay in the coordination. In the example above, the emulator may execute several states between the time the analyzer completes its trace and the emulator breaks to its monitor program.

The analyzer can be set up to generate a trigger output on trig1 and/or trig2 on the following events:

- Recognition of the analyzer trigger (**tgout trig1 trigger**).
- Completion of a trace, the trigger captured and trace memory filled (**tgout trig1, trig2 complete**).
- Capture of a state that is <DELAY> number of states after the trigger was captured (**tgout trig2 -t 20**).
- Capture of a state that occurs after the trigger state and is <DELAY> number of states before the end of trace memory (**tgout trig1, trig2 -c 10**).
- Recognition of a specific state (**tgout trig1 addr=100**).

Recognition of a pattern when a particular sequence term is active in the complex mode (**tgout 1 p1**)

To set up the deep emulation-bus analyzer so its counts are enabled by an external instrument

- 1 Connect the rear panel BNC to deliver trig1 to the analyzer by typing:
bnct -d trig1 (BNC input drives trig1).
- 2 Set the emulation-bus analyzer to be armed when it receives trig1 by typing: **tarm =trig1** (trace analyzer arm signal supplied on trig1). The first time that the trig1 signal goes from false to true after the trace is started, the arm signal will switch to true and remain true for the rest of the measurement.
- 3 Set the emulation-bus analyzer to perform its count only when it is armed by typing: **tcq arm** (trace tag counter armed (enabled) by the arm signal).

The **tcq arm** feature is available on the deep analyzer, but not on the 1K analyzer.

You can connect a logic analyzer to the rear panel BNC and give it the power to control when the emulation-bus analyzer counts states. Perhaps you would like to have the emulation-bus analyzer count the number of calls to a particular routine, but only after execution of an external event. You could set up the logic analyzer to monitor that event, and to supply a TTL level to the rear panel BNC when the event occurs. Then the emulation-bus analyzer could be set up to count calls to the routine of interest, but only after the TTL-true is supplied from the logic analyzer.

7



Configuring the Emulator

How to adapt the emulator to your system

Each target 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.

To ensure that timing of accesses to emulation memory are the same as accesses to target system memory, you can lock termination of emulation memory cycles and monitor bus cycles to the target system TA and TEA signals. If your target system termination signals are not available, you can allow termination of the emulation-memory cycles to be done by signals within the emulator.

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.)

You can set a "keep-alive address" from which the background monitor will periodically read a byte. This can be used by your system to update a watchdog timer 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 in a configuration question.

A foreground monitor must be used when the MMU of the MC68040 is enabled, or the caches are enabled, or the target system does dma activity.

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

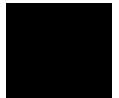
counter aren't initialized. A configuration item allows you to specify values for these registers so that the above sequence will work correctly.

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 data and instruction cache memories. The emulation-bus analyzer can't trace instructions (or data) that are fetched from cache. This can make trace displays difficult to interpret. When you disable the caches, all instructions and data are fetched on the external processor buses, 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 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.)

lock indicates that accesses to emulation memory are terminated by target system TA and TEA instead of the internal termination signals of the emulator.

tci asserts the $\overline{\text{TCI}}$ line to the MC68040 for all addresses in this memory block, indicating that accesses to this block should not be cached.

(Combine multiple attributes by separating them with commas, for example: **dp,lock**.)

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 MC68040 emulator has 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 4 Kbytes with the **dp** attribute, all 4 Kbytes are allocated because that is the minimum block size for that memory. If you specify a block size less than 4 Kbytes and the dual-port memory is unmapped,

Chapter 7: Configuring the Emulator

Mapping and Configuring Memory

the emulator uses that memory to more closely match the requested address range to the block size.

There are also two memory sockets on the probe. This memory is not dual-ported; the monitor is used to read and write the locations when you display or modify memory. You can install 256-Kbyte, 1-Mbyte, or 4-Mbyte emulator memory modules in these sockets in 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 not recommended because it does not allocate blocks as well as Installation 3.

** Memory Slot 0 and Memory Slot 1 are marked on the probe board as BANK 0 and BANK 1. Their locations are also shown in illustrations in the Installation and Service Chapter of this manual.

Your selection of wait states may be affected if you install 4M memory modules. See "To enable one wait state" later in this chapter.

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 **lock** attribute, the emulator waits for your target system \overline{TA} or \overline{TEA} signal to terminate an emulation memory cycle. This makes the bus cycle length identical to that of your target system so that timing will be the same. If your target system does not return \overline{TA} or \overline{TEA} in the address range mapped to emulation memory, don't use the **lock** attribute because the system will hang while waiting for your \overline{TA} or \overline{TEA} . (See "To interlock monitor cycles with your target system termination signals" for more information.)

If you don't specify the **lock** attribute when you map a memory block, your target system \overline{TA} and \overline{TEA} signals are ignored on accesses to that block.

If you specify the **tc** attribute, the \overline{TCI} (transfer cache inhibit) line is asserted for accesses to that memory block. This prevents instructions or data from that memory block from being loaded into the processor cache memory. If you need to disable caching for all memory accesses, use the **cf cache** 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 monitor before you map memory. Otherwise, you may need to reenter the map commands.

Example

Suppose you are using the emulator in-circuit, and there is a 12-byte I/O port at 1c000 hex in your target system. You have ROM in your 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 you must use the background monitor if you want to use the dual-port emulation memory to store your program.

Chapter 7: Configuring the Emulator

Mapping and Configuring Memory

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 64-Kbyte blocks and one 128-Kbyte block are used from the other emulation memory, leaving two 64-Kbyte blocks and one 128-Kbyte block. One of the 64-Kbyte blocks is used for the first map term, but 32 Kbytes of that block are unused and unavailable. The third term uses the other 64-Kbyte block. The second term uses part of the 128-Kbyte block, leaving the rest unavailable.

The mapper's resolution is independent of the block allocation. In the above example, if you had **map other grd** and your program accessed 8000h, the emulator would do a break on access to guarded memory.

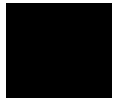
To assign the memory map default

- 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.

You can include attributes such as **tci** and **lock** with your **map other ...** command to inhibit read data from being written to the caches when unmapped addresses are on the emulation bus, and to allow the emulator to terminate bus cycles without waiting for the target system termination signals in unmapped ranges of addresses.



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

- If your external bus clock (BCLK) frequency is greater than 25 MHz, type:
cf wait=en
- If your external bus clock (BCLK) frequency is less than 25 MHz, type:
cf wait=dis
- To check the wait state setting, type: **cf wait**

When the clock speed of BCLK is above 25 MHz, the emulator requires one wait state for all accesses to memory, including burst mode accesses. Without this wait state, emulator operation is erratic. No wait states are required for memory accesses, including burst mode accesses, when the clock rate of BCLK is below 25 MHz.

When operating above 25 MHz, the target system is responsible for adding a wait state to its accesses. The emulator will not attempt to add a wait state to target accesses, other than to ignore cycle terminations until a wait state has passed. The target system is responsible for making sure cycle terminations and data are valid after the wait state.

The 4-Mbyte memory modules (HP 64173A) are not as fast as the 256-Kbyte and 1-Mbyte memory modules. The emulator always adds one wait state to accesses to emulation memory when it detects the presence of any of these 4-Mbyte memory modules on the emulation probe.

To enable the memory management unit

- To turn on the MMUs in the emulation processor, enter: **cf mmu=en**
- To turn off the MMUs in the emulation processor, type: **cf mmu=dis**
- To see the present state of the MMU, type: **cf mmu**
- To obtain additional information about the MMU, type: **help cf mmu**

Once enabled, the MMU of the MC68040 can be set up by the operating system to manage logical (virtual) memory in physical address space. The selection of a root pointer and the value in the translation control register determine how the MMU will manage memory. The MMU must be enabled by this configuration question before the operating system can establish those control values.

Your target system may enable the MMU during program execution by using the MDIS signal. The target system can disable the MMU even if it is enabled via the configuration question.

A foreground monitor must be used when the MMU of the MC68040 is enabled. If the background monitor is selected when you type in the **cf mmu=en** command, a message will advise you to select the foreground monitor first.

Note

Make sure the foreground monitor is mapped to memory space that has a 1:1 translation, and is not write protected. Refer to the end of this chapter for instructions on how to map the foreground monitor to 1:1 address space when using the MMU.

Examples

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

M>cf mmu=en

To select the emulation monitor

- To select the background monitor, type: **cf mon=bg**
- To select the foreground monitor, type: **cf mon=fg**
- To use the emulator without a monitor, type: **cf mon=none**
- 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, the MMU, caches, and dma cannot be used with the background monitor. Also, interrupts are disabled (including level 7) when the emulator is running in background. These conditions 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 MC68040 is enabled, or the caches are enabled, or the target system does dma activity. If the background monitor is selected when you try to enable the MMU, a message will appear to tell you that you must change to use of a foreground monitor before you can enable the MMU.

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.

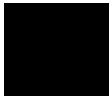
Chapter 7: Configuring the Emulator
To select the emulation monitor

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. The monitor (either foreground or background) will be loaded when the processor transitions out of emulator reset.

The following table summarizes the implementation of the monitor configuration.

Background monitor (cf mon=bg)	Foreground monitor (cf mon=fg)
monaddr, monlock, monintr —not available	monaddr —sets address block that will contain monitor in dual port memory
monkaa —sets address from which to periodically read a byte during background monitor operation	monlock —interlocks target system cycle termination signals to terminate emulation monitor cycles instead of using internal emulation cycle termination signals
	monintr —allows lowering of interrupt mask to this level during foreground monitor execution
	monkaa —not available



The special option **none** specifies that no monitor will be used. This option is useful when you are first connecting the emulator to a target system (refer to the chapter on plugging in the emulator to a target system). Sometimes the task of connecting an emulator to a target system can be complicated by characteristics of the emulation monitor. For example, foreground monitor bus cycles are visible to the target system. By selecting **none**, you eliminate the question "am I having trouble connecting to my target system because of something the monitor is doing?"

When you choose **none**, you will be able to run the emulator from reset (assuming you loaded a program earlier), and you will be able to take a trace with the analyzer to see what activity is being executed by your emulator. You will not be able to use any of the other emulator capabilities and features, such as loading programs or displaying memory. When your system is running successfully with the **none** selection, then select one of the other monitor options to see if your target system will operate with an emulation monitor.

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 foreground monitor, type:
cf monaddr=<ADDRESS>

where <ADDRESS> is a hexadecimal address on a 4-Kbyte boundary (XXXXXX000h).

- To check the monitor base address, type: **cf monaddr**

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, and 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/or **monlock** configuration items.

If the memory management feature of the MC68040 emulator is enabled, be sure the foreground monitor is mapped in an area that is translated 1:1, and it is not write protected. Refer to the end of this chapter for instructions on how to map the foreground monitor to appropriate address space.

To interlock monitor and target system cycle termination signals

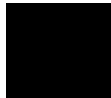
- To interlock the emulator and your target system cycle termination signals for monitor accesses, type: **cf monlock=en**
- To terminate monitor accesses with the emulator-generated cycle termination signals, type: **cf monlock=dis**
- To check the interlock setting for monitor accesses, type: **cf monlock**

When you enable interlocking (**cf monlock=en**), emulation monitor cycles aren't terminated until your target system asserts TA. If the monitor is in an address range where your target system does not assert TA, 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 disable interlocking (**cf monlock=dis**), the emulator-generated termination signals will terminate monitor cycles. Your target system cycle termination signals (such as TA and TEA) will be ignored during monitor accesses.

Bus cycles will be visible to your target system during foreground monitor operation. If interlocking is disabled, these cycles may cause erratic system operation if your target system isn't expecting them.

This configuration item determines whether or not the *lock* memory attribute is used in the foreground monitor memory map term.



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 when you are using the monitor, especially systems using real-time control where interrupt servicing must be done immediately.

To solve this problem, you can select the foreground monitor and lower the interrupt priority to a level that allows your system to function correctly, yet avoids excessive interrupt processing during use of the monitor routines. The emulator is reset when you change the setting of **monintr**.

When using this configuration item, enable target system interrupts with **cf ti=en**. Otherwise, interrupts from your target system will be blocked. See “To disable target system interrupts.”

If the processor’s interrupt priority level is greater than the value set by **monintr** at monitor entry, the processor’s priority level (not **monintr**) will be used. Otherwise, the interrupt priority is lowered to the setting of **monintr**.

The foreground monitor only lowers the interrupt priority to the level specified by **monintr** when it is executing non-critical code. When the foreground monitor is executing critical code (for example, on monitor entry and exit), 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

- 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 background monitor of the MC68040 emulator does not drive monitor cycles to your target system. Some target systems need to receive cycles during monitor operation. For example, your target system may have a watchdog timer that will time out if its keep-alive address isn't read periodically.

In this situation, you can 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.

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 define initial values for the interrupt stack pointer and the program counter when the emulator is reset, enter the monitor, and then attempt to run the target program, type: **cf rv=<RESET_ISP>,<RESET_PC>**

where **<RESET_ISP>** and **<RESET_PC>** are both 32-bit address values in hexadecimal. Both values must be word aligned. These values usually should correspond to the values loaded into offsets 0 and 4 of your vector table.

- To set the emulator to obtain the initial values for the interrupt stack pointer and program counter from the target system exception vector table, type: **cf rv=auto**
- 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 are not initialized. Therefore, the run will fail.

The **cf rv** configuration item is provided as a convenience to initialize the stack pointer and program counter to predefined values when the emulator enters the monitor after a reset. You can either specify values of your own choice, or allow the emulator to obtain the present values in the reset vector from the target systems exception table (**auto**). 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.)

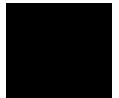
If entering specific values for the interrupt stack pointer and program counter, you will normally set the interrupt stack pointer to the value contained at offset 0 of your vector table, and the program counter to the value contained at offset 4 in your vector table. Because the monitor does not duplicate the reset exception processing sequence to access the reset vector, the **auto** option may not be suitable for all target systems.

Example

Assume the memory range 7000..7fff is mapped as **eram** and reserved as stack space. To 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 will be set to 7ff0 and the pc will be set to 400.



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.
- Interlocking breakpoint cycle termination.

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 its monitor program to implement 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 target memory.

While this configuration item affects which commands will be accepted, it does not affect hardware breakpoints such as write to ROM, break on analyzer trigger or guarded memory access breaks. It also doesn't affect the emulator's response to software breakpoints. When you set this configuration item to **dis**, all commands are accepted.

To disable the processor cache memories

- To disable the processor caches, type: **cf cache=dis**
- To enable the processor caches, type: **cf cache=en**
- To check the cache enable/disable setting, type: **cf cache**

The MC68040 processor has a cache that stores recently used instructions and another cache that stores recently used data.

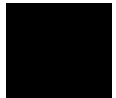
By using the cache memory, processor performance is improved, but the emulation-bus analyzer can't trace processor accesses that are completed in an internal cache. This may cause confusing trace displays or failure to trigger, especially if the code being analyzed is a small loop where all the instructions and operands fit into cache and registers.

When you disable the caches, 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 caches. If you are making analyzer measurements at the same time, you may need to experiment to find suitable trigger combinations.

This configuration item enables or disables the on-chip caches by controlling the `CDIS` signal. To disable the caches, the emulator will assert the `CDIS` signal to prevent your target system from enabling the caches. If **cf cache** is enabled, the `CDIS` signal from your target system and the cache control register (CACR) enable bits determine whether the caches are enabled.

If you need to disable caching only for accesses to a specific memory block, use the **tci** memory map attribute. This allows you to capture analysis information for specific memory ranges without dramatically affecting overall system performance. See "To assign memory map terms" in this chapter.



To disable target 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**

When target system interrupts are enabled, the emulator will respond to interrupts generated by the target system. When target system interrupts are disabled, the emulator will ignore all interrupts generated by the target system, including level 7 NMI interrupts.

Regardless of the selection you make here, 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.”

When the background monitor is used, all interrupts, including level 7 NMI interrupts, are ignored while executing in the monitor.

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 are not assigned. You can enable the interrupts when you're ready to test the interrupt handling routines.

To enable breakpoint acknowledge cycle termination interlocking

- To interlock the emulator and your target system cycle-termination signals for breakpoint acknowledge cycles, type: **cf bpllock=en**
- To terminate breakpoint acknowledge cycles with the emulator-generated cycle-termination signals, type: **cf bpllock=dis**
- Check if breakpoint acknowledge cycles are interlocked by typing: **cf bpllock**

When a software breakpoint instruction (BKPT) is executed, the processor generates a breakpoint acknowledge cycle as part of the instruction. As long as software breakpoints are enabled (**bc -e bp**), the emulator will terminate the breakpoint acknowledge cycle and initiate a transition into the monitor. If your target system cannot tolerate this behavior, the emulator can be configured to wait for the target system to terminate the breakpoint acknowledge cycle.

When interlocking is disabled (default), the emulator will terminate breakpoint acknowledge cycles with the **TEA** signal. Any termination signals generated by the target system will be ignored.

When interlocking is enabled, the target system is responsible for terminating breakpoint acknowledge cycles by asserting **TA** or **TEA**. If the target system fails to provide the required cycle termination, the processor will remain in a wait state indefinitely.

Modifying this configuration item will reset the processor.

Providing MMU Address Translations for the Foreground Monitor

When using the memory management unit (MMU) of the MC68040, the target system must provide correct address translations for the foreground monitor. To ensure correct address translations, you will need to understand your target system's memory map and MMU address translation structure. You may need to modify your mapping scheme or some of its mapping protections. If you do not obtain correct address translation for the foreground monitor, any attempt to break into the monitor after the MMU has been enabled may result in a target system bus error or undefined execution.

The foreground monitor will reside in the 4-Kbyte block of dual-port emulation memory. The dual-port memory can be mapped to begin on any 4-Kbyte address boundary. Simply specify an address ending in 000h when you answer the monitor address question when you set up the emulation configuration.

In order for the monitor to operate after the MMU is turned on, the target system must provide 1:1 address translation (logical address = physical address) for the block of memory occupied by the monitor. For example, if the monitor code begins at logical address 0fff1000h, then the MMU must translate that address to physical address 0fff1000h, logical address 0fff1004h to physical address 0fff1004h, etc.

Do not write protect the address range occupied by the foreground monitor.

There are two ways to provide the proper address translation for the memory space occupied by the foreground monitor:

- Locate the foreground monitor in a block of memory that is transparently translated via ITTx and DTTx transparent translation registers (TTRs). The monitor contains both code and data so two TTRs are needed to provide translations: one for instructions, and the other for data. When the MMU processes translations, it first compares the logical address with the parameters of the TTRs. If it finds a match, the MMU uses the logical address as the physical address for the access (obtaining the needed 1:1 translation).

The minimum block size that can be transparently translated by the TTRs is 16 Mbytes. If your target system already sets up one pair of data and instruction TTRs for supervisor, or both supervisor and user, access and no write

Chapter 7: Configuring the Emulator Providing MMU Address Translations for the Foreground Monitor

protection, then you may be able to find an unused 4-Kbyte block within this 16-Mbyte range where the monitor can reside.

If your target system does not use a pair of TTRs, then you may want to modify your MMU boot code to configure an instruction and data TTR specifically for the monitor.

Example

This example shows how to modify boot code to use a pair of TTRs. Assume your target system does not access any physical addresses in the 16-Mbyte range 02000000..02ffffffh, and DTT0/ITT0 are unused. By locating the monitor at address 02000000 and adding the following code fragment to your boot code, you should be able to break into the monitor while the MMU is turned on:

```
* configure ITT0/DTT0 for emulation monitor
MOVE.L  #$0200C000,D0
MOVEC   D0,ITT0
MOVEC   D0,DTT0
```

Without these transparent translations for the monitor, the MMU will probably generate an access fault when you attempt to break into the monitor. The access fault would occur because addresses in the 02000000 range would have no valid translations (they would be on a non-resident page).

If you cannot modify your boot code, you may be able to use an execution breakpoint to break into the monitor before the MMU is enabled and use the monitor to configure the TTRs. Do this only as a last resort because the MC68040 processor automatically disables all TTRs whenever an emulation or target reset occurs.

- Locate the monitor within a page that is controlled by the MMU address translation tables; one that is always resident, writeable, supervisor accessible, and translated 1:1. The monitor occupies one 4-Kbyte page of emulation memory. It will be stored in the 4-Kbyte range of the dual-port memory.

Locating the Foreground Monitor using the MMU Address Translation Tables

Locate the foreground monitor at a specific page address and add the proper address translation for this page in your supervisor address translation tables. The minimum page size is 4 Kbytes so the monitor only requires a single translation. The page that contains the foreground monitor must always be resident, translated 1:1 (logical address = physical address), and never be write protected.

The most direct way to do this is to modify the address translation tables in your source code, rebuild your executable file, and download the executable into RAM, or reprogram the executable into ROM. For systems that use an operating system to manage dynamic translation tables in RAM, the page allocated to the monitor must not be allowed to be swapped out by the operating system. This may require that the page selected for the monitor reside in unused space within the operating system (assuming the operating system is translated 1:1). The easiest way to create unused space is to globally define an 8-Kbyte array of data that is never referenced by your software. After rebuilding your operating system software, refer to the linker symbol map file to determine the address range of this array. Use the lowest address that resides on a 4-Kbyte boundary within this range as the starting address for the monitor.

As a last resort, if your target system software cannot be rebuilt, you can use the emulator to modify your translation tables directly.

The emulator provides a command to display individual address translations in detail, including address, value, and mnemonic information about each descriptor from the translation tables. You may be able to provide the proper address translation for the monitor by simply modifying a single descriptor (long word) to convert an invalid page into a resident page.

If the translation tables are located in ROM, you will need to copy them into emulation memory before you attempt to modify them. This is done by storing all or part of your ROM to a file, and then mapping emulation memory over the ROM address range and reloading the file.

8



Solving Problems

What to do when the emulator does not behave as expected

If the emulator appears to be malfunctioning

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 the emulator appears to be malfunctioning

- Check to make sure that the cables connecting the Emulation Control Board to the Emulation Probe are connected correctly. Refer to the Installation and Service chapter in this manual for details.
- Run the performance verification procedure as described in the Installation and Service Chapter of this manual. If the emulator fails this test, contact your Hewlett-Packard representative.
- If the emulator passes the performance verification procedure, look for other reasons for the problem. Performance Verification is a thorough test, but it cannot find every hardware failure in the emulator. It is a good indication that the emulator is functioning correctly, but if you are still convinced the emulator is malfunctioning, contact your local Hewlett-Packard representative.

If the trace listing states column contains "dma long write (retry)" repeatedly

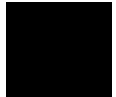
- Check to see if the internal ribbon cable that connects the last sixteen channels of the 80-channel internal analyzer to the HP 64783 emulator control board is missing. If it is, locate the supplied ribbon cable and connect one end to the slot in the analyzer board and the other end to the slot in the 68040 control board. Refer to the Installation and Service Chapter in this manual to see the proper location of this cable.

If the analyzer fails to trigger on a program address

- Check to make sure that the program address is a long-word address (an address ending in 0, 4, 8, or C hex). The MC68040 fetches instructions on long-word addresses. Other instruction addresses never appear on the processor bus, and therefore are never seen by the analyzer. Modify the trigger address so that the two least significant binary digits of your trigger address are zeroes. For example, to trigger a trace on address 2316H, specify your trigger to occur on address 2314H. Note that this only applies to instruction fetches; data reads and writes are made directly to the destination address, regardless of whether it is a long-word address or not.

If the analyzer triggers on a program address when it should not

- Check to see if the analyzer is triggering on an instruction prefetch. The analyzer cannot distinguish between prefetch and execution because the processor does not provide that information. Usually your actual trigger address is within 16 words of the address where trigger is occurring.
- Try to pad the program code with NOP instructions to move the trigger address away from the other code so that it won't be prefetched until it is time to trigger.
- You may be able to insert a write instruction to a meaningless variable in your code immediately following the trigger address. Then you can trigger on a write to the address of the variable. Write transactions never appear in instruction prefetches.



If trace disassembly appears to be partially incorrect

- Check to see if the analyzer began disassembly of the trace on a long-word boundary but the instruction started on the low word within the long word. This will make disassembly incorrect. You can start disassembly on the low word within the long word by use of **tl -d -ol <trace list line number>**.
- If the trace list seems correct for a few states after disassembly starts, and then it seems incorrect, restart disassembly of the trace at the low word where disassembly first becomes incorrect **tl -ol <trace list line number>**.
- If an instruction seems to have incorrect data associated with it, you can read down the trace list to see if you can find correct data for the instruction on another line. You can cause the disassembler to realign the instruction with the correct data by entering a command like **tl -d -ol <trace list line number containing instruction> <trace list line number containing data>**. For example, **tl -d -ol 38 47**.

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 the chapter titled "Configuring the Emulator.")

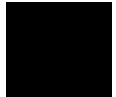
If the analyzer won't trigger

- 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 MC68040 fetches instructions from program memory, it addresses 32-bit longwords. These addresses are always multiples of 4 (ending in 0h, 4h, 8h, and Ch). 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 target processor remains in a wait state

- When you enable interlocking of the breakpoint acknowledge cycle termination (**cf bblock=en**), the target system is responsible for terminating breakpoint acknowledge cycles by asserting **TA** or **TEA**. If the target system fails to provide the required cycle termination signal, the processor will remain in a wait state indefinitely. Make sure your target system is providing the required cycle termination signal.



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, and reconnect the reset flying lead (See Chapter 15, “Installation and Service”).
- 4 Apply power to the emulator.
- 5 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.)

See the section “Mapping and Configuring Memory” in Chapter 7 for more information on memory allocation.

If emulation memory behavior is erratic

- Check to see if you have installed HP 64171A or HP 64171B memory modules on the emulation probe board. These memory modules are too slow to work with the MC68040 emulator. Use HP 64172A, HP 64172B, and/or HP 64173A memory modules.
- Ensure that you have answered the configuration question correctly for the memory modules in use. The configuration question establishes the required number of wait states to be used within 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

- Connect the reset flying lead to some point in your target system that distributes the reset signal to components that need to be reset when the processor is reset. This will make sure that all critical components in your target system are reset by the emulator. Suppose your system reset circuit drives several critical system components as well as the processor. Suppose also that the critical components are memory-mapping circuits that locate ROM containing the vector table at address zero for startup, then move it to a high address range after system initialization. An emulator reset cannot drive your reset line directly. Therefore, an attempt to run after emulation reset will fail because the vector table is not located in the correct place. For further information, refer to the chapter on plugging the emulator into a target system.

If the deMMUer runs out of resources during the loading process

- Check the physical address ranges that will be reverse translated by the present setup of the deMMUer. Enter **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.

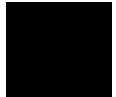
Read "Using the deMMUer" in Chapter 9 for ways to make more efficient use of deMMUer resources.

If "out of deMMUer resources" with less than eight mappings

- Check if you are using both root pointers in your memory mapping scheme? The deMMUer may have run out of resources for only one of the root pointers.
- Read "Using the deMMUer" in Chapter 9 to understand how deMMUer resources are allocated when using different root pointers or when using function-code mappings.

If only physical memory addresses in 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

- Some physical accesses are normal, especially accesses to the MMU tables.
- Check to see which physical memory spaces are being reverse translated by the deMMUer. Enter the **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 MMU

- Enter the command: **rst -m**. If your MC68040 is now running in the monitor, look at your MMU Tables or the transparent translation register that maintains 1:1 mapping for your foreground monitor. The mapping has failed. Modify your MMU tables or the transparent translation register to obtain the 1:1 mapping for the address space occupied by the foreground monitor.
- Refer to the end of the chapter titled, "Using Memory Management" for a detailed example that discusses how to solve a "can't break into monitor" problem.

If you see exclamation marks "!" in count columns of the trace lists

- This is a normal condition. It indicates the counter overflowed (began again at 0) before the present state was captured. The exclamation mark warns you that the counter value may not be accurate because the analyzer is unable to determine how many times the counter overflowed between the preceding state and the state where the exclamation mark is shown.

If you were to scroll through a trace list of the entire trace memory in relative count mode, a "!" would be seen beside the first state after each occurrence of counter overflow (each 22.9 minutes). If you were to scroll through the entire trace memory in absolute count, the "!" would be seen beside every state after the first occurrence of counter overflow. Refer to the last illustrations in this chapter for an example of a terminal interface trace list with exclamation marks.



If you see negative time or state counts in trace lists

- If counter overflow occurs during a trace measurement, you may see a count of negative time or negative states in trace lists using the absolute time count mode. This indicates that the counter value stored with the trigger state was greater than the counter value stored with the present state. In absolute time counts, negative times will continue to be seen until a state is captured whose counter value is greater than the trigger state counter value. In relative time counts, the counter value is corrected so no negative time is seen.

If you do not see the counter overflow indication "!" where you expected to see it in a trace list

- This may be a normal indication. If you scroll through a reduced portion of the trace memory, one that contains no counter overflow, no counter overflow indication will be seen, even if counter overflow occurred before the line range you specified in your **display/store/copy** command. The routine that reads trace memory to compose a trace list only reads the portion of the trace memory you specify in your **display/store/copy** command.

If your target system loses sync when emulation breakpoints are executed

- Try the configuration specification **cf bblock=en**. This option causes the emulator to wait for cycle termination signals to arrive from the target system when a breakpoint is executed. Normally, the target system is ignored when the emulator executes a breakpoint. On rare occasions, your target system may lose sync with the emulator during breakpoint cycles.

Part 3

Reference

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 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 68040.

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 the MC68040 emulator.

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 Memory Management

Understanding logical and physical emulation and analysis

Understanding Emulation And Analysis Of The Memory Management Unit

You only need to read this chapter if you are using the on-chip MMU (Memory Management Unit) of the MC68040 or MC68LC040 microprocessor. If you are using an MC68EC040, or if you are using an MC68040 or MC68LC040 with its MMU disabled, you won't need the information in this chapter.

This chapter begins with a discussion of terms and conditions you need to understand when you are using the MC68040 or MC68LC040 emulator/analyzer with the MMU enabled. Under these conditions, many capabilities and features become available that are not otherwise offered. Also, some of the features you have been using behave differently. These are discussed in this chapter.

Terms And Conditions You Need To Understand

The following paragraphs explain the differences between logical and physical memory, and between static and dynamic virtual memory systems.

Logical vs Physical

When you develop a program, compile it or assemble it, and link it, addresses are assigned to contain each of the bytes of the program. These addresses are logical addresses. When the program is loaded into hardware memory so that it can be executed by the microprocessor, it is loaded into physical address space. When you are not using an MMU, the program is loaded into physical memory hardware at the logical addresses assigned in the linker load map. Under these conditions, there is no need to differentiate between logical addresses and physical addresses because they are the same (simply addresses). When you use the MMU, it becomes necessary to understand the difference between logical addresses and physical addresses.

Most emulation and analysis commands that require an address as part of the command use logical addresses. Some emulation and analysis commands will accept either logical or physical addresses.

What are logical addresses?

Logical addresses are the addresses that are assigned to your program code when you develop your program. They are the addresses represented by symbols in your symbols data base (the symbol "Main" represents a logical address).

What are physical addresses?

Physical addresses are the addresses assigned by the MMU to contain your program. Physical addresses identify locations where you actually have memory hardware in your target system. Physical addresses appear on the processor address bus instead of logical addresses.

Static and dynamic system architectures

There are several design strategies where memory management can help in developing a system or product. Three of these are described in the following paragraphs. One shows memory management used in a static memory system. The other two show memory management used in different dynamic memory systems. The MC68040 emulator is designed to work in any of these system types; however, the deMMUer which provides reverse translations to the analyzer is primarily intended for use in static systems.



Static system example

A static system design may use the MMU simply to protect supervisor code and I/O space against accesses from a user program. Once a static system is initialized, it never changes. Your HP emulator and analyzer can give you complete support for a static memory management system. After the MMU has been set up to manage memory in a static system, the deMMUer can be loaded with information to reverse the MMU translations over the entire range managed by the MMU.

Non-paged dynamic system example

Assume three programmers are developing separate programs to run in a real-time operating system environment. The programmers each write their programs to begin at address 0h. The operating system accepts the responsibility to know where in physical memory space each of these programs will be located. The programmers don't have to worry that some additional code they write in their programs might overwrite some of the code that was written by another programmer. The operating system will place all of the code in available memory space and place appropriate translation mappings in the MMU to ensure that when the logical address for one of the programs (tasks) is present in the program counter, the appropriate physical address will appear on the bus to access the desired physical memory location.

Your HP emulator/analyzer can give you partial support for a non-paged, dynamic system. When the MMU has been set up to manage memory during execution of one of the above tasks, you can update the deMMUer to translate addresses for that task. When that task is executing, the analyzer will be able to make trace measurements and provide correct results. When any of the other tasks are executing, trace measurement results will be invalid because the other tasks will depend on different translation tables in the MMU and there is no way to automatically update the deMMUer when execution switches from one task to another.

Paged dynamic system example

Assume you have developed a program that occupies 10 megabytes of logical address space. Perhaps you have only 2 megabytes of physical address space in your system. Still, you want to be able to run the entire program. You set up a specification in the MMU translation control register to divide the address space into pages (the 68040 lets you divide the memory space into one of two page sizes: either 4Kbytes, or 8Kbytes). Assume you set up the MMU to divide the memory into 4-Kbyte pages. Your program will occupy 2,500 pages of code, and 500 of these pages can be contained within your physical memory space at any given time.

As your program executes, the operating system moves pages of your program code into address space in physical memory. When execution goes beyond the addresses contained on the presently active page, the MMU checks to see if the next logical address is on a page that has already been placed in physical memory. If it is, the MMU performs the appropriate translation for the next logical address, placing the appropriate physical address on the bus, and execution continues. If it is not, the operating system moves the page that has the next address to be executed up from

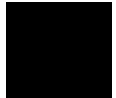
an external storage device to physical memory space, overwriting one of the pages that had occupied physical space before. The operating system updates the translation tables to identify the new logical address space that now occupies that 4 Kbyte of physical memory, and program execution continues.

As pages are swapped back and forth between an external storage device and the physical memory, the relationship between any one logical address and its corresponding physical address may change many times.

Your HP emulator will let you run a paged, dynamic system, but the analyzer will not be able to provide support for such features as symbolic addresses, or display of corresponding source files. The deMMUer cannot detect changes in the MMU mappings. The longer the system runs, the further out of date the deMMUer will become. Of course, the analyzer will still be able to show activity captured at physical addresses. By experimenting with several starting points for the inverse assembler, you can obtain a trace list with activity inverse assembled into an equivalent assembly language listing (**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 supervisor and user privilege modes

The MMU allows separate tables to be set up for supervisor and user access. For example, you can create one set of mapping tables to translate addresses in supervisor space and another set of mapping tables to translate addresses in user space. The supervisor space uses the SRP (supervisor root pointer). The user space uses the URP (user root pointer). The supervisor address space can begin at supervisor address 0 and the user address space can begin at user address 0. The MMU must ensure that these addresses are placed in different physical spaces.

You can use the MMU to protect your program space from unauthorized accesses. If you map a portion of your program through the MMU and identify it as supervisor space, the MMU will not allow any access to that program space unless the privilege mode is supervisor at the time the access is attempted. Take care to ensure that supervisor or user is specified with addresses if the MMU will be making the distinction (example: <address>@s).

How the MMU is enabled

The MMU depends on a hardware enable and a software enable. Both of these enables must agree to enable the MMU before it can translate logical addresses to physical addresses. If either one (or both) of these enables fail to enable the MMU, it will remain disabled.

Hardware enable

The hardware enable is performed by the $\overline{\text{MDIS}}$ signal. When $\overline{\text{MDIS}}$ is asserted, the MMU is disabled. When $\overline{\text{MDIS}}$ is negated, the MMU is enabled to translate addresses. The emulator controls the $\overline{\text{MDIS}}$ line according to the way you set the "mmu" configuration parameter.

If you enter **cf mmu=dis**, the $\overline{\text{MDIS}}$ line is held asserted. If you enter **cf mmu=en**, the $\overline{\text{MDIS}}$ 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 an enable value into the translation control register (TC). If the enable bit of the TC register is "e=1", the MMU will be enabled. If the enable bit in the TC register is "e=0", the MMU will be disabled.

Restrictions when using the emulator with the MMU turned on

There are only three restrictions: you must use a foreground monitor, it must not be write protected, and you must map it to address space that the MMU translates 1:1 (logical=physical) for supervisor accesses.

You must use a foreground monitor. The background monitor does not have the capabilities to support the MMU functions. The foreground monitor can operate with the MMU turned on.

You must map the monitor code to address space that the MMU translates 1:1 for supervisor accesses. The emulator executes monitor code to implement many of its emulation features. The emulator must be able to find the monitor code whether the MMU is turned on or off. By mapping the monitor into address space that has a 1:1 translation, the monitor stays within known address space at all times, and the emulator can always find it when it needs to use it.

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 in RAM space are always logical addresses. When you set a breakpoint at an address, that address is translated by the MMU and the BKPT instruction replaces the instruction at the appropriate physical address. When the breakpoint is executed, the emulator restores the original instruction to the physical address, by first translating the logical address through the MMU.

Consider what happens if you set a breakpoint at a particular address, and before the breakpoint is hit, you update the translation tables in the MMU, changing the mapping to the location where the breakpoint is set. This is discussed in detail under "Solving Problems" at the end of this chapter.

If you enter a command to display memory or modify memory, your command is directed to logical address space. If you want to display memory at a physical address, you have to change your command. For example, the command to display memory at address 100H (**m 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 emulator will temporarily modify the translation tables to allow the access.

Seeing details of the MMU Translations

The following paragraphs discuss emulator displays that help you understand translations made by your MMU. There are three displays, each giving a different level of detail of the MMU translations.

- The present address mappings in your MMU tables.
- The translation table entries for a single logical address.
- The contents of a single level of the translation tables pointed to by a selected logical address.

How the emulator helps you see the details of the MMU mappings

To see all of the logical-to-physical translations presently mapped, enter the command **mmu**. The emulator will read the present state of the translation tables

```
M>mmu
Logical Address      Physical Address     Attributes
000089000..000089fff@s 0fff89000..0fff89fff@sa S W
00008a000..00008afff@s 0fff8a000..0fff8afff@sa S W
00008b000..00008bfff@s 0fff8b000..0fff8bfff@sa S W
00008c000..00008cfff@s 0fff8c000..0fff8cfff@sa S W
00008d000..00008dfff@s 0fff8d000..0fff8dfff@sa S W
00008e000..00008efff@s 0fff8e000..0fff8efff@sa S W
00008f000..00008ffff@s 0fff8f000..0fff8ffff@sa S W
000090000..000090fff@s 0fff90000..0fff90fff@sa S W
000091000..000091fff@s 0fff91000..0fff91fff@sa S W
000092000..000092fff@s 0fff92000..0fff92fff@sa S W
000093000..000093fff@s 0fff93000..0fff93fff@sa S W
000094000..000094fff@s 0fff94000..0fff94fff@sa S W
000095000..000095fff@s 0fff95000..0fff95fff@sa S W
M>
```

How the emulator helps you see the details of the MMU mappings

and show all of the valid mappings in those tables. The display will be similar to the following:

The above listing shows privilege modes were included in the mapping scheme. The logical and physical addresses are shown in supervisor space. Notice that the physical addresses also show "a" beside the privilege mode indication. The "a" indicates physical address space.

Note that the emulator enters the monitor to obtain the information it shows in the MMU displays. Execution of your target program is suspended while the emulator gathers information for an MMU display. If there are portions of your program that should not be interrupted during execution, insert an execution breakpoint in some safe area of your program code and run until the breakpoint is executed. Then you can safely view the MMU mappings.

The display you get with the **mmu** command can show as much 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.

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..0fff** instructs the emulator to show the valid mappings for only the logical addresses in the range of 0 through 0fff, instead of all possible mappings.

Another way to limit the number of address ranges shown in an mmu mappings display is to limit the listing to only user or supervisor address space. The command **mmu 0..0fff@u** will show all of the mappings for addresses from 0 through 0fff in user address space.

The display shows TT beside address ranges that are overridden by the transparent translation registers. In these ranges, logical-to-physical address translation will be 1:1. The MC68040 always compares logical addresses to the content of the transparent translation registers before it attempts a translation. If it finds a match in the transparent translation registers, it accepts the logical address as the physical address and performs no translation.

Supervisor/user address mappings

If you are using separate supervisor and user mappings, the emulator will support this choice and show appropriate information.

- To see only the mappings in supervisor address space, use the command: **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 in user address space, use the command: **mmu <address>[.<address>]@u**.

The MC68040 uses the URP as the root pointer for user address space, and the SRP as the root pointer for supervisor address space.

No distinction is made between program and data space.



Translation details for a single logical address

To see translation details for a logical address, enter a command such as: **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.

```
M>mmu -t 40f8H
Logical Address (hex)      0    0    0    0    4    0    F    8
Logical Address (bin)    0000 0000 0000 0000 0100 0000 1111 1000
Table Level              AAAA AAAB BBBB BBCC CCCC PPPP PPPP PPPP

LEVEL INDEX LOCATION   CONTENTS   TBL/PAGE G Ux S CM M U W UDT/PDT
SRP
A      000 02028200   ffffffff 02028200                y y RESIDENT
B      000 fffffe00   ffffffff fffffe00                y y RESIDENT
C      004 ffffff10   ffffffff fffff000 y 11 y in y y RESIDENT

Physical Address (hex) = fffff0f8
M>
```

Address mapping details

The example display shows:

- The translation mapping for logical address 40f8H in supervisor space. Both the hexadecimal and binary values are shown for the logical address.
- The Table Level line shows how each address bit is mapped. The first seven bits are used as an offset into Table A. The next seven bits offset into Table B. The next six bits offset into Table C. The example display was made with 4-Kbyte pages selected; only five bits index into Table C when 8-Kbyte pages are selected. The lowest-order 12 bits contain the offset into the physical page.
- The index used in Table A is 0 which points to physical address 2028200. The content of this address is ffffffff, indicating a B level table located at base address fffffe00. The status also indicates that this table has been used "U", and the address is write protected.
- The physical address is finally calculated by adding the physical page offset to the base address of the physical page.

Status information

Status can be assigned to an address at any point in its mapping. To interpret status, you must OR the status information at each level of the mapping. For example: the "M" bit shows that the content of the page indicated by Table C has been modified (by a write or read-modify-write). This applies only to addresses in this page. A "y" might have been shown under the "S" status bit in the A line. It would indicate that only supervisor accesses are allowed for pages under the A table. This restriction would apply to all addresses of this table, even though S=1 only appeared at the upper level of the table.

Note that the address shown in the example display was mapped through the supervisor root pointer. If you wanted to see the mapping through the tables under the user root pointer, you would use a command like `mmu -t 40f8@u`. You can add the desired function code table index to your command to see how any address is mapped through the tables under the selected root pointer (e.g. `u` or `s`).

The specific status bits shown beside each table entry are defined as follows:

- TBL/PAGE indicates the base address of the next table.
- G means the entry is global.
- Ux shows the values of the user programmable attributes (signals UPA0 and UPA1).
- S means supervisor mode protection.
- CM identifies the cache mode: cw (cachable, writethrough), cc (cachable, copyback), is (inhibited, serialized), or in (inhibited, nonserialized).
- M means the page has been modified.
- U means the page (or pages) has been used, or previously accessed.
- W means the page is (or pages are) write protected.

UDT/PDT indicates whether the page at the next level is RESIDENT or INVALID.



Table details for a selected logical address

The lowest level of detail you might like to see is the content of one of the tables used to map a particular logical address. You might enter a command such as: **mmu -tc 40F8**. The emulator would interpret this as a command to show the details of Table C where it is used to map logical address 40F8. There might be a great many Table C's, but this command will only show the Table C that is used to map the logical address you specified in your command.

In the example display of table details:

- The LOCATION column shows the physical address of each indexed location in Table C.
- The TBL/PAGE column shows the base addresses of physical pages indicated by each location in Table C.
- The first indexed location in Table C shows that its associated physical page has been accessed, but not modified ("U" bit = "y", and "M" bit = "n").

```

U>mmu -tc 40f8
Logical Address (hex)      0    0    0    0    4    0    F    8
Logical Address (bin)    0000 0000 0000 0000 0100 0000 1111 1000
Table Level              AAAA AAAB BBBB BBCC CCCC PPPP PPPP PPPP

LEVEL INDEX LOCATION  CONTENTS  TBL/PAGE G Ux S CM M U W UDT/PDT
SRP
A      000 00000200  00000200  00000200          y n RESIDENT
B      000 00000400  0000040b  00000400          y n RESIDENT
C      000 00000600  000008f   00000000 n 00 y cw n y y RESIDENT
C      001 00000604  00001087  00001000 n 00 y cw n n y RESIDENT
C      002 00000608  00002087  00002000 n 00 y cw n n y RESIDENT
C      003 0000060c  00003087  00003000 n 00 y cw n n y RESIDENT
C      004 00000610  00004087  00004000 n 00 y cw n n y RESIDENT
C      005 00000614  00005087  00005000 n 00 y cw n n y RESIDENT
C      006 00000618  00006087  00006000 n 00 y cw n n y RESIDENT

```

Using the DeMMUer

The deMMUer circuitry reverses the translations made by the MMU (translates the physical addresses it finds on the processor buses back to their corresponding logical addresses) before sending the addresses to the analyzer.

What part of the emulator needs a deMMUer?

Actually, the emulator doesn't need the deMMUer; the analyzer does. It can't provide its full symbolic features unless it has help from the deMMUer. The analyzer normally receives its address information directly from the processor address bus. It uses the symbols data base created for the program loaded in memory to cross reference the addresses it receives to the symbols and corresponding code in your source files. When the MMU is used, logical addresses are translated to physical addresses before they are placed on the processor address bus. Therefore, they no longer match the symbols data base.

What would happen if the analyzer didn't get help from the deMMUer?

The analyzer would get its address information directly from the address bus of the emulation processor. It would have no way to know what translation had occurred in the MMU. Therefore, it could not trigger or qualify its trace on any symbol defined in the symbols data base. Further, its trace list could only show you the physical address value it found on the address bus; it would not be able to show any symbols associated with that physical address, or any corresponding source file lines. You would have to figure out for yourself what portion of your program address space was executing when that physical address appeared on the bus.

How does the deMMUer serve the analyzer?

The analyzer 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 disabled. 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. The deMMUer will be enabled as soon as it is loaded.
- **-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.
- **-c** clears all reverse translations in the deMMUer.
- **-r** prepares the deMMUer to receive a register name and value. The value is to be used instead of the present register value when loading the deMMUer. For example, **dmmu -l -r srp=02028308**.

What the emulator does when it loads the deMMUer

- **-t** prepares the deMMUer to receive your entry of a reverse translation from the command line. For example, **dmmu -t 0..1fffh 8000000..8001fffh**. The logical address is first, followed by the corresponding physical address.

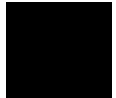
What the emulator does when it loads the deMMUer

When the emulator loads the deMMUer from memory, it does the following:

- Temporarily breaks into the monitor.
- Reads the MMU registers and translation tables from memory to determine all logical-to-physical address translations.
- Loads the reverse translations into the deMMUer hardware.
- If the accessible physical memory exceeds the 256-Mbyte limitation of the deMMUer, the emulator reads the emulation memory map for help in selecting appropriate address ranges to reverse translate.

When the emulator loads the deMMUer from a file, the only difference in the above algorithm is that the address translations are obtained from the file instead of by reading MMU registers and translation tables from memory.

See Resource Limitations later in this chapter.



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 URP registers in the MMU, and the present translation tables, and calculates the address translations that can be performed (all possible physical-to-logical translations are determined during this process). Then the emulator loads the deMMUer to reverse those translations. After the deMMUer is loaded, any change to the MMU, its registers, or its translation tables will make the deMMUer out of date. The only way to update the deMMUer for changes in the translation setup is to load the deMMUer (**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 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 breakpoint is executed.

The analyzer must be off

Your analyzer must not be making a trace when you load the deMMUer. Otherwise, part of the trace will be based on physical addresses and the other part will be based on logical addresses.

Expect strange addresses if you analyze physical memory with multiple logical mappings

The deMMUer can only translate a physical address into one logical address. If two programs both use the same physical space (such as when two programs use a single data location), they might refer to that space by two different logical address values (and two different logical address symbols). The deMMUer translation RAM will be loaded with only one of the logical addresses. This means that you

Chapter 9: Using Memory Management Restrictions when using the deMMUer

might be analyzing execution of your program and find it accesses a data space at an address you don't recognize, even though the data may be what you expect to see. The unexpected address will be the logical address known to the other program that also uses this location.

The way the deMMUer selects a logical address for a physical address when two or more logical addresses are available is as follows:

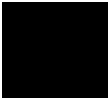
- The deMMUer selects the logical address with the lowest address value.
- If one of the addresses is controlled by the MMU tables and one by a transparent translation register, the deMMUer sends the address defined in the MMU tables.
- If one of the logical addresses is within a range defined in the emulation configuration memory map and another is not, the logical address defined in the memory map is sent.



Resource limitations

If you enter the 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 "Out of deMMUer resources". Note that when you see one error message, there may have been other messages generated at the same time. Display the error log to see all of the error messages that were generated. This will give you additional information about the error that caused the message to appear.

The deMMUer has a table where it records ranges of physical addresses that it can reverse translate to logical addresses. This table has eight entries, and each entry contains a single physical address range. Each address range in the table is 32 Mbytes. Up to 256 Mbytes of physical addresses can be reverse translated. Normally, entries in this table are allocated automatically, without intervention.



address..address
address..address
address..address
address..address
address..address
address..address
address..address
address..address

Example to show resource limitations

Consider the following program arrangement:

4M RAM	Unused	2M Peripherals	Unused	4M ROM	Unused
0	4M	256M	258M	512M	514M

Assume a system contains memory and peripherals at three different ranges: one from 0 to 4 Mbytes, one from 256 to 258 Mbytes, and one from 512 to 514 Mbytes. The rest of the physical address space is unused.

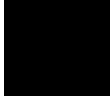
If your MMU mapping tables are set up to only allow access to memory in these ranges, your deMMUer will load properly and you can proceed with your measurements. If you failed to restrict your MMU mappings to these physical ranges (instead you provided valid address translations for the entire 4-Gbyte address space), the deMMUer will allocate all eight of its resource blocks in the first 256-Mbyte range, and no deMMUing will be provided for the peripherals and ROM space in the above program.

The Emulation Memory Map Can Help

When the emulator tries to load the deMMUer and finds more physical memory identified in the MMU mapping tables than it can translate in its deMMUer table, it will assign resources to terms defined in the emulation memory map. If the emulation memory map is arranged as follows, the deMMUer will load in a way that ensures the physical ranges of interest will be in the deMMUer.

```
00000000..003fffff eram
10000000..101fffff tram
20000000..203fffff trom
other tram
```

When the emulator reads the emulation memory map for help in loading the deMMUer, it sorts the entries: first by size, and second by address range. The smallest address range (256M to 258M) will occupy the first resource block in the deMMUer translation table. Address range (0 to 4M) will occupy the second resource block, and address range (512M to 514M) will occupy the third resource block. The remaining five resource blocks will be assigned to other physical ranges found in the MMU tables, beginning with the lowest



Chapter 9: Using Memory Management

Resource limitations

addresses. You may see a message indicating some physical addresses will not be translated by the deMMUer, or Out of DeMMUer resources, because the deMMUer might run out of resource blocks before all of the physical addresses have been assigned reverse translations, but the program spaces you care about will all be reverse translated. You can use the **verbose** option of the deMMUer load command to make sure the program spaces you care about will be reverse translated.

If you map a space greater than 256 Mbytes in the emulation memory map, you will run out of resource blocks before you satisfy the map.

The best way to ensure that all of the address ranges you care about will be reverse translated is to compose an emulation memory map that allocates blocks of physical memory only large enough to accommodate the address space occupied by code you are trying to develop. The deMMUer algorithm will allocate resource blocks in its eight-entry table to reverse translate only those physical address ranges.

With the above example, you could have avoided running out of resources. If you had placed invalid descriptors in your MMU tables in the paths that lead to unused physical address ranges, the deMMUer would have had more than enough resource blocks in its eight-entry table to reverse translate the valid address ranges.

Finally, you can store the present setup of the MMU to a file, and then use an editor to eliminate address ranges that do not need to be reverse translated. This only leaves address ranges that need to be reverse translated in the file. Then you can load this file into the deMMUer. When this file is loaded, the deMMUer creates a set of reverse translations for it, ignoring the MMU setup in the emulator. Refer to "Saving and Restoring DeMMUer Setup Files" in the chapter titled "Using the Emulation-Bus Analyzer" for how to store and load a deMMUer file.

Dividing the deMMUer table between user and supervisor address space

You can have two sets of MMU translation tables, one under each root pointer (URP and SRP). In this case, the emulator divides the deMMUer table into two equal address spaces. The first four resource blocks provide reverse translations for user physical address ranges, and the last four resource blocks provide reverse translations for supervisor physical address ranges.

There are cases where the deMMUer table will not be divided into two sets of four resource blocks each, even if you are using both root pointers (URP and SRP). If the values of your user root pointer and supervisor root pointer are the same (URP = SRP), and if the user and supervisor function codes are ignored in all of the transparent translation registers, then the deMMUer table will not be divided. It will make its eight resource blocks available to reverse translate either user or supervisor space.

If the user root pointer and supervisor root pointer contain different values, or if function-code mapping is used in any of the transparent translation registers, the deMMUer table will be divided into two 4-block tables as shown below.

address..address@u
address..address@u
address..address@u
address..address@u
address..address@s
address..address@s
address..address@s
address..address@s



Solving Problems

Your program and emulator may be running fine until you turn on the MMU. Then program execution may fail. You may not be able to use features of your emulator. How can this happen? It can happen if the MMU mapping tables are incorrect. When the MMU turns on and starts managing memory by performing tablewalks in tables that are invalid, pages of logical memory may overwrite your stack space, your emulation monitor, or any other address space, making your entire system unusable. If this happens, note where the program is executing. The stack may be inaccessible. The monitor (with its emulation feature set) may be inaccessible. The vector table may be placed in guarded memory. Program data space may become inaccessible.

Using the "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, URP, 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, URP, and SRP registers, the emulator will let you specify correct values to be used when composing the display of translations by using the **-r** option with **tc**, **urp**, and/or **srp** register names in the **mmu** command.

Use the analyzer with the deMMUer to find MMU mapping problems

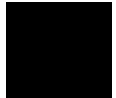
If your system operates properly until you turn on the MMU, and then it fails, the problem is most likely in the mappings used by the MMU to translate logical addresses to physical addresses. You could go down the list of logical-to-physical translations to see the mapping scheme used to translate each 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**).



Chapter 9: Using Memory Management

Execution breakpoint problems

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.

Execution breakpoint problems

If you set a breakpoint in RAM, the emulator modifies memory using a logical address. If you set a breakpoint in ROM, the emulator translates a logical address into a physical address and remembers the physical address as the address where it will jam the breakpoint instruction when it is fetched. If your MMU address translations change while breakpoints are activated, you can get the "undefined software breakpoint" message when you run your program or the "breakpoint code already exists" message when you attempt to modify the breakpoints.

You set a breakpoint. Then the MMU changes its mappings. Now the logical address where the breakpoint is to occur is translated to a different physical address. No emulation break occurs when the logical address is translated to the new physical address. Some different logical address is translated through the MMU to reach the physical breakpoint address, and the emulator jams the BKPT instruction. When the BKPT instruction is executed, it is at a point in your program where you never set a breakpoint.

You should disable any hardware breakpoints before changing the MMU address translations. Reenable the hardware breakpoints after the MMU address translations have been modified.

A "can't break into monitor" example

The following example assumes you mapped your foreground monitor beginning at address 4000H. You connected your emulator into your target system and ran your target program (which set up the MC68040 MMU). You tried to break into the emulation monitor and got the message, "Can't break into monitor."

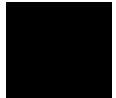
The emulator can't break into the monitor because it can't find the monitor. The MMU mapped the foreground monitor to physical address space that is not a 1:1 translation from logical address space.

A variety of failure modes can happen at this point. Your emulation system may execute unknown code, or it may simply halt.

To analyze this problem, reset into the monitor with the command: **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.



Chapter 9: Using Memory Management

A "can't break into monitor" example

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
00004000..00004fff  0000C000..0000cfff@a
```

The next step in this analysis is to display the MMU mapping table for the logical base address of the foreground monitor. You might enter the command: **mmu -t 4000**. In this example, you would see the following display of mappings:

```
Mmmu -t 4000
U>mmu -t 00004000
  Logical Address (hex)      0   0   0   0   4   0   0   0
  Logical Address (bin)      0000 0000 0000 0000 0100 0000 0000 0000
  Table Level                AAAA AAAB BBBB BBCC CCCC PPPP PPPP PPPP

  LEVEL INDEX LOCATION   CONTENTS   TBL/PAGE G Ux S CM M U W UDT/PDT
  SRP
  A      000 00000200    0000040b  00000400
  B      000 00000400    0000060b  00000600
  C      016 00000640    0000c01f  0000C000 n 00 y cw y y n RESIDENT

  Physical Address (hex) = 0000c000
U>
```

In the example display, the foreground monitor whose logical address is 4000 was placed in physical address C000. Table C points to the page containing the foreground monitor. The base address of Table C is 00000600, and the content used by logical address 4000 is at index 016 whose physical address is 00000640. The content of this address is 0000C000H (the address of the page containing the monitor).

To solve the problem in this example, you can obtain the needed 1:1 mapping by modifying the content of the MMU table directly with the following command: **M>m -dl 00000640=0000401f**

Chapter 9: Using Memory Management A "can't break into monitor" example

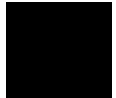
After this modification, you can get a new display of the mapping tables for logical address 4000 to see if your modified MMU tables now map your foreground monitor correctly. Enter the command: **M>mmu -t 4000**

```
U>mmu -t 00004000
Logical Address (hex)      0      0      0      0      4      0      0      0
Logical Address (bin)    0000 0000 0000 0000 0100 0000 0000 0000
Table Level              AAAA AAAB BBBB BBCC CCCC PPPP PPPP PPPP

LEVEL INDEX LOCATION  CONTENTS  TBL/PAGE G Ux S CM M U W UDT/PDT
SRP                00000200 00000200
A      000 00000200 0000040b 00000400
B      000 00000400 0000060b 00000600
C      016 00000640 0000401f 00004000 n 00 y cw y y n RESIDENT

Physical Address (hex) = 00004000
U>
```

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 64783 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 your target (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**

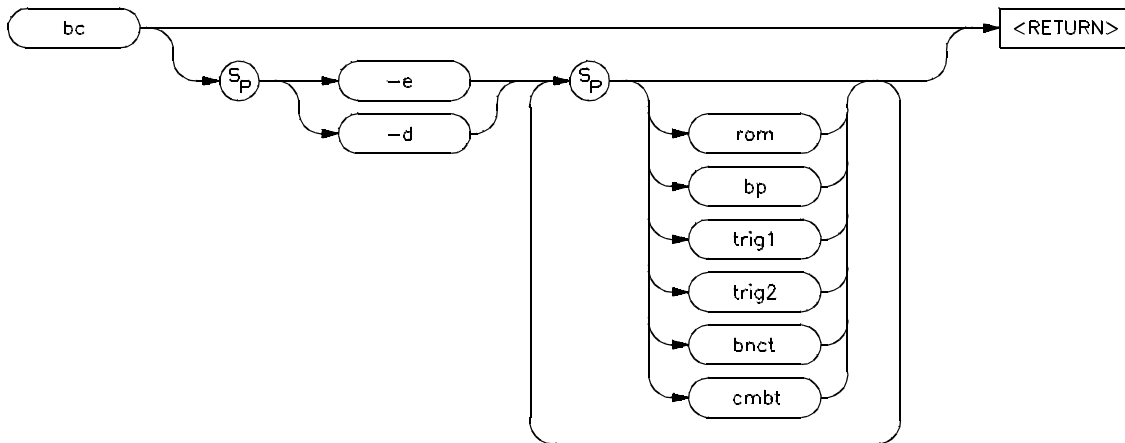
If the emulator is in the reset state when a break occurs, it will be released from reset and will begin execution within the emulation monitor.

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. If no parameters are specified, the enable/disable status of all six break conditions is displayed.

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 target (user) program. Several program instructions may execute after a write to ROM occurs before the emulator enters the monitor.
 - bp Enable/disable recognition of execution breakpoints inserted with the **bp** command. When a breakpoint instruction is executed in the emulation microprocessor, the monitor is entered immediately.
 - 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. Several instructions may execute between occurrence of the event that initiated this break and the emulator's entry into the monitor.

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. Several instructions may execute between occurrence of the event that initiated this break and the emulator's entry into the monitor.
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. Several instructions may execute between occurrence of the event that initiated this break and the emulator's entry into the monitor.
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. Several instructions may execute between occurrence of the event that initiated this break and the emulator's entry into the monitor.

Examples

Display the status of all six break conditions:

```
M> bc
```

Enable breaks on writes to ROM and upon assertion of the **trig1** signal, and disable execution breakpoints and breaks generated by the **trig2** signal:

```
M> bc -e rom trig1  
M> bc -d bp trig2
```

You can independently enable or disable the six different break conditions: write to ROM, execution 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 execution of a breakpoint instruction 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

Chapter 10: Emulator Commands

bc

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.

If an error occurs during processing of the **bc** command, a particular break condition may be 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)

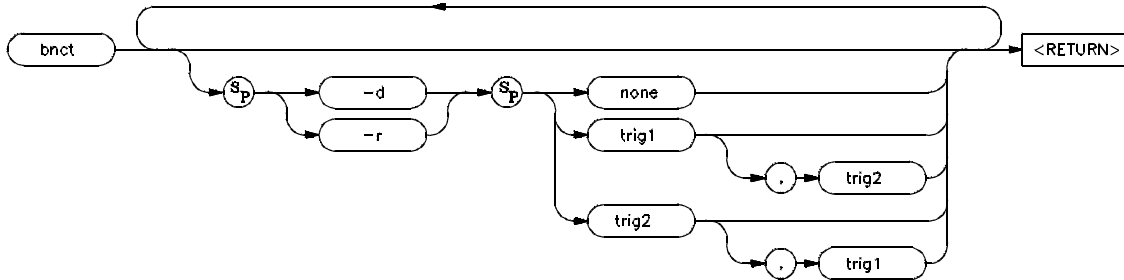
cmbt (specify drivers and receivers of the CMB trigger signal)

bp (set/delete execution 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 **bnc -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)

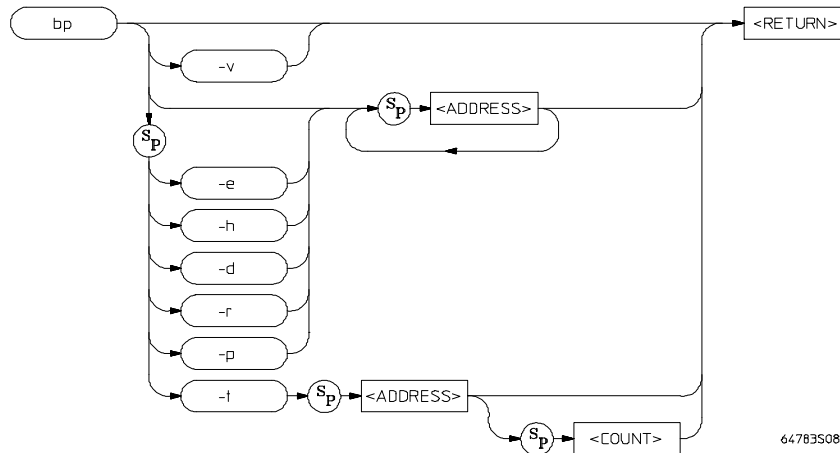
tarm (analyzer trace arm; used to specify arming (begin to search for trigger) conditions for the analyzer -- **trig1/trig2** can be used to arm the analyzer)

tgout (specifies which of the **trig1/trig2** signals are to be driven when the analyzer trigger is found)



bp

bp



64783508

The **bp** command is used to insert, delete, display, or modify the status of execution breakpoints. Upon powerup or **init** initialization, the breakpoint table is cleared and the breakpoint feature is enabled. If no parameters are specified, the current status of all breakpoints is displayed.

The parameters are as follows:

<ADDRESS>

The <ADDRESS> parameter allows you to specify the address where the execution breakpoint is to be inserted. If you specify options **-d**, **-e**, or **-r**, then the address specifies the location of the breakpoint to be deleted, enabled, or removed. For these options, you may specify the address "*" to indicate the operation is to be performed on all addresses in the 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, the breakpoint instruction will be placed in the word-aligned address.

-r

Deletes the execution breakpoint(s) at the software 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

in software (if in RAM memory), and then the address is removed from the breakpoint table.

- h Specifies treatment of the breakpoint as if it is for a ROM address.
- 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.
- p Identifies the associated breakpoint as permanent. When the breakpoint instruction is executed, the original instruction is replaced in memory so that the program can run when you return from the break. Immediately after you start the program again, the BKPT instruction is again placed at the breakpoint address so that when the breakpoint address is hit again, another break will occur. This will continue until you disable the breakpoint.
- t Identifies the associated breakpoint as temporary. When the breakpoint is executed, the original instruction is restored to the memory address so that the program will run without breaking. If you want to break at the temporary address again, you will need to reenable the breakpoint.
- v Requests a verbose display of existing breakpoints.
- d Disables (deactivates) the execution breakpoint(s) at the software address(es) specified. The breakpoints remain in the breakpoint definition table and can be enabled again by using the **bp -e <ADDRESS>** command. If the breakpoint is already disabled, no action is taken.

Examples

The following examples use the demo program.

Assume you must verify the processor is reaching the `_clocktic`, `_alarm`, and `_userbuf` addresses in the demo program. You can insert execution breakpoints at these addresses and run to each successive breakpoint. First, enable the execution breakpoint feature.

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

Now define temporary breakpoints at each address by typing:

```
M> bp -t _clocktic _alarm _userbuf
```



Chapter 10: Emulator Commands

bp

View the current breakpoint settings:

```
M> bp
```

With each run of the demo program, one of the breakpoints will be executed by the processor and will be disabled. You can reenable the breakpoints:

```
M> bp -e _clocktic _alarm _userbuf
```

You could also type **bp -e *** to reenable all breakpoints in the table. Also, you can define each breakpoint to be permanent (**bp -p <ADDRESS>**).

To disable the breakpoint at `_clocktic`:

```
M> bp -d _clocktic
```

To remove the breakpoint at `_alarm` from the breakpoint table:

```
M> bp -r _alarm
```

To disable all breakpoints in the table:

```
M> bp -d *
```

To remove all breakpoints in the table:

```
M> bp -r *
```

To disable the breakpoint feature,

```
M>bc -d bp
```

The MC68040 emulator uses the BKPT instruction to implement execution breakpoints. There are four different operations to maintain the execution breakpoint table.

Inserting Breakpoints

Specifying only an address inserts the breakpoint instruction in memory (if the memory is writeable RAM) and makes a breakpoint table entry corresponding to that address. If a software break instruction (BKPT) already exists at the address specified, no change occurs. If the breakpoint instruction cannot be written, the

emulator uses one of its eight hardware resources to remember the breakpoint address so that it can jam a BKPT instruction on the data bus when that address is read.

Enabling Breakpoints

Enabling a breakpoint at a specified software 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. If the breakpoint instruction cannot be written, the emulator uses one of its eight hardware resources to remember the breakpoint address so that it can jam a BKPT instruction on the data bus when that address is read.

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 specified 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** or **bp -v** command, the enable/disable status of each breakpoint is tested by reading the memory locations in question. If a break instruction is found in software or if one of the eight hardware resources contains a ROM address on which to break, the breakpoint is displayed as **enabled**; if not, the breakpoint is displayed as **disabled**.

Removing Breakpoints

Removing a breakpoint causes a search of the breakpoint table for a corresponding entry; if found, the original memory contents are written to the specified address if in RAM, and the entry is removed from the breakpoint table.

When a breakpoint instruction that was inserted by **bp** is executed by your program, it is removed from memory and marked **disabled** in the breakpoint table. If the breakpoint was specified to be permanent, then it will be restored to memory and marked enabled again as soon as the breakpoint address is read for an instruction fetch.

A status message is issued to **stdout** (see the **po** command) indicating that a breakpoint was found.



Chapter 10: Emulator Commands

bp

If the emulator executes a 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 target program when you define or modify breakpoints, it will break into the monitor for each breakpoint defined or modified. The emulator will return to execution of your target program after breakpoint definition or modification is complete.

In general, you should only define execution breakpoints at addresses that contain program instructions. If you set breakpoints at other locations, it is likely they will be treated as operands, never executed. The only exception to this might be when you suspect your program is jumping into a data block and attempting to execute code; setting an execution breakpoint in this area will let you verify the problem (and stop a runaway program).

Remember that any operation that 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 execution 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>**).

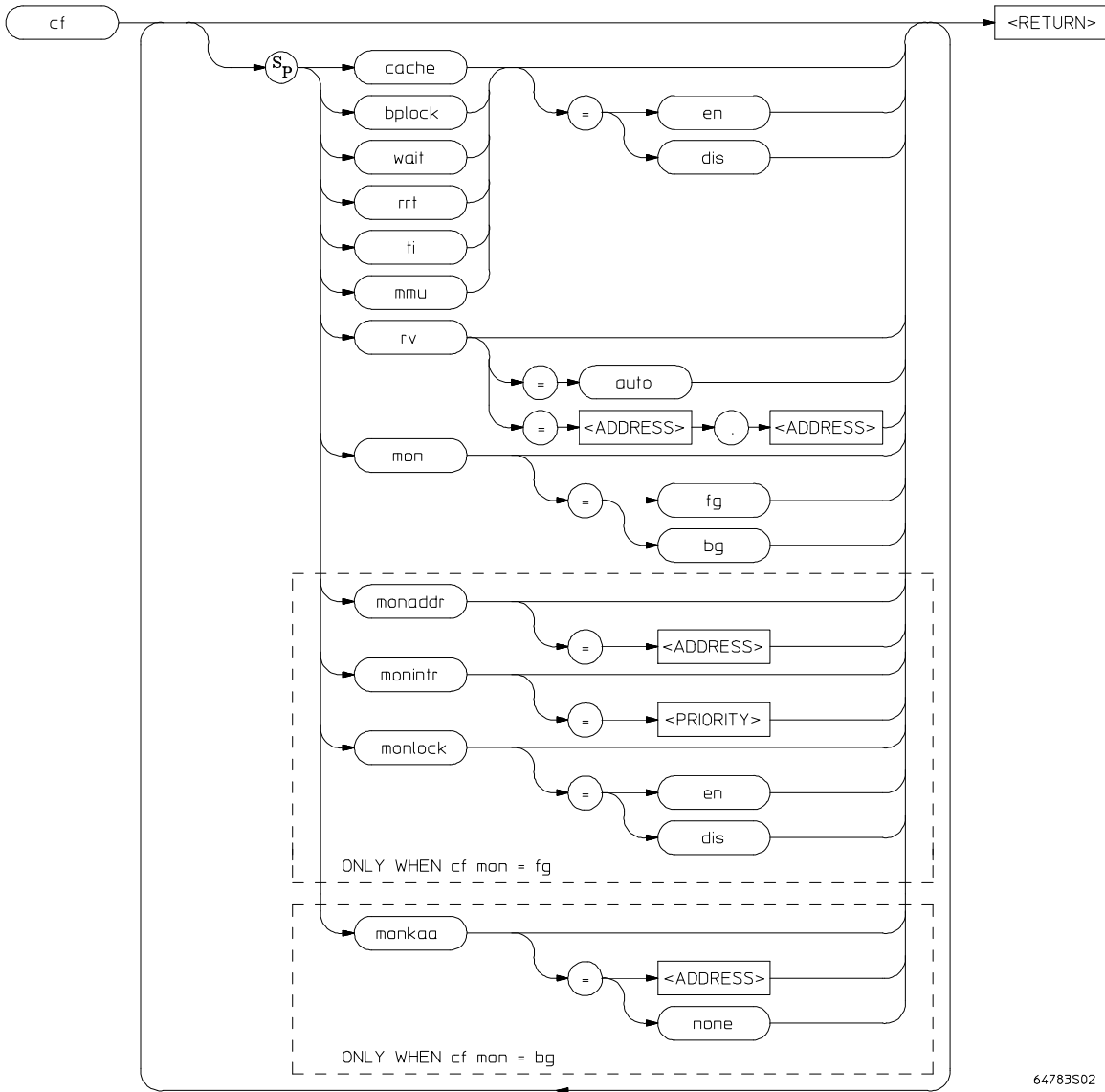
You may define an unlimited number of execution breakpoints in software contained in RAM. You may also define up to eight execution breakpoints in software contained in ROM (or RAM that is write protected). When the emulator tries to write the BKPT instruction in ROM hardware and finds, as it test-reads the breakpoint address, that the BKPT instruction is not written there, the emulator will use one of its eight hardware resources to store the breakpoint address. When that address is read during an instruction fetch, the hardware resource will jam the breakpoint instruction on the data bus so that the BKPT will be fetched by the processor.

See Also

bc (enable/disable breakpoint conditions (including **bp**))

mo (defines memory access and display modes; the **bp** command uses the currently defined modes when writing execution breakpoints into software)

cf



64783502



Chapter 10: Emulator Commands

cf

The **cf** command allows you to modify various emulator specific configuration parameters. The MC68040 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. If you enter the configuration item name without a setting, the current setting is displayed.

The parameters are as follows.

cache	Enable/disable the instruction cache and data cache.
bplck	Enable/disable interlocking of breakpoint acknowledge cycle termination. When enabled, the target system is responsible for terminating breakpoint acknowledge cycles by asserting TA or TEA. Default is disabled.
wait	Enable this if you are using a BCLK clock speed greater than 25 MHz. Disable this configuration item if you are using a BCLK clock speed less than 25 MHz and you are not using 25-ns memory modules in the emulation probe. A special "cf wait=dis,en" must be used if you are using a BCLK clock speed less than 25 MHz and you are using HP 64173A (25-ns) memory modules in the emulation probe.
monlock	Enables $\overline{\text{TA}}$ and $\overline{\text{TEA}}$ interlocking between the emulator and target system for foreground emulation monitor bus cycles.
rrt	Restricts the emulator to real time runs.
ti	Enable/disable target system interrupts.
mmu	Enables or disables the MMU. If enabled, the $\overline{\text{MDIS}}$ line and TC register determine whether or not logical-to-physical address translations are performed. If disabled, the emulator asserts the $\overline{\text{MDIS}}$ line to prevent address translations.
mon	Chooses a foreground or background monitor. Before you can enable the MMU, or the caches, or do dma activity in your target system, you must select a foreground monitor.
monaddr	Sets the base address for the foreground monitor (and maps a corresponding memory block). Not available when the background monitor is selected.
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 when the background monitor is selected.

monkaa Defines an address from which the background monitor periodically reads a byte. Used to “keep-alive” circuits that depend on constant bus activity, such as watchdog timers.

Examples

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 MC68040 emulator after initialization is as follows:

- Configuration items:

```
R>cf
```

```
cf cache=en  
cf bpllock=dis  
cf mmu=en  
cf mon=fg  
cf monaddr=0  
cf monintr=0  
cf monlock=en  
cf rrt=dis  
cf rv=auto  
cf ti=en  
cf wait=dis
```

- One memory map term assigned to contain the emulation monitor. All other memory mapped to target RAM (**tram**).
- All break conditions (**bc**) are enabled.

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.

Chapter 10: Emulator Commands

cf

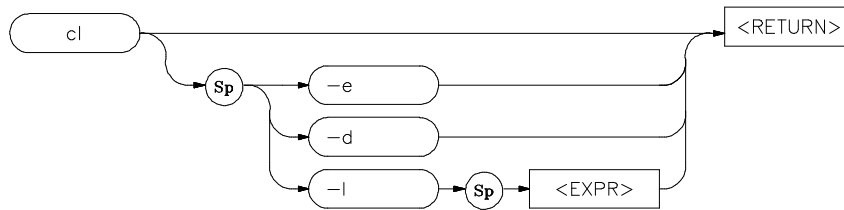
See Also

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 the chapter titled "Configuring the Emulator" in 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.

Chapter 10: Emulator Commands

cl

The commands in control mode are as follows:

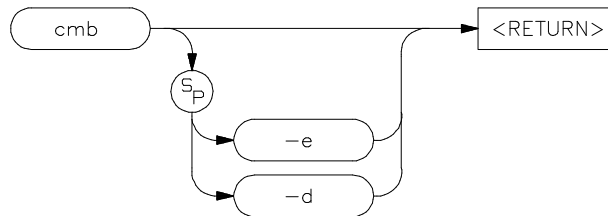
Command	Action
i	Insert before current character.
A	Append to end of line.
dd	Delete command line.
\$	Move cursor to end of line.
^	Move cursor to start of line.
l	Move right one character.
j	Fetch next command.
a	Insert after current character.
x	Delete current character.
D	Delete to end of line.
0	Move cursor to start of line.
h	Move left one character.
k	Fetch previous command.
r	Replace current character.
/ <code><string></code>	Find previous command matching <code><string></code> .
n	Fetch previous command matching <code><string></code> .
N	Fetch next command matching <code><string></code> .



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

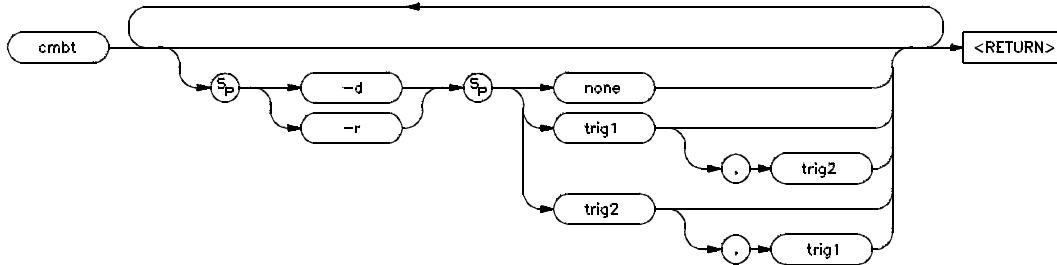
rx (allows you to specify the starting address for user program execution when the CMB /EXECUTE line is asserted)

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**.

Chapter 10: Emulator Commands

cmbt

Examples

To view the current **cmbt** setting, type:

```
M> cmbt
```

Trigger the analyzer in another 68040 emulator connected to the CMB:

```
M> tcf -e  
M> tg addr=100  
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=100  
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=200  
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 vector table to a base address of 20000:

```
M>cp 20000=0..3ff
```

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

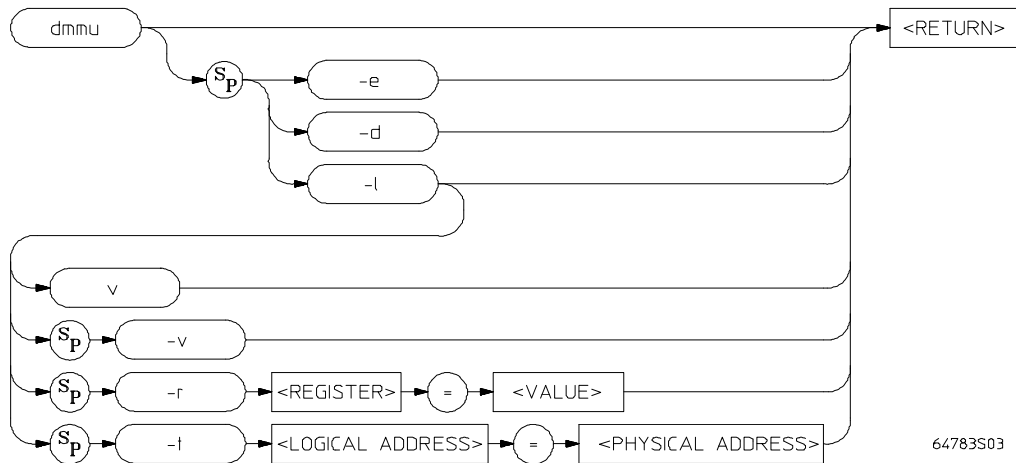
m (allows you to display or modify memory locations or ranges)

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)



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.

The deMMUer can translate up to 256 Mbytes of physical address space.

The parameters are as follows:

- c Clears all reverse translation information in the deMMUer.
- 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.

- r Allows you to specify a value for one or more of the MMU registers. The value is to be used instead of the register's present value when loading the deMMUer with information to reverse MMU translations.
- t Prepares the deMMUer to accept reverse-translation information from the command line. This allows you to enter desired reverse translations on the command line.
- 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 address qualifications, they are shown beside the addresses (example: 000000000..003ffffff@s).

Examples

To enable the deMMUer to translate addresses for the analyzer:

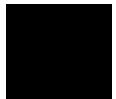
```
M> dmmu -e
```

To load the deMMUer to reverse translate addresses using the current translation tables:

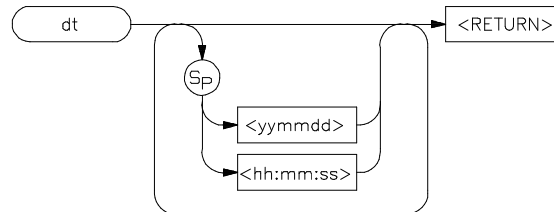
```
M> dmmu -l
```

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:

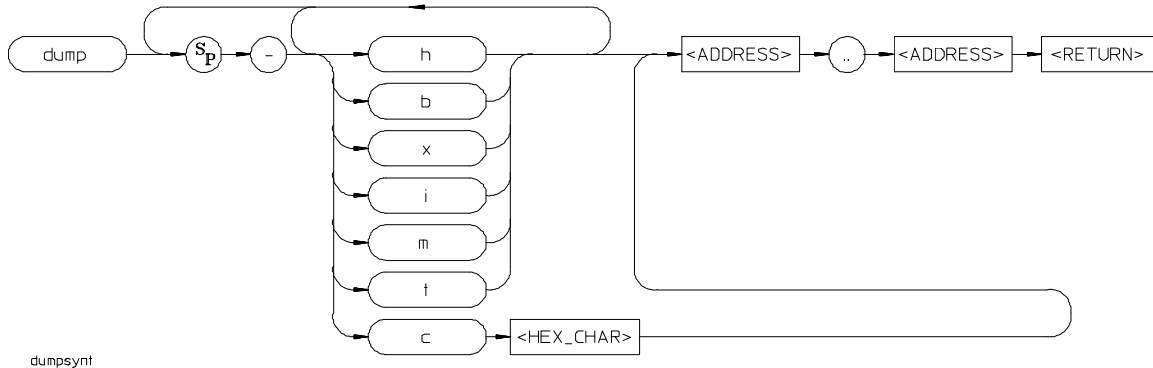
```
M> dt 910818
```

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.

Chapter 10: Emulator Commands

dump

<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.

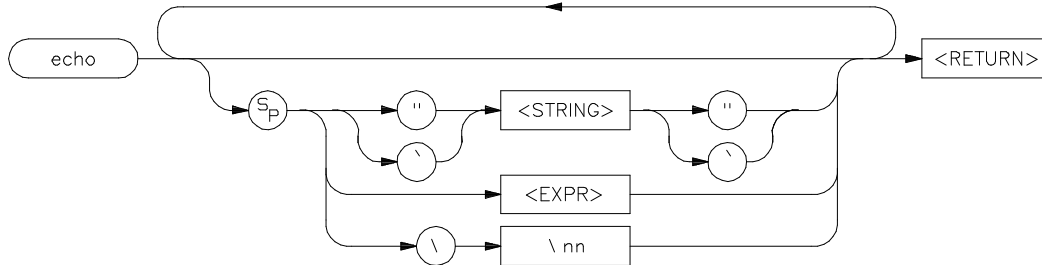
The HP 64000 format “.X” file created with a “dump -hox” command has records that contain 136 fewer bytes of data than the file format standard allows. Because of this, HP 64000 format “.X” files which are created with the **dump** command may take longer to be processed by consumers of the “.X” file (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.

Chapter 10: Emulator Commands

echo

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)

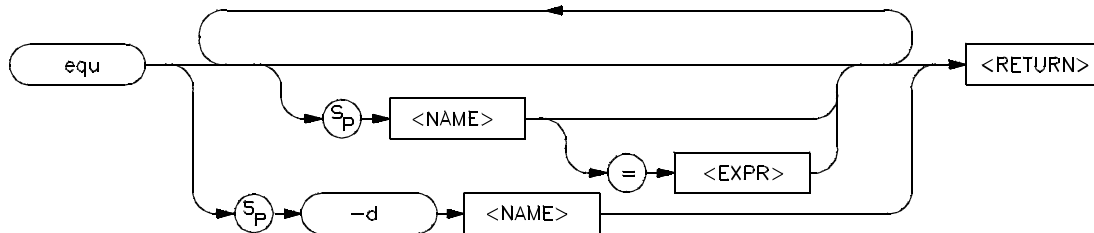
mac (grouping a set of commands under a label for later execution)

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.

MC68040 Equates

Name	Value	Description
ack	11xxxxxxxx1xxxxxy	Acknowledge access.
alt0	10xxxxxxxx1x001xy	Alternate logical function code 0.
alt3	10xxxxxxxx1x011xy	Alternate logical function code 3.
alt4	10xxxxxxxx1x100xy	Alternate logical function code 4.
alt7	10xxxxxxxx1x111xy	Alternate logical function code 7.
burst	0xxxxx0xxxxxxxxxy	Burst cycle.
byte	0xxxxxx01xxxxxxxxxy	Byte transfer request (SIZ1/SIZ0=01).
cpush	0xxxxxxxx1x000xy	Data cache push access.
d_tblwk	0xxxxxxxx1x011xy	Data translation table access.
data	0xxxxxxxx1xx01xy	Data space access.
dma	0xxxxxxxx0xxxxxy	Direct memory access.
i_tblwk	0xxxxxxxx1x100xy	Instruction translation table access.
line	0xxxxx11xxxxxxxxxy	Line transfer request (SIZ1/SIZ0=11).
logical	0xx0xxxxxxxxxxxxxy	Logical memory address.
long	0xxxxxx00xxxxxxxxxy	Longword transfer request (SIZ1/SIZ0=00).
physical	0xx1xxxxxxxxxxxxxy	Physical memory address.
prog	0xxxxxxxx1xx10xy	Program space access.
read	0xxxxxxxxxxx1xxxxy	Read cycle.
retry	0xxxxxxxx00xxxxxy	Retrying a previous bus cycle.
snp_hit1	0xx01xxxxx0xxxxxy	Snoop operation 1 (SC1/SC0=01)
snp_hit2	0xx10xxxxx0xxxxxy	Snoop operation 2 (SC1/SC0=10)
snp_inhb	0xx00xxxxx0xxxxxy	Snooping inhibited.
snp_miss	0xx11xxxxx0xxxxxy	Snoop miss.
sup	0xxxxxxxx1x1xxxxy	Supervisor space.
supdata	0xxxxxxxx1x101xy	Supervisor data space.
supprog	0xxxxxxxx1x110xy	Supervisor program space.
ta	0xxxxxxxx10xxxxxy	Transfer acknowledge.
tea	0xxxxxxxx01xxxxxy	Transfer error acknowledge.



Chapter 10: Emulator Commands

equ

upa0	0xx00xxxxxxxxxy	User prog attributes UPA[1:0]=00.
upa1	0xx01xxxxxxxxxy	User prog attributes UPA[1:0]=01.
upa2	0xx10xxxxxxxxxy	User prog attributes UPA[1:0]=10.
upa3	0xx11xxxxxxxxxy	User prog attributes UPA[1:0]=11.
user	0xxxxxxxx1x0xxxxy	User space.
userdata	0xxxxxxxx1x001xy	User data space.
userprog	0xxxxxxxx1x010xy	User program space.
word	0xxxxx10xxxxxxxxxy	Word transfer request (SIZ1/SIZ0=10).
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 `update_sys:write_hdwr`. You would like to have each trace numbered.

Enter the following commands:

```
M> tg addr=update_sys:write_hdwr
M> equ c=0
M> mac numtrclist={t;w -m;equ c=c+1;echo "trace # "
c;t1}
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 `c` has been removed. Now type:

```
M> equ -d *
```

This removes all equates, including the system-defined equates. You can restore the system-defined equates by initializing the emulator (`init`).

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.

Chapter 10: Emulator Commands

equ

The HP 64783A/B emulator predefines 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 (equate 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 $\overline{\text{TA}}$ from target system
- bus grant to the target system

The exact messages and information displayed varies slightly, depending on the emulator in use.

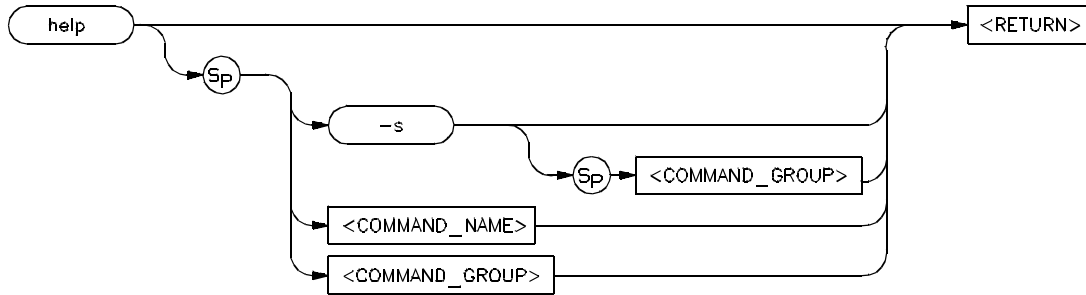
The emulator will not break to the monitor to obtain information. Therefore, any information that can only be obtained while in the monitor will not be displayed if the emulator is not in the monitor.

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**

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:

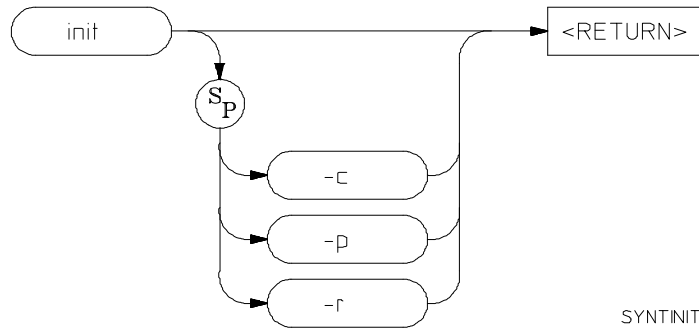
```
M> help emul
```

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. If you are using the emulator through a LAN connection, this option will break your LAN connection. Use the `-c` option to avoid breaking the LAN connection.
- 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.
- r The **-r** option causes a powerup initialization sequence. This initializes the operating system and data communications, and ignores other optional products. It also ignores the emulator and analyzer. It is primarily used by the progflash feature to update firmware.

Examples

Perform a powerup initialization sequence:

```
M> init -p
```

If you have used telnet to connect to the emulation card using LAN, your connection will be broken by this initialization sequence.

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)

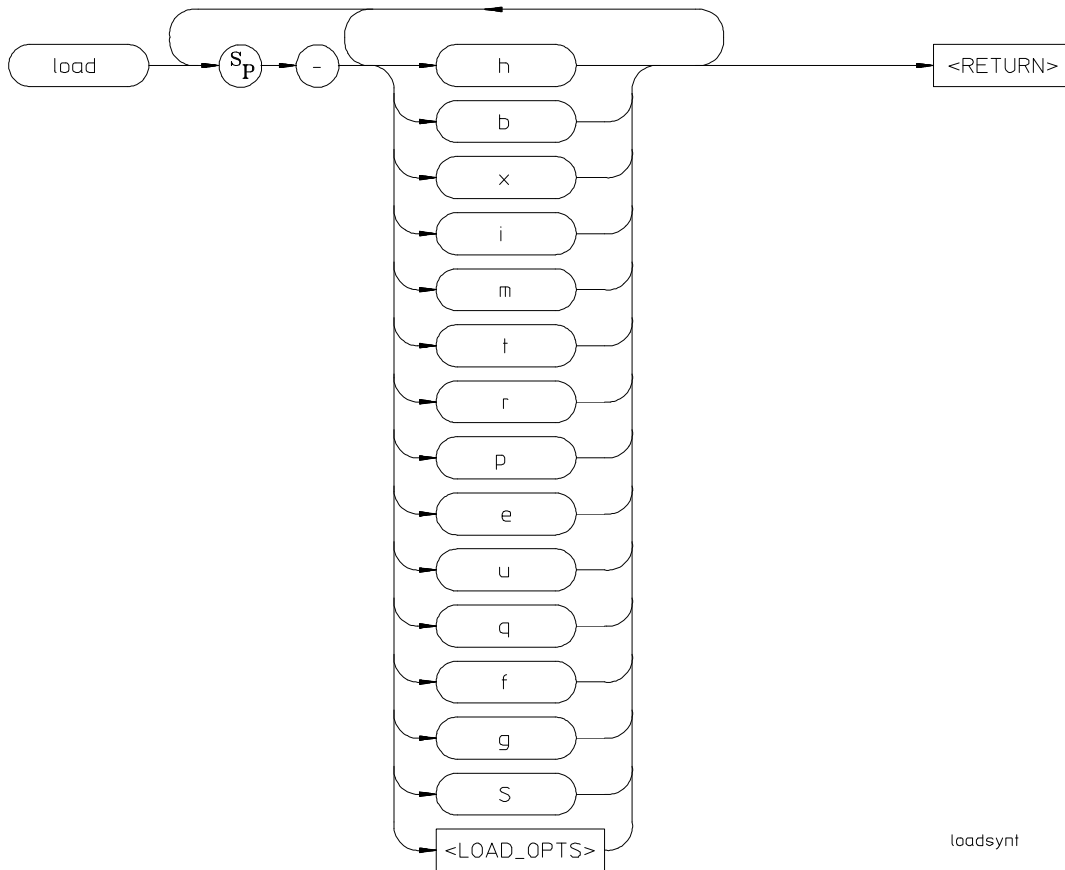
map (define the emulation memory map)

stty (set data communications parameters)

tinit (reset the analyzer to powerup defaults)



load



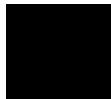
The **load** command lets you load program code into emulation or target memory. Various file formats are supported via options to the load command.

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 “#” 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 MC68040 emulator does not support any custom load options.
- <FILE> This represents the absolute file to load into the emulator.
- 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.



Chapter 10: Emulator Commands

load

-f You specify the **-f** option if you are loading a custom foreground monitor into the emulator.

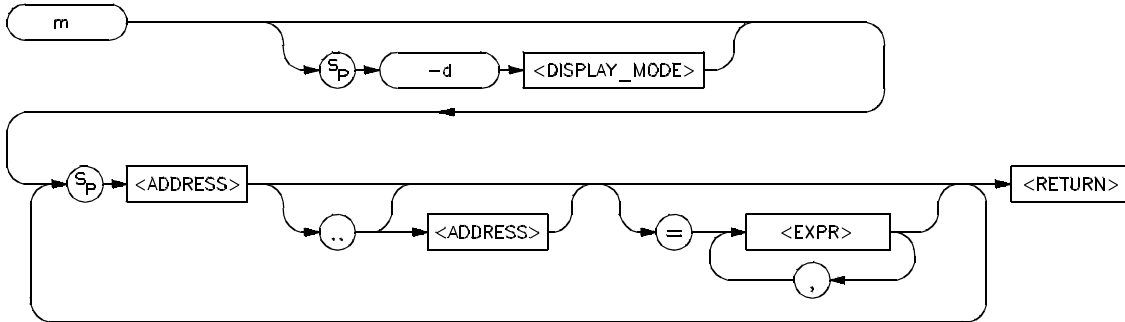
In the default, at least one file format option must be specified.

See Also

See Chapter 4 for instructions on loading programs using different communications configurations.



m

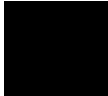


The **m** command allows you to display and modify emulation and target system memory. Options allow you to specify the display mode, specific address or addresses for display or modification, and the data values to be inserted.

The parameters are as follows.

- d The **-d** option allows you to set the display mode for memory accesses.
- <DISPLAY_ A one-character mnemonic specifying the display mode to use in creating memory
MODE> 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
MC68040 emulator allows the use of function codes and the distinction between
logical and physical when specifying address information. However, 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.



Chapter 10: Emulator Commands

m

<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 _sys_demodisp.._sys_demointr
```

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 the monitor upon execution of the command. After the command is complete, the processor will be returned to target program 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 MC68040. 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 with your MMU translations, 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.

Chapter 10: Emulator Commands

m

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.

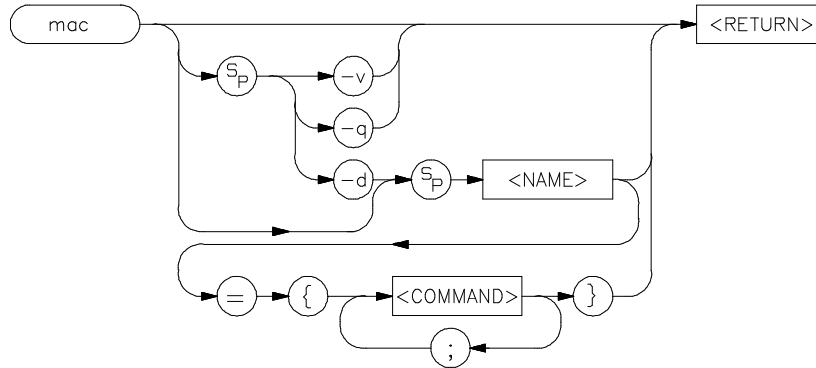
If symbols have been defined, either by loading a symbol file or by using the **sym** command, these symbols can be used in the **m** command and will appear in the mnemonic mode (**-dm**) memory display. The command processor retains the name of the last module referenced. If a symbol does not contain 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)

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 (;).

Chapter 10: Emulator Commands

mac

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.

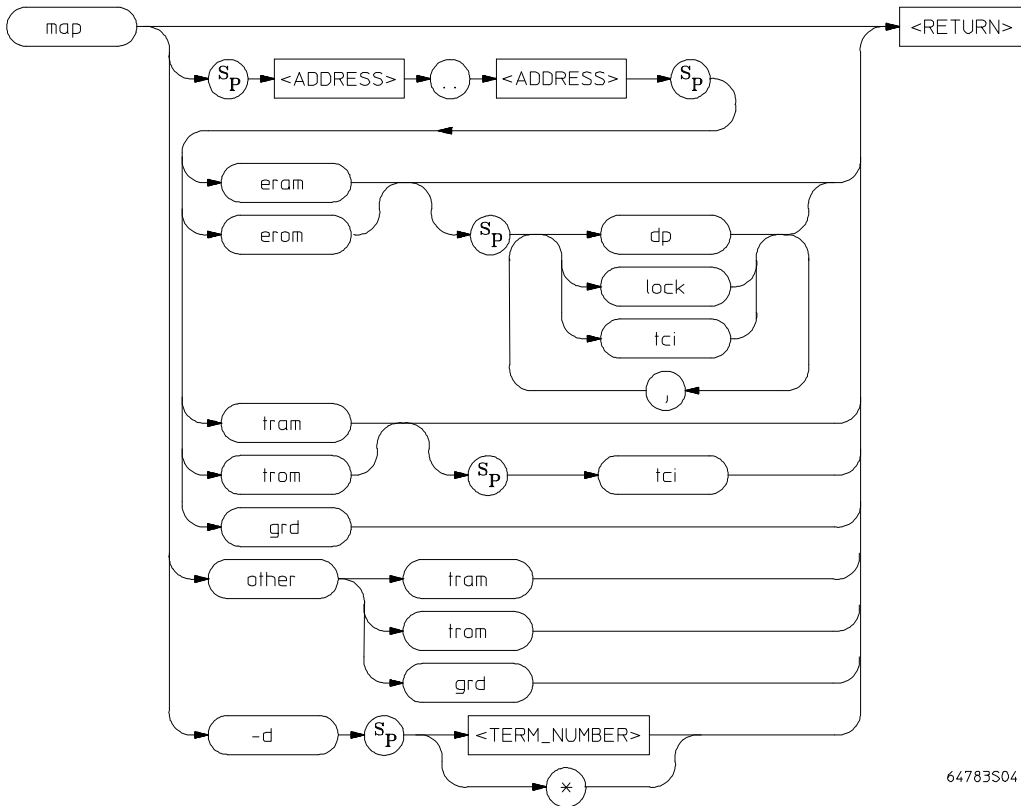
Chapter 10: Emulator Commands

mac

See Also **rep** (repeat; allows you to repeat any command, including macros)



map



64783S04

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

Chapter 10: Emulator Commands

map

information defaults to a hexadecimal value. 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 emulator restricts 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 does not protect emulation memory from being modified when a write to emulation ROM occurs.

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.

lock

Interlock target system and emulation \overline{TA} and \overline{TEA} signals (only valid for erom and eram blocks).

tci

Inhibit caching for this memory block.

If the command **map** is entered with no parameters, the current memory map is displayed.

Examples

View the memory map:

```
M> map
```

Suppose you need to map the following ranges:

- 1000 through 1fff, using dual-port emulation RAM and interlocking transfer termination signals with the target system.
- 2000 through 2fff using emulation RAM and interlocking transfer termination signals with the target system.
- 5000 through 7fff using target RAM and inhibiting caching.
- 8000 through 8fff using target ROM.
- All other memory is mapped as guarded.

Implement this map by entering

```
R> map 1000..1fff eram dp,lock
R> map 2000..2fff eram lock
R> map 5000..7fff tram tci
R> map 8000..8fff trom
```

Delete all map terms (reset the map):

```
R> map -d *
```

The emulation system assigns a term number to each address range you specify 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 256 bytes. A map term can represent a block as small as 256 bytes or as large as the memory space available in your system.

Chapter 10: Emulator Commands

map

When you create a map term, it will be rounded upwards to occupy the minimum number of 256-byte blocks required to contain the term.

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 map 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

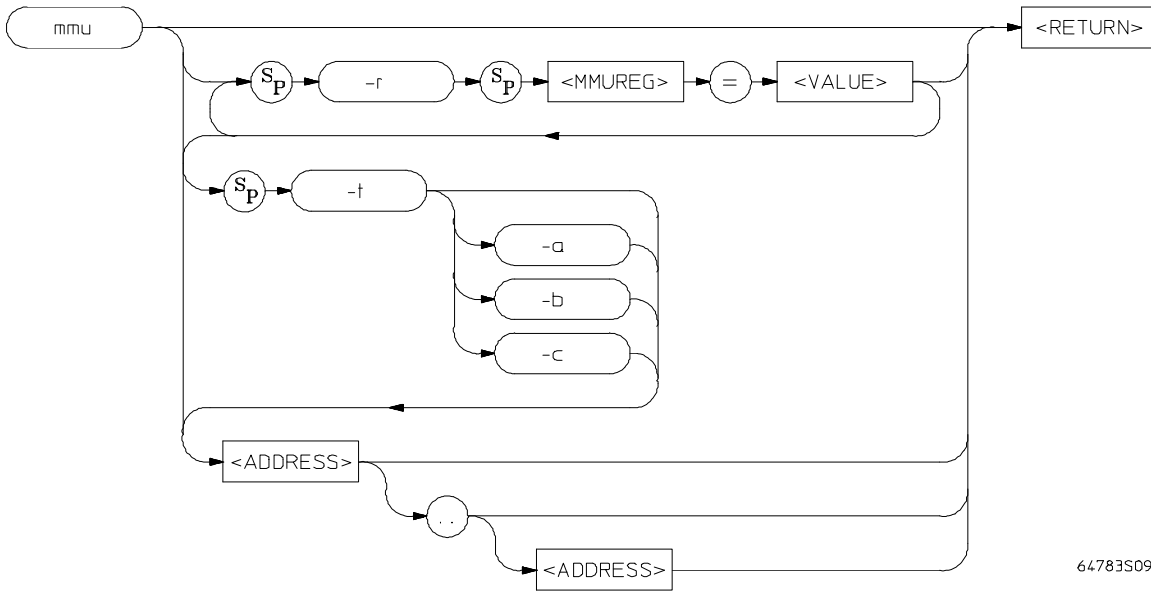
bc (break conditions; determines whether emulator breaks to monitor upon write to space mapped as ROM)

m (memory display/modify)

bp (set/delete execution breakpoints)

Chapter 7, “Configuring the Emulator,” has a complete description of the block allocation strategy used for the emulation memory resources.

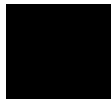
mmu



64783S09

The **mmu** command is used to display valid logical-to-physical address translations. You can display 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 one or more of the MMU registers is invalid. Parameters in the **mmu** command let you specify values to use when none exist in the MMU registers.



Chapter 10: Emulator Commands

mmu

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.
- <MMUREG> The name of a register in the set of registers used by the MMUs (mmusr, tc, srp, urp, itt0, dtt0, itt1, dtt1).
- r Lets you specify the name of any MMU register and supply a value to be used when composing MMU displays, instead of using the present content of the register.
- t Shows the details of the translation through the tables for the logical address you included in your command.
- <VALUE> A number to be used in place of the present value within the referenced MMU register. 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 SRP register value of 00102000 instead of the present SRP register value, and mTT0 register values of 0:

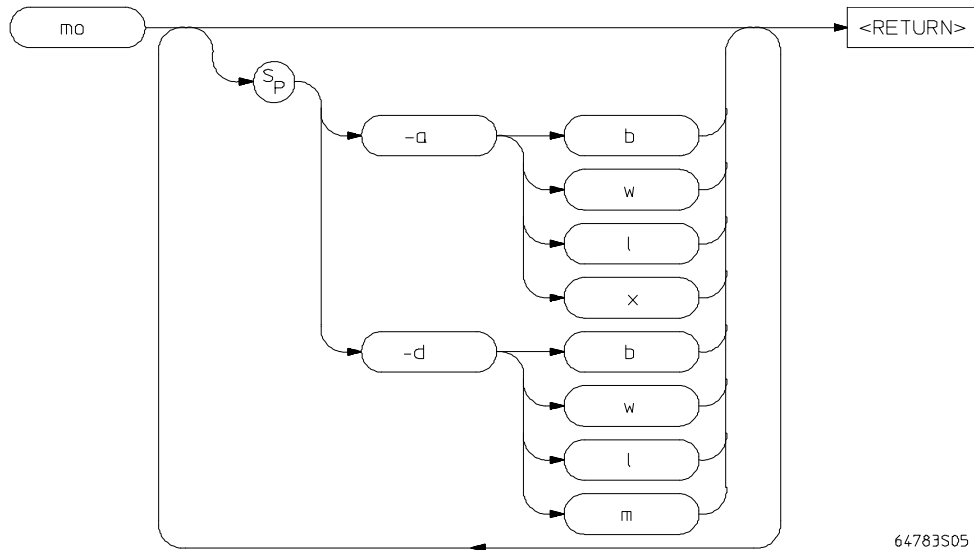
M> mmu -r srp=00102000 -r itt0=0 -r dtt0=0

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 your 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 MC68040 emulator allows the following access modes:

l long word (four bytes) access mode

w word (two bytes) access mode

b byte access mode

x emulator selects the optimum access mode (this is the default).

-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 MC68040 emulator allows 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 `_sys_startup` of the demo program in mnemonic format, enter:

```
M> m -dm _sys_startup
```

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:



Chapter 10: Emulator Commands

mo

```
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.

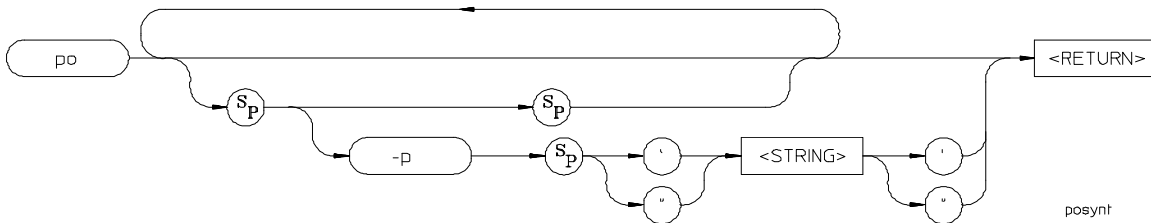
The emulation monitor uses the access mode to determine whether to use byte, word or long word instructions during accesses to target system memory and emulation memory that is not dual-port. It does *not* affect how that data is displayed on screen, or the way in which data is interpreted for memory modification. That is controlled by the display mode.

By default, the emulator selects the most efficient mode for the access. You will only want to specify use of byte, word, or long word if you must ensure that the specified access mode is used due to restrictions in your memory hardware.

See Also

m (memory display/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:

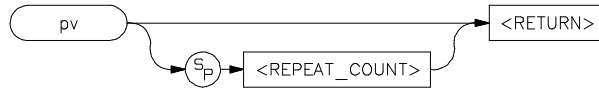
```
M> mac yourid={po -p "\YOURID>"}
M> mac herid={po -p "\HERID>"}
M> mac hisid={po -p "\HISID>"}

```

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

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 plugged into the demo board.

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.

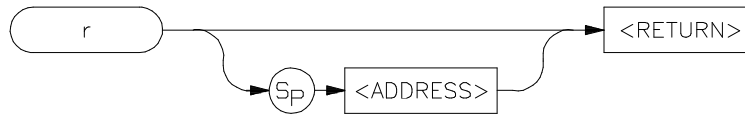
See Also

init (reinitializes the emulator)



Chapter 10: Emulator Commands
pv



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 the emulator is already reset, an **r** command is the same as **r rst**.

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 rst;b
```

Run the processor from address 700 (the entry address) in the demo program:

```
R> r 700
```

See Chapter 4 for information on how the emulator handles an **r rst** command.

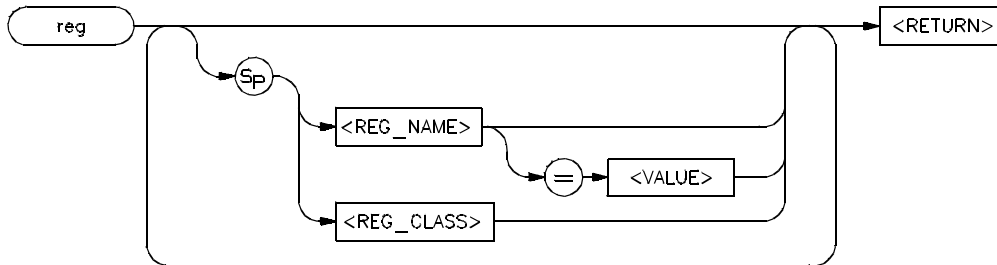
See Also

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	fpcr, fpsr, fpiar, fp0..fp7
mmu	itt0, dtt0, itt1, dtt1, mmusr, tc, srp, urp

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:

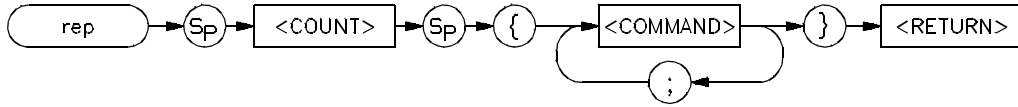
M> **reg d0 st**

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}
```

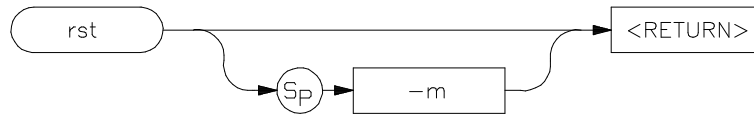
Command macros that you define using the **mac** command can be used within a command group for repetition.

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
```

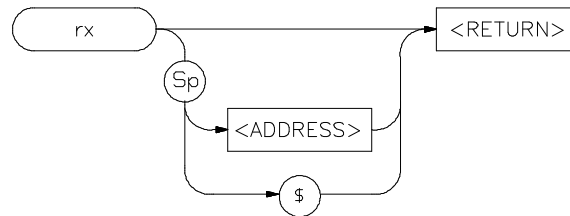
Reset the processor and have it immediately commence emulation monitor execution:

```
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, but will first break from emulation into the monitor; these include commands such as **reg**.

Commands or hardware signals which will take the emulator out of a reset state include **b**, **r**, **s**, **reg**, **m**, etc. 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 700 when the CMB-EXECUTE pulse is received:

```
M> rx 700
```

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.

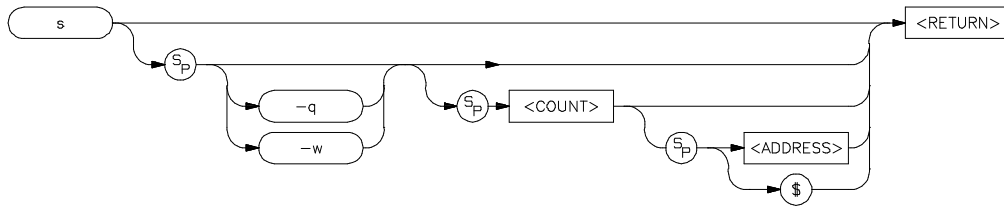
See Also

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 MC68040 emulator allows 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 entry:

```
M> s 1 entry
```

You can step through two additional instructions in the program by typing:

```
M> s 2
```

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
```

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
```

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 when specifying step information. For example, you could step 20 instructions from entry using the following equate. Type:

```
M> equ readcount=20  
M> s readcount entry
```

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.

Chapter 10: Emulator Commands

s

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.

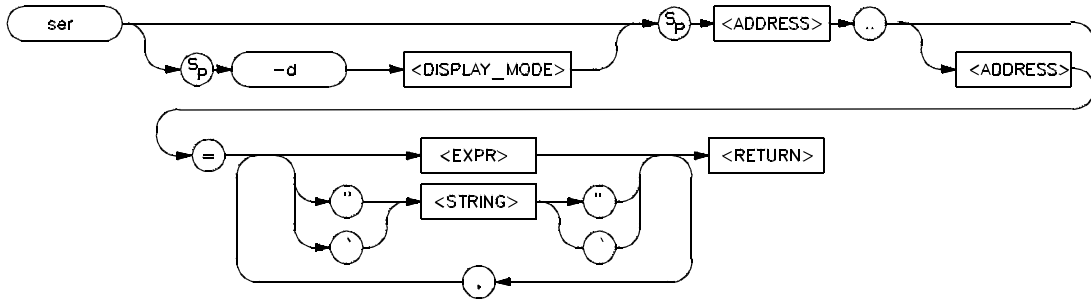
If you have loaded a symbol file or user defined symbols, you will see the module and symbol in the output when an instruction address has a corresponding symbol.

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 MC68040 emulator supports **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 MC68040 emulator allows 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.

Chapter 10: Emulator Commands

ser

<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 a particular address range:

```
M> ser 0..0ffff="This"
```

You can also combine searches for numeric values, numeric expressions, and ASCII strings:

```
M> ser -db 0..0ffff=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 (,).

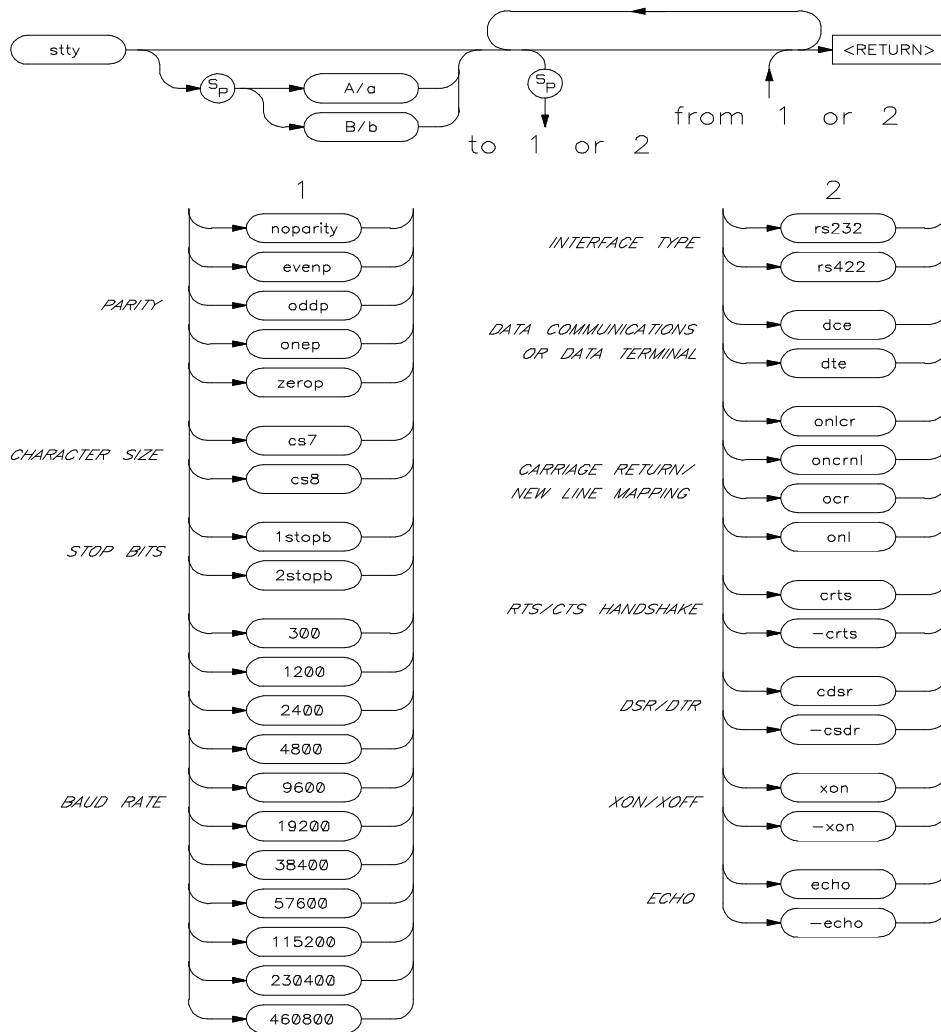
See Also

cp (used to copy the contents of one memory range to another)

m (used to display/modify memory locations)



stty



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 COMMUNICATIONS 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.



Chapter 10: Emulator Commands

stty

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
```

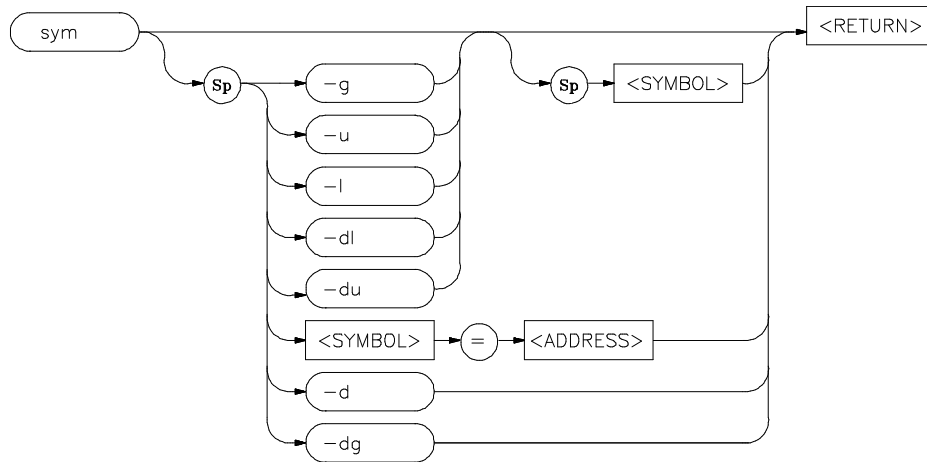
Change the baud rate back to 9600 and disable local echo on the Serial Port:

```
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.

Chapter 10: Emulator Commands

sym

<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 `_sys_demodisp`:

```
M>sym -g _sys_demodisp
```

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 a local module named `handle_msg` (note the demo program does not contain this module. It is here for example, only):

```
M>sym -l handle_msg:
```

Delete all global symbols:

```
M>sym -dg
```

Display all symbols or local modules whose names begin with “_sys_”:

```
M>sym _sys_*
```

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)

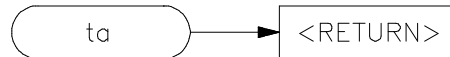
th (halts a trace in process)

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



When used in the 1K analyzer, the **ta** command displays activity on each of the analyzer input lines. Each signal may be low, high, or moving. There are no parameters. The **ta** command can be accepted by the deep analyzer, but it will provide no information.

Examples

Display current signal activity in the 1K analyzer:

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.

Chapter 10: Emulator Commands
ta

The trace activity measurement is interpreted as shown in the following table.

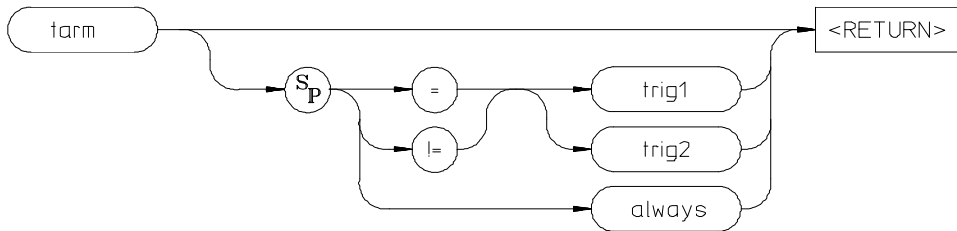
Type of signal activity	Symbol displayed
Signal is low	0
Signal is high	1
Signal is moving	?

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
```

Chapter 10: Emulator Commands

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

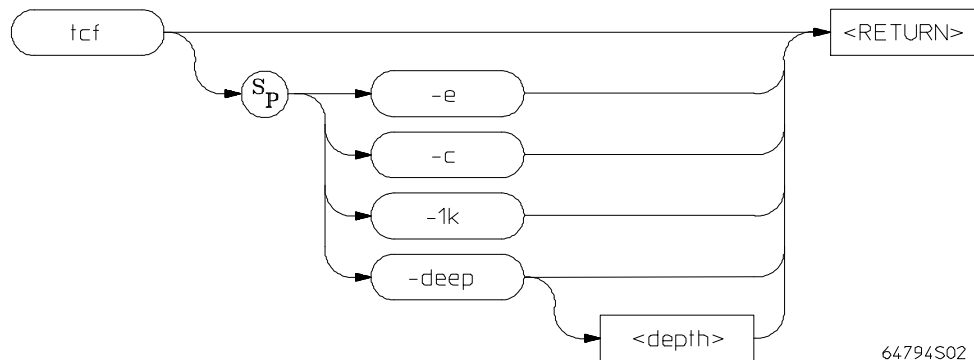
bc (can be used to cause the emulator to break to monitor execution upon receipt of the trig1 and/or trig2 signals)

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 **-1k** and **-deep** parameters are available only when using the deep 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.
- 1k Specifying **-1k** (available only in the deep analyzer) sets the analyzer to the compatible mode. In this mode, the deep analyzer is compatible with emulator interfaces that were designed to work with the 1K analyzer. The trace memory is 1024 states deep if you have no state or time count included in your trace specification; it is 512 states deep if a state or time count is included in your trace specification. This is the default mode at power up.
- deep [**<depth>**] Specifying **-deep** (available only in the deep analyzer) sets the deep analyzer to the deep mode; the trace memory depth will be the maximum depth available in the analyzer. Specifying **-deep <depth>** sets the analyzer to have a trace memory of the depth you specify (less than the maximum depth, if desired). You may want to specify a reduced depth when making a series of traces to be sent to post-processing software.

Specify **<depth>** using a decimal number. For example, to obtain the same depth as the 1K analyzer, enter the **tcf -deep 1024** command. This gives you the same trace depth as the 1K analyzer without imposing the memory tradeoff for counting that is imposed in the 1K analyzer (**tcf -1k**).

Chapter 10: Emulator Commands

tcf

When using the terminal interface the -1k mode is selected during powerup. You must select the **-deep** mode after power up by entering the **tcf -deep [depth]** command. In high-level interfaces, the **-deep** mode is automatically selected after power up.

If no parameters are supplied, the current analyzer configuration is displayed. After powerup, the default analyzer configuration is **tcf -e** and **tcf -1k**. The **tinit** command will set **tcf -e**, but will not affect the **tcf -1k/-deep** specification.

Examples

Display the current analyzer configuration:

```
M> tcf
```

Set the analyzer to complex configuration:

```
M> tcf -c
```

Set the analyzer for a 4K memory depth:

```
tcf -deep 4096
```

If no parameters are supplied, the current analyzer configuration is displayed. After powerup or **tinit**, the default analyzer configuration is **tcf -e**.

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 1K analyzer has a two-stage prestore pipeline; it can display up to two prestore states before each store state. The deep analyzer has a single-stage prestore pipeline; it can display only one prestore state before each store state. 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 the next store-qualified event is found (matching the **tsto** qualifier), the pipeline is flushed, placing the prestore-qualified event(s) immediately before the corresponding store-qualified event. You can use the prestore feature to observe the relationships between certain program variables and program routines or between program routines. (For example, you might store-qualify writes to a variable. Using prestore, you can capture the instructions that caused those writes to occur. You might store-qualify entry to a code module, and prestore qualify calls to that module.)

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)

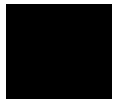
tpat (used to label complex analyzer expressions with a pattern name; the pattern name is then used by the analyzer setup commands. Only valid in complex configuration)

tpq (specifies trace prestore qualifier in either analyzer configuration)

trng (defines a range of values to be used in complex analyzer expressions)

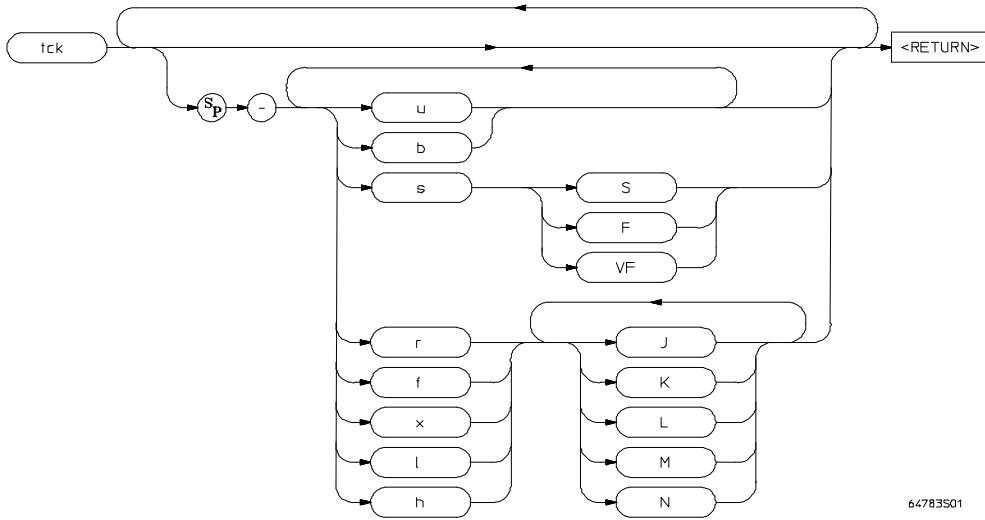
tsto (specifies a qualifier to be used when storing analyzer states)

tsq (used to modify the trace sequencer’s number of terms and trigger term)

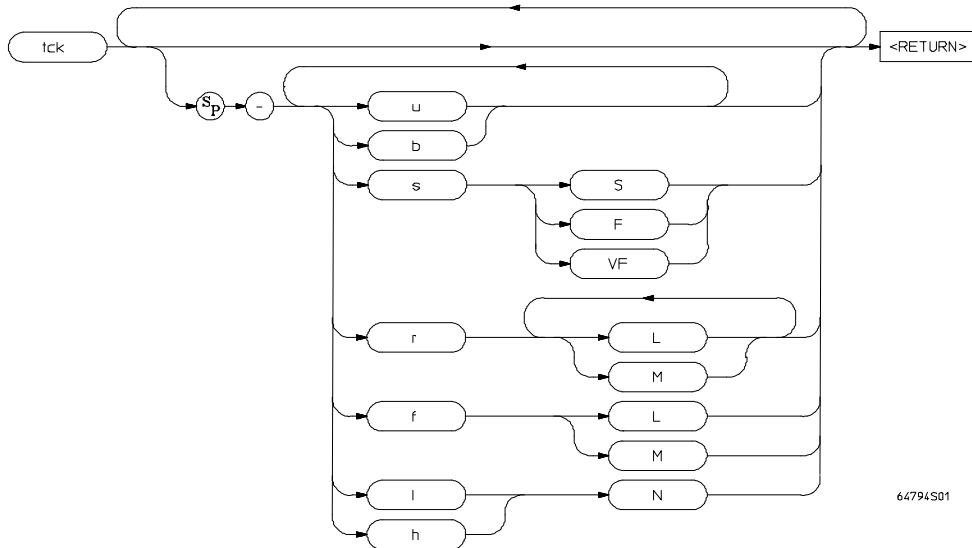


tck

The tck command used in the 1K analyzer:



The tck command used in the deep analyzer:



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 clock speed selection has no effect in the deep analyzer.

The parameters are as follows:

- b** If the **b** option is specified, only background monitor code will be qualified to be captured by the analyzer.
 - u** If the **u** option is specified, only user code (your target program code) will be qualified to be captured by 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 for the 1K analyzer is to be modified per a one or two letter code immediately following.
 - S** Specifies a bus speed of SLOW; less than or equal to 16 MHz.
 - F** Specifies a bus speed of FAST; between 16 MHz and 20 MHz.
 - VF** Specifies a bus speed of VERY FAST; greater than 20 MHz.
 - r** Specifying **r** indicates that the analyzer is to be clocked on the rising edge of the indicated clock signal. If you specify both clocks in the deep analyzer, the rising edge of either the **L** or the **M** clock can clock a state.
 - 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 accept other clock signals when the associated clock signal is low (less positive/more negative voltage). Used as a qualifier (example: clock on rising edge of **L** only if **N** is low). Note that **-l N** is the same as **-b**, which captures only background monitor execution.
 - h** Specifying **h** indicates that the analyzer should only accept other clock signals when the associated clock signal is high (more positive/less negative voltage). Used as a qualifier (example: clock on falling edge of **M** only if **N** is high).
- CLOCK SIGNALS** In the deep analyzer, the **l** and **h** operators can be used on clock signal **N**; the **r** and **f** operators may be used on clock signals **L** and **M**. In the 1K analyzer, the **r**, **f**, **x**, **l**, and **h** operators may be used on the following clock signals: **J**, **K**, **L**, **M** or **N**.

Chapter 10: Emulator Commands

tck

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 you are using the 1K analyzer, the target system clock rate for the MC68040 processor is 40 MHz, and **cf wait=en**. The resulting data rate is 20 MHz, so you must set the clock rate to fast, and disable the 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
```

Specify that trace data can be clocked into the analyzer on either the rising edge of the **L** clock or on the rising edge of the **M** clock, but only when the **N** clock is low:

```
R> tck -r LM -l N
```

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 1K 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 of the 1K analyzer operate on five different clock signals: **J**, **K**, **L**, **M** and **N**. In the deep analyzer, only three clock signals are used. 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; you should not change them.

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; the trace is clocked when one or more of the qualifiers is 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.)

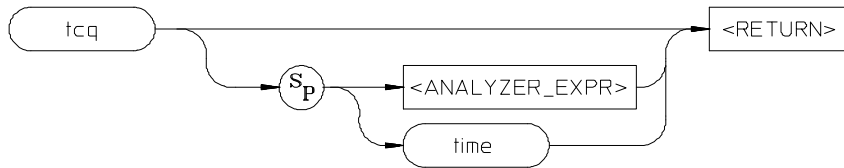
tcq (used to specify trace count qualifier for states, time, or none; maximum clock speed set in **tck** affect which **tcq** parameters are valid)

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 the type of count to be made by the emulation trace tag counter, and specify a qualification for the counter, if desired. Using this command, you can specify whether the analyzer measures time between each state it captures, counts occurrences of certain types of states you specify, or makes no count at all. If using the deep analyzer, you can also specify that counts be made only when the **arm** signal is asserted. If using the 1K analyzer, **tcq arm** is not permitted in any configuration.

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).

time

If you specify **time** rather than an analyzer expression, the trace tag counter measures the amount of time between stored states.

In the 1K analyzer, 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.

arm

In the deep analyzer, if you specify **tcq arm**, the trace tag counter will make its counts only after the **arm** signal is true. The **arm** signal can be supplied on either trig1 or trig2, and can be asserted by either the analyzer itself, by an associated emulator or analyzer on the coordinated measurement bus, or by an instrument connected to the rear panel BNC on the instrumentation card cage.

The **arm** signal begins in the false state. It switches to the true state and remains true after the first false-to-true transition of the selected signal(s).

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 change the trace listing so that the time intervals are displayed as a value relative to the last state stored, type:

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

When you display the trace list, asterisks will be shown in the **count** field if your clock rate is slow enough to allow time or state counts.

To count time intervals, do the following:

```
R>tck -ss  
R>tcq time
```

To measure the time between each character output by the demo program:

```
R>tck -ss;tcq time  
R>tsto addr=_sys_demodisp  
R>t  
R>demo;r;m _sysbuf="12345@"  
U>th; t1
```

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
```

To specify that counts be made in the deep analyzer only after an external instrument provides a false-to-true transition to the rear panel BNC, type:

```
R> bnct -d trig1  
R> tarm =trig1  
R> tcq arm
```

Chapter 10: Emulator Commands

tcq

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 uses trace memory for stored states; 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)

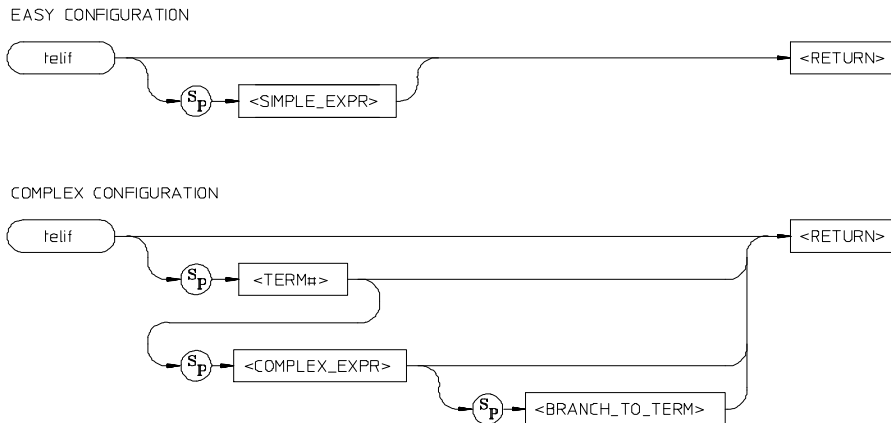
tp (specifies position of the trigger within the trace; note that **tcq** affects the number of states the analyzer can store and therefore may affect trigger positioning)

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

Chapter 10: Emulator Commands

telif

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 address 100 occurs, and then 200 occurs before 300 occurs, do the following:

```
U> tsq -i 2
U> tif 1 addr=100
U> tif 2 addr=200
U> telif addr=300
```

In complex configuration, **telif** commands can branch to other terms. For example:

```
R> tcf -c
R> tpat p1 addr=100
R> tpat p2 addr=200
R> tpat p3 addr=300
R> tsq -t 3
R> tif 1 p1 2
R> tif 2 p2 3
R> telif 2 p3 1
```

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)

Chapter 10: Emulator Commands

telif

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)

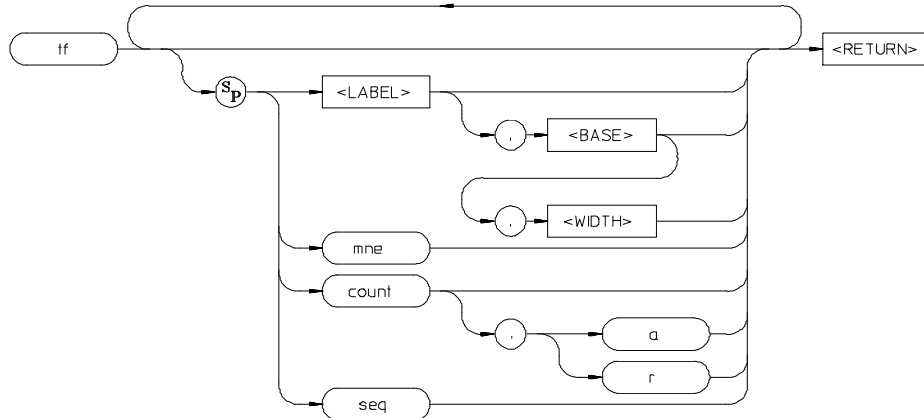
tpat (used to assign pattern names to simple expressions for use in specifying complex expressions. These complex expressions are used to specify **telif** qualifiers in analyzer complex configuration)

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 trace memory of the emulation-bus analyzer 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.

Chapter 10: Emulator Commands

tf

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.

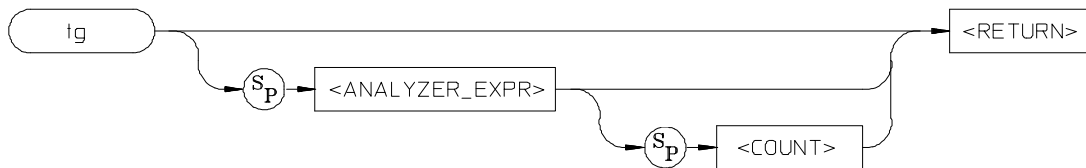
Changing the trace format *does not* change the type of information captured by the analyzer; it only specifies how the captured data should be displayed.

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 address 100 occurs on the emulation bus:

```
U> tg addr=100
```

To trigger the analyzer on any address in the range from 200 through 2ff that includes a write transaction, type:

```
U> tg addr=200..2ff and stat=write
```

To trigger the analyzer on the 32nd occurrence of address 100, type:

```
U> tg addr=100 32
```

To trigger the analyzer when address 100 occurs on the emulation bus:

```
U> tcf -c
U> tpat p1 addr=100
U> tg p1
```

To trigger the analyzer on any address in the range from 200 through 2ff that includes a write transaction, type:

```
U> trng addr=200..2ff
U> tpat p5 stat=write
U> tg r and p5
```

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.

Chapter 10: Emulator Commands

tg

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)

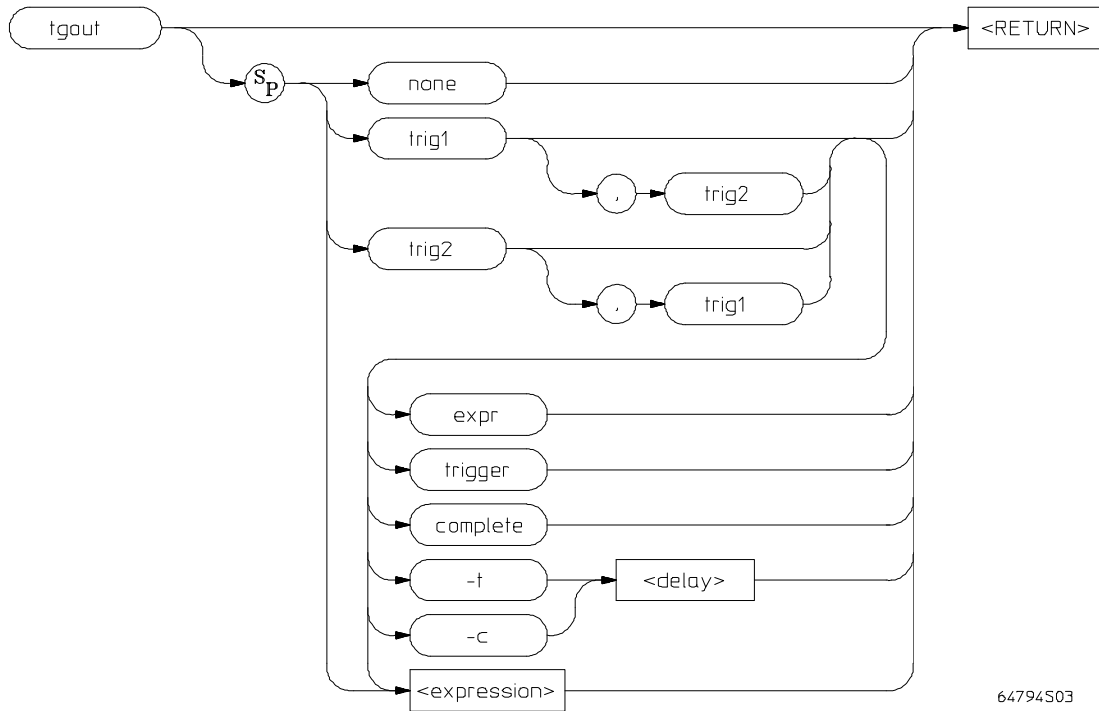
trng (used to specify a range of values for a particular group of analyzer lines; this range may be used in specifying complex analyzer expressions for triggering the analyzer)

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 syntax diagram above shows the syntax available when using the deep analyzer. If using the 1K analyzer, the **expr**, **trigger**, **complete**, **-t** and **-c** options are not available; by default, your trig1/trig2 choice selects the trigger function.

The **tgout** command allows you to specify either trig1, trig2, or both trigger signals to be driven when the emulation-bus analyzer finds the condition you specify. Trig1 and trig2 are bidirectional signal lines that can be used to coordinate measurement activity between emulators and analyzers installed in the instrumentation card cage, and instruments connected to the BNC or the CMB on the rear panel of the card cage. For details of how to configure and use trig1 and trig2, refer to the chapter on making coordinated measurements in your emulator/analyzer manual(s).

Note that there is delay in measurements that use **tgout** for measurement coordination. For example, you may specify that the emulator break to its monitor program when it receives trig1 from the analyzer. Several states may be executed

Chapter 10: Emulator Commands

tgout

in the emulator between the time the analyzer recognizes its trigger condition, generates trig1, delivers trig1 to the emulator, and the emulator responds to trig1 by breaking to its monitor program.

The parameters are as follows:

- none** If **none** is specified, neither trig1 nor trig2 will be driven by the analyzer.
- trig1** If **trig1** is specified, the trig1 signal will be driven by the analyzer when the condition you specify is found. If you do not specify a condition in your command, recognition of the analyzer trigger is assumed to be the specified condition.
- trig2** If **trig2** is specified, the trig2 signal will be driven by the analyzer when the condition you specify is found. If you do not specify a condition in your command, recognition of the analyzer trigger is assumed to be the specified condition.
- To specify that both trig1 and trig2 should be driven, concatenate both options with a comma: **tgout trig1,trig2**.
- trigger** Drive the selected trig1/trig2 signal(s) when the analyzer satisfies its trigger specification.
- complete** Available only in the deep analyzer: drive the selected trig1/trig2 signal(s) when the analyzer completes its measurement (captures trigger plus fills trace memory).
- t <delay>** Available only in the deep analyzer: drive the selected trig1/trig2 signal(s) when the analyzer satisfies its trigger specification and captures the additional number of states specified in <delay>. If you are using this feature to capture a continuous stream of target program activity, you may find some stacking cycles at the end of each trace memory if your emulation processor does stacking before a break.
- Note that if you use **-t <delay>** or **-c <delay>**, your trace will be completed automatically when the analyzer has captured enough states to satisfy the delay specification.
- c <delay>** Available only in the deep analyzer: drive the selected trig1/trig2 signal(s) when the analyzer captures the state that is after the trigger and is <delay> states before the end of trace memory. If you are using this feature to capture a continuous stream of target program activity, you may find some stacking cycles at the end of each trace memory if your emulation processor does stacking before a break.
- Note that if you use **-t <delay>** or **-c <delay>**, your trace will be completed automatically when the analyzer has captured enough states to satisfy the delay specification.

<expression>	Drive the selected trig1/trig2 signal(s) when the analyzer recognizes the state(s) that satisfies <expression>. The <expression> is a simple expression in easy configuration, and a complex expression in complex configuration. If you have already specified a tgout <trigger(s)> <expression> , you can change the expression without having to reenter the <trigger(s)>; simply type tgout <new_expression> .
expr	Available only in the deep analyzer: drive the selected trig1/trig2 signal(s) when the analyzer captures the state that satisfies the most recently defined <expression>. This is useful if you have already defined a complex tgout expression, and now you want to use that same expression to drive a different trigger. If no parameters are specified, the current state of tgout is displayed. Upon powerup or tinit , the default state is tgout none .

ExamplesDisplay the state of **tgout**:M> **tgout**

Set the emulator so that it will break from target program execution to monitor execution upon receipt of the analyzer trigger:

```
M> tcf -e
M> bc -e trig1
M> tgout trig1
M> tg addr=710
```

The emulator will break to its monitor program after the analyzer encounters address 710, asserts trig 1, and trig 1 is recognized by the emulator. This form of emulation break includes delay in the break response time. Therefore, it is not possible to predict which state will be executing when the emulator responds to the trig1 break signal and enters the monitor.

To generate trig1 when the analyzer detects a write to address 1000 when in easy configuration:

M> **tgout trig1 addr=1000 and stat=write**

To generate trig2 when the deep analyzer completes a trace:

M> **tgout trig2 complete**

Chapter 10: Emulator Commands

tgout

To generate trig1 and trig2 when the deep analyzer stores the tenth state after its trigger:

```
M> tgout trig1, trig2 -t 10
```

To generate trig2 when the deep analyzer captures the fifth state before the end of its trace memory:

```
M> tgout trig2 -c 5
```

To generate trig1 on any access to any address from 2000 through 2010 using easy configuration:

```
M> tgout trig1 expr  
M> tgout addr=2000..2010
```

or

```
M> tgout trig1 addr=2000..2010
```

To generate trig1 on any write to any address in the range from 2000..2010 while in complex configuration:

```
M> tcf -c  
M> trng addr=2000..2010  
M> tpat p5 stat=write  
M> tgout r and p5
```

While in complex configuration, to generate trig1 if pattern p1 is found while the sequencer is at level 1, or if pattern p2 is found while the sequencer is at level 2:

```
M>tgout trig1  
M>tgout 1 p1  
M>tgout 2 p2
```

To define an expression, and then assign it to generate trig2 in the deep analyzer:

```
M>tgout addr=100 and data=44 and stat=write
M>tgout trig2 expr
```

The most recent expression defined for the **tgout** command is remembered by the analyzer. Once defined, the expression can be assigned to drive either trig1, trig2, or both in a later command.

To define an expression for trig2 and then reassign it to trig1 in a later deep analyzer command:

```
M>tgout trig2 addr=100 and data=44 and stat=write
M>tgout trig1 expr
```

Note: To stop the analyzer from driving the trig1/trig2 line, issue the **th** (trace halt) command.

See Also

bc (allows you to specify a break to emulation monitor when the **tgout** condition is satisfied)

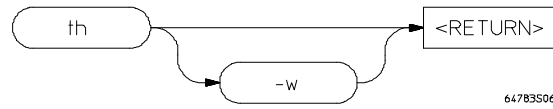
bncf (specifies whether or not trig1 and trig2 are used to drive and/or receive the rear panel BNC connector signal line)

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, for synchronizing measurements that include other analyzers)

th (halts the analyzer trace and turns off any active drive of trig1/trig2)

w -m (wait_measurement_complete. The point where measurement complete is recognized is affected by any specification that includes the **-t** or **-c** options of the **tgout** command)

th

The **th** command stops an emulation trace. This command has one parameter.

w

This suppresses display of trace output and error messages.

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 **bncd -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)

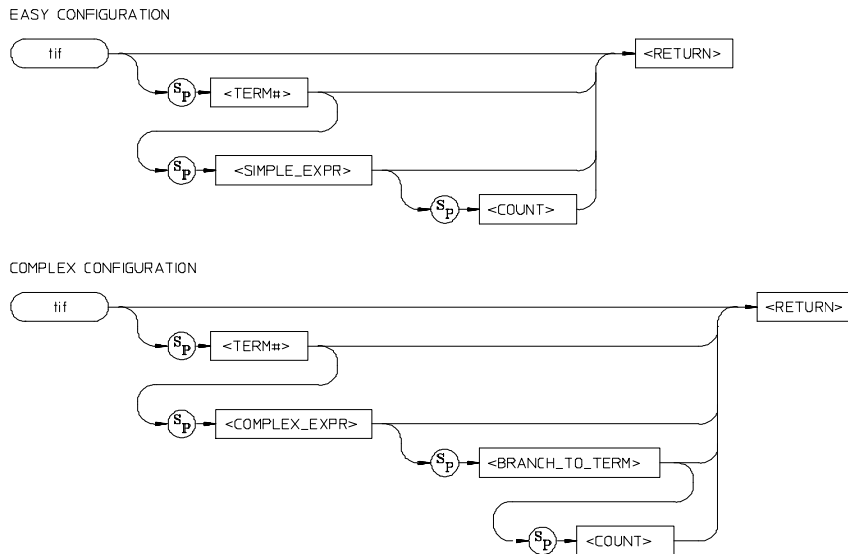
ts (allows you to determine the current status of the emulation-bus analyzer)

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#> 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> 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
```

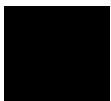
The | symbol represents an intra-set OR operator. For more information on complex expressions, operators, and pattern sets, see Chapter 11, “Expressions.”

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)**).

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.



Chapter 10: Emulator Commands

tif

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 the analyzer on the occurrence of address 300, but only after the occurrence of address 100, followed by address 200.

```
M> tif 1 addr=100
M> tif 2 addr=200
M> tif 3 addr=300
M> t
M> m 100 200 300
M> ts
```

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, tcf -lk
```

Note that the 68040 Graphical User Interface changes tcf -lk to tcf -deep, and if the trace configuration was complex, it is reset to easy.

Analyzer master clocks

```
tck -r L -u -s S
```

Trace count qualifier

```
tcq none
```

Trace format

```
tf addr,H mne
```

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

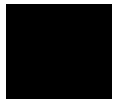
```
tsck -o 1
tsck -o 2
tsck -o 3
tsck -o 4
tsck -o 5
```

Trace Upon Execute

```
tx -d # ignore the execute signal
```

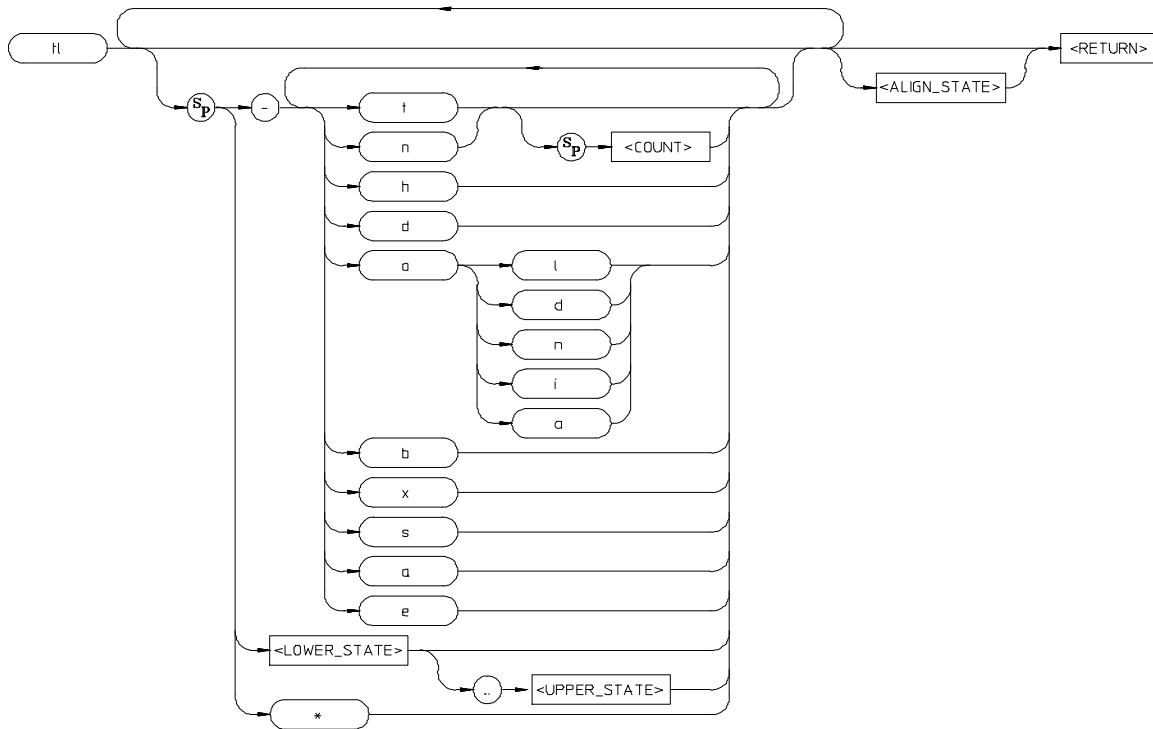
See Also

tinit (used to initialize selected portions of the emulator or the entire emulator, dependent on the options given)



tl

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 **tl** 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.

Chapter 10: Emulator Commands

tl

- 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)

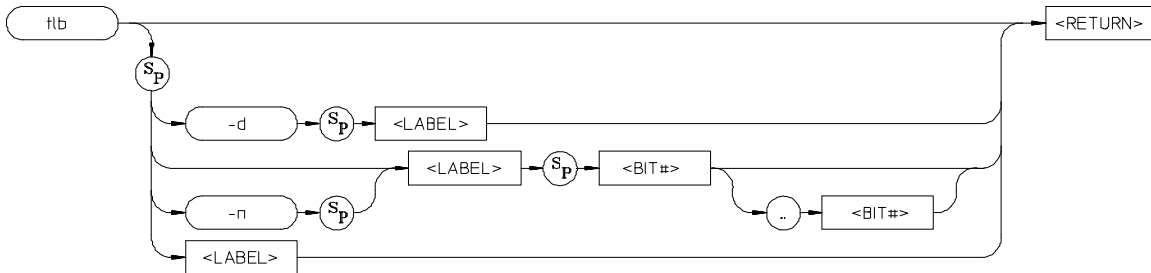
th (halts a trace in process)

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 **tfinit**, 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 byte3,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.

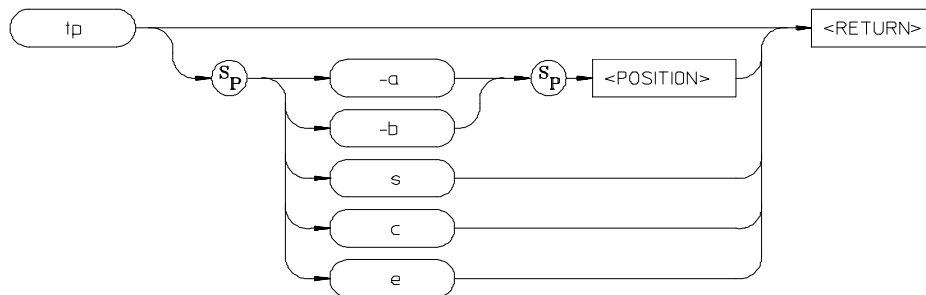
The number of labels that can be defined is limited only by system memory.

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 on an interrupt-acknowledge bus cycle, type:

M> **tg stat=ack**

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)

Chapter 10: Emulator Commands

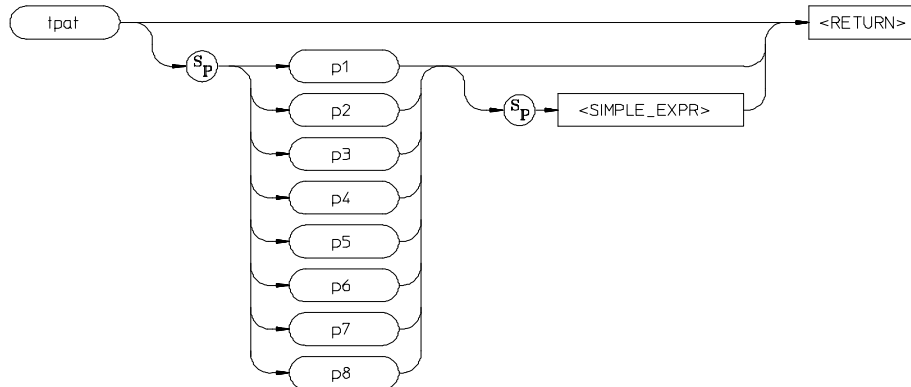
tp

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

COMPLEX CONFIGURATION ONLY



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.

Chapter 10: Emulator Commands

tpat

Upon entering complex configuration after powerup or a **tinit** initialization, all eight patterns are defined as **tpat <pattern#> any**.

Examples

Set pattern assignments on addresses, status conditions, and data values:

```
M> tpat p1 addr=100
M> tpat p2 addr=200
M> tpat p3 addr=300
M> tpat p5 stat=write
M> tpat p6 data=20
```

To set up a trigger so that any one of the above addresses will trigger the analyzer, type:

```
M> tg p1|p2|p3
```

To trigger the analyzer when address 100 is found to have a write cycle, type:

```
M> tg p1 and p5
```

To trigger the analyzer when address 200 is found to have an associated data value of 20, type:

```
M> tg p2 and p6
```

To ensure that a symbol is recognized on a long-word-aligned boundary:

```
R> tpat p1 addr=~3&Main
```

Notice the symbols "**~3&**" in the above command. These AND the value 111111111111100 (the inverse of 3) with the address of **Main** to ensure that the last two address bits are zeros. This long-aligns the address of **Main**.

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)

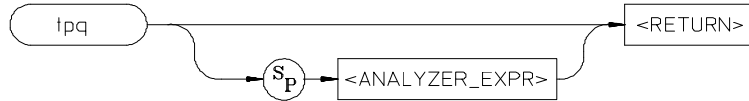
trng (defines a range of values on a set of analyzer input lines; this range may be used in conjunction with the patterns defined by **tpat** in setting up complex analysis qualifiers)

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/all** (prestore all states), **none/never** (disable prestore), or **arm** (external qualifier received from the coordinated measurement bus).

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

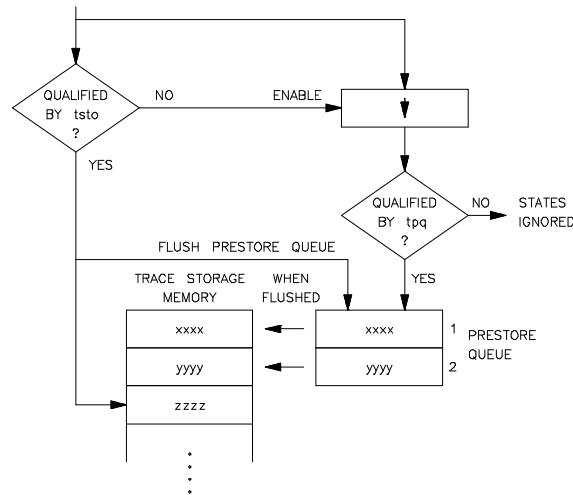
Display the current prestore qualifier:

```
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.

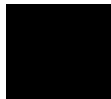
Because the prestore memory shares trace memory with store events, the number of store events recorded will be reduced by the number of prestore states recorded.

See Also

tcf (specifies whether the analyzer is to operate in easy configuration or complex configuration)

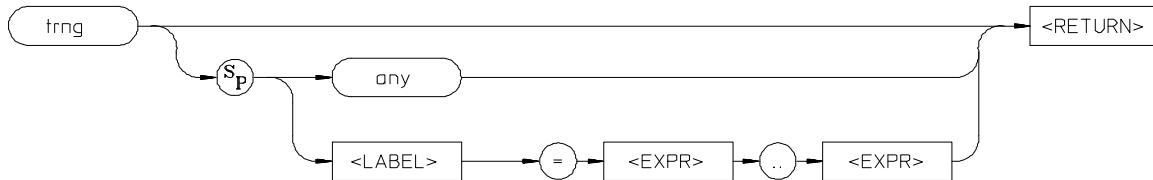
tsq (used to manipulate the trace sequencer)

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 the first access to any address in the range from 200 through 2ff:

```
M> tcf -c
M> trng addr=200..2ff
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)

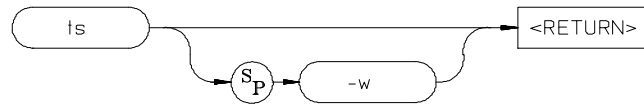
Chapter 10: Emulator Commands

trng

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:

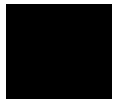
```
U> ts -w
```

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
  
```



ts

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:

Chapter 10: Emulator Commands

ts

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)

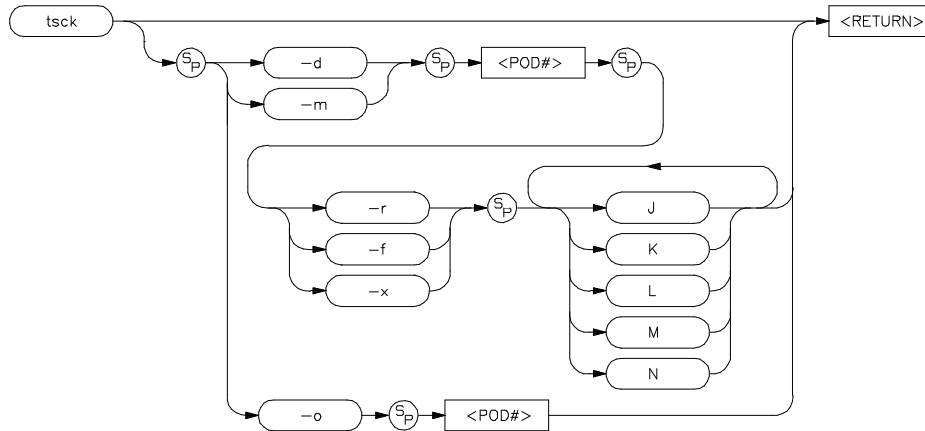
th (halt the current trace in process)

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



In the 1K analyzer, the **tsck** command allows you to specify the slave clock edges used for the emulation-bus analyzer trace. Slave clocks are not supported in the deep analyzer.

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

Chapter 10: Emulator Commands

tsck

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
```

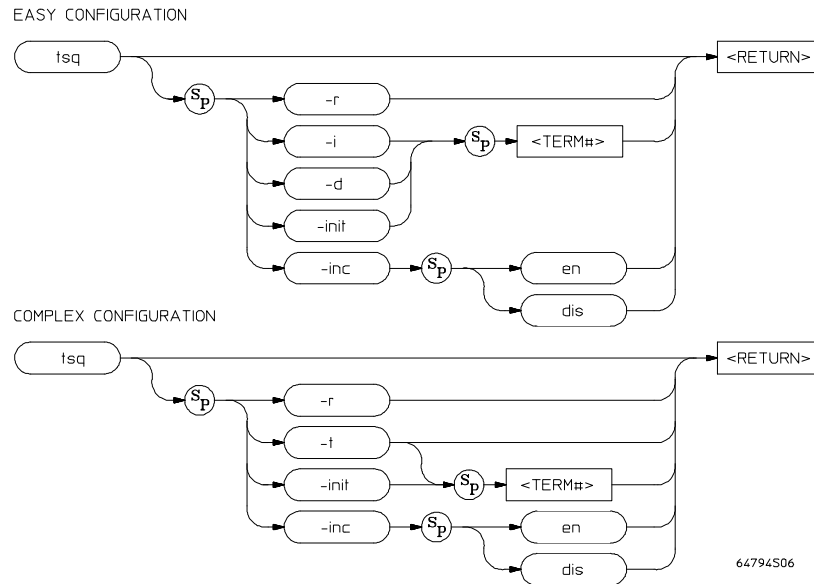
Each analyzer pod has the capability of latching certain signals with a slave clock instead of the master clock. (You set up the master clock with the **tck** command.) You should generally not use this command. It is provided for use by HP 64700 high-level interfaces.

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



The `tsq` command allows you to manipulate or display the trace sequencer. Note that the `tif` and `telif` commands are used to define each sequencer state specification.

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.

Chapter 10: Emulator Commands

tsq

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.

inc Specifying **inc** allows you to choose whether or not the states that cause the sequencer to transition from one state to another (or to the same state, in the case of restart) are qualified for storage in trace memory. By default, all states that satisfy sequencer advance specifications are stored in the trace memory (**tsq -inc en**). There may be times when an elaborate series of sequencer-advance steps is required to obtain a single traced state. You can prevent your trace memory from being filled with sequencer-advance states by using this command.

init Using **init** allows you to specify which sequence term will be the first active sequence term when a new trace begins. By default, term 1 is the first active sequence term when a new trace begins.

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
```

To allow all states that satisfy sequencer-advance specifications to be stored in the trace memory of the deep analyzer (the default), enter the command:

```
R> tsq -inc en
```

To disable storage of sequencer states in trace memory of the deep analyzer, enter the command:

```
R> tsq -inc dis
```

Chapter 10: Emulator Commands

tsq

To specify that sequence term 5 will be the active term when the deep analyzer trace first begins, enter the command:

```
R> tsq -init 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)

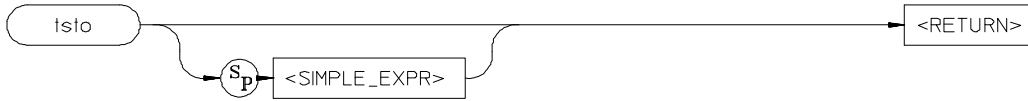
tg (defines the trigger qualifier)

tif (sets the primary branch qualifier in both easy and complex configuration)

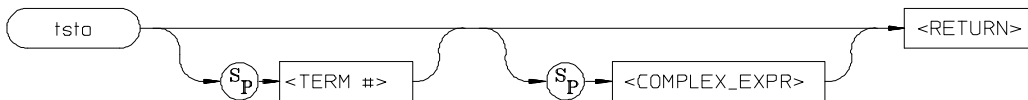
tsto (defines the analyzer global storage qualifier)

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

Chapter 10: Emulator Commands

tsto

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)

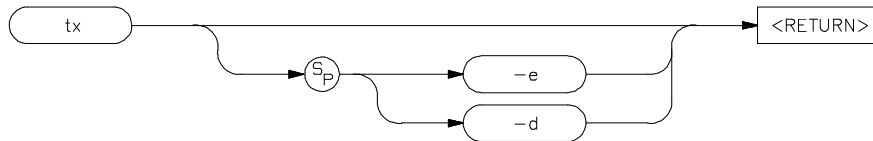
tif (used to specify a primary branch qualifier in either analyzer configuration)

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 the entry address of the demo program whenever the CMB /EXECUTE pulse is received, type the following commands:

```
M> cmbt -d none
M> tx -e
M> tg addr=entry
M> rx entry
```

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

cmbr (specifies whether the CMB trigger signal is driven or received by the internal trig1 and trig2 signals)

tar (specifies the arm condition for the analyzer)

tg (specifies a trigger condition for the analyzer)



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**

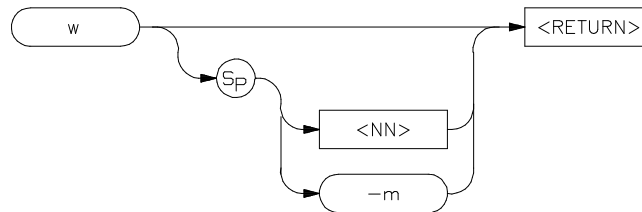
The system returns a display similar to the following:

```
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.04.00 22Oct92
```

```
HP64783 Motorola 68040 emulator
Version:  A.01.00 20Feb93
Control:  HP64748C ABG Control Board
Memory:   260 KBytes
Bank 0:   HP64172A (20ns) 256 Kbyte Memory Module
```

```
HP64740 Emulation Analyzer
Version:  A.02.02 13Mar91
```

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).

Chapter 10: Emulator Commands

w

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

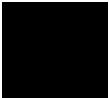
Numeric and logical expressions used in the Terminal Interface

Chapter 11: Expressions

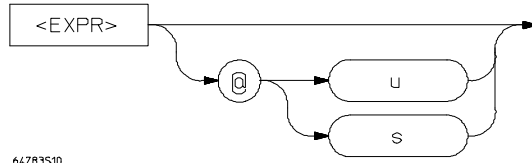
This chapter includes information about these expression types:

- 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



64783510

The address expression (EXPR) allows you to enter an address in a form recognized by the MC68040 emulator. When you see the address variable in various syntax diagrams, remember that it is unique to the MC68040 emulator.

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 either supervisor or user to further qualify an address. The @ symbol is required if you specify either supervisor or user address space. Otherwise, the @ must be omitted.

Examples

Suppose you create the following memory map:

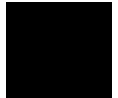
```
R>map 0..0fff eram
```

Now, the following memory display commands are valid:

```
R>m 0..0f
R>m 1000..100f@s
R>m 1000..100f@u
```

You can specify the base with the address. For example:

```
100t (100 base ten)
701o (701 base eight)
234o@s (234 base eight in supervisor space)
```

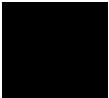


See Also

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:

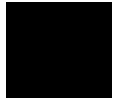
```
tpat p1 addr=2000
```

```
tpat p2 addr!=3000
```

```
tpat p5 data!=5
```

```
trng data=20..30
```

Then we create the actual complex expressions within the analyzer commands:



Chapter 11: Expressions

ANALYZER_EXPR

tg p1

tif 1 r

(**r** specifies the range defined with the **trng** command)

telif 1 p2 or p5 3

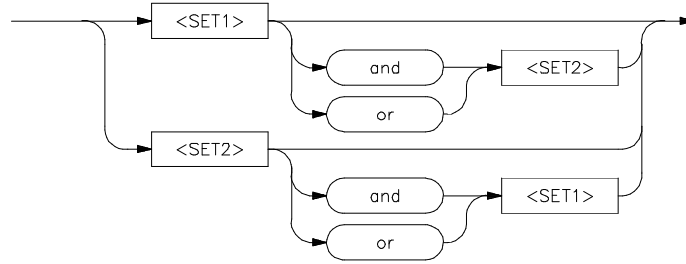
Any syntax diagram in this manual that indicates <ANALYZER_EXPR> means that a simple expression is required when the analyzer is in easy configuration, and a complex expression is required when the analyzer is in complex configuration.

See Also

See the <SIMPLE_EXPR> and <COMPLEX_EXPR> syntax pages for complete details on each expression.

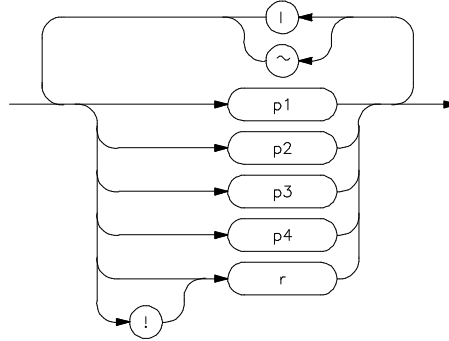
COMPLEX_EXPR

<COMPLEX_EXPR>



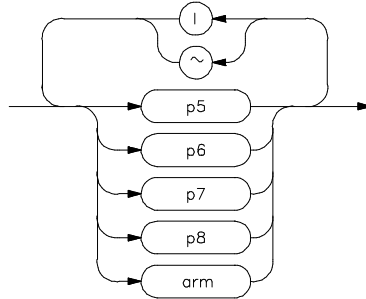
<SET1>

(restricted to one operator type in the set)



<SET2>

(restricted to one operator type in the set)



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.)

Interaset Operations

You use interaset operators to form relational expressions between members of set 1 and set 2. The operators are:

and (interaset logical AND)

or (interaset logical OR)

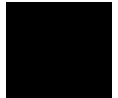
You can then form the following types of expressions:

(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

$$A \text{ NOR } B = (\text{NOT } A) \text{ AND } (\text{NOT } B)$$

and

$$A \text{ NAND } B = (\text{NOT } A) \text{ OR } (\text{NOT } B)$$

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

$$(\text{addr}=2000) \text{ NAND } (\text{data}=23)$$

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:

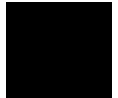
p1~p2

This is effectively the same as:

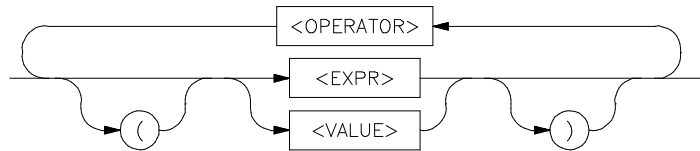
(NOT addr=2000) NOR (NOT data=23) = (addr=2000) AND (data=23)

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

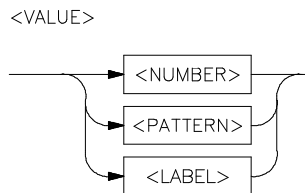


Numeric expressions are the root of all HP 64700 Terminal Interface expression types, including analyzer expressions, address specifications, equates, and expressions you might want to calculate using the **echo** command.

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

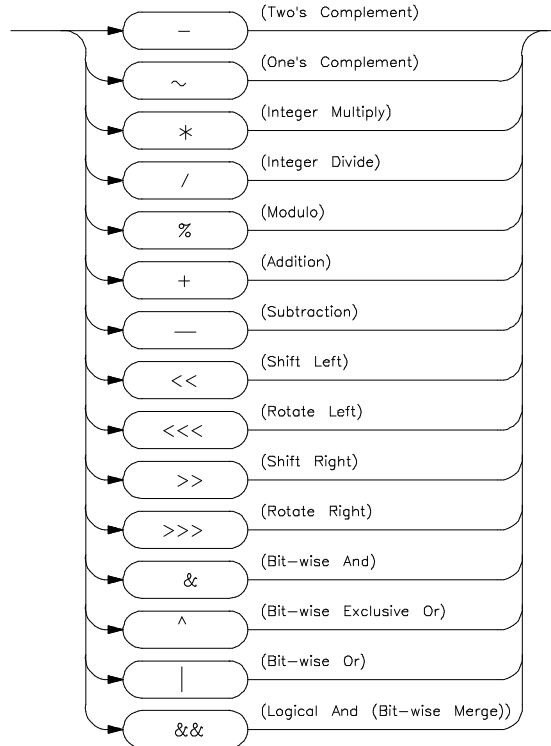
(This actually gives a range from 10 to 17 hex.)

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:

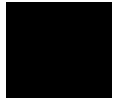
03dh+03fh = 07ch
1110Y-101Y = 1001Y

<< <<<
>>>>

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



$$1x01Y>>>1 = 10000000000000000000000000000001x0Y$$

$$0xf0abcdH>>>4 = 0dxf0abcH$$

&

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:

$$10xy&11x1Y = 10xxY$$

^

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

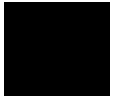
An overlap, indicated by a * in the merge truth table, may occur if two patterns specify different values for a pattern bit. If an overlap occurs, the first pattern's value for that bit overrides the second pattern's value.

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.



Chapter 11: Expressions

EXPR

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

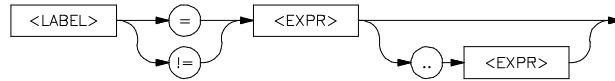
See the <ANALYZER_EXPR>, <SIMPLE_EXPR>, and <COMPLEX_EXPR> pages for information on the use of expressions in forming analyzer expressions.

See the **echo** and **equ** command syntax pages for information on use of expressions in expression calculation and equates.

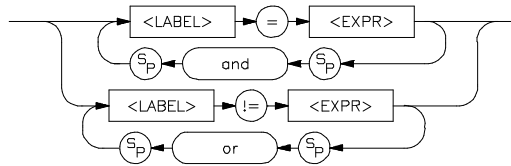
See the <ADDRESS> syntax pages for information on use of expressions in addressing.

SIMPLE_EXPR

EASY CONFIGURATION ONLY



EASY AND COMPLEX CONFIGURATION



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

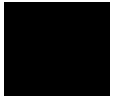
Examples addr=2000h
 data=25h+20h
 stat=0110xxxxY

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
```

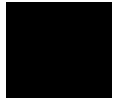
You would represent the above (incorrect) simple expression as a complex expression of the form:

```
p1 | p2
```

You could also do this:

```
tpat p1 label!=expression
```

```
tpat p5 label=expression
```



Chapter 11: Expressions

SIMPLE_EXPR

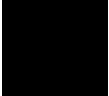
Represent this in complex form as:

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

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

!ERROR 1! I/O port access not supported

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: execution of a breakpoint instruction).

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 MC68040 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. Refer to error number 147 in this chapter for a detailed explanation that may apply directly to the problem causing this error message.

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). The emulator will not allow temporary breaks while the emulator is in the reset state or while the target program is running.

Action: Break to the monitor using the **b** command, and then execute the desired command or disable 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 and the emulator expects the stack to be aligned on a word boundary.

Action: Modify the stack pointer to an even value.

140 **!ERROR 140! Invalid attribute for memory type : <attribute>**

Cause: The memory type attributes **dp** and **lock** 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**).

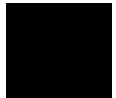
Action: Refer to the chapter titled, "Using the Terminal Interface" for information on the memory type attributes.

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.



!ERROR 142! Dual ported memory already in use

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 and you tried to map another term using the **dp** attribute. If you select the foreground monitor (**cf mon=fg**), this block is used by the monitor and is not 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! Monitor address is not set to <addr> for downloaded monitor
!ERROR 144! Continuing with default foreground monitor**

Cause: You have downloaded a custom foreground monitor which was linked at an address other than the monitor address specified within the emulation configuration.

Action: Change the monitor address within the emulation configuration or link your custom monitor at the address specified in the configuration.

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.



!ERROR 147! Request cannot be satisfied with remaining map resources

147

!ERROR 147! Request cannot be satisfied with remaining map resources

Cause: Although you have not exceeded the maximum number of map terms that can be specified in the memory map, you have run into a hardware resource limitation in the emulator that arises when target memory is mapped using the **tc**i attribute.

There are eight hardware resources on the emulation probe for mapping emulation memory and driving the TCI signal for target memory ranges. When two emulation memory modules are installed, the emulator requires seven of these resources to map all of the emulation memory. Target memory ranges require either zero or one resource, depending on whether or not use of the **tc**i attribute matches its use in the "other" term. For example, if "other" is mapped to target RAM without the **tc**i attribute, one hardware resource is required to add a map term for target memory that uses the **tc**i attribute. Consuming additional hardware resources for mapping target memory will reduce the amount of emulation memory available for mapping. Once all eight hardware resources have been consumed, mappable emulation memory will be reduced to zero and you will get this message.

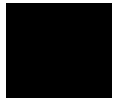
Action: Try to minimize the number of hardware resources used for mapping target memory by mapping the "other" term to target memory both with and without the **tc**i attribute. Find out whether you use less hardware resources by specifying the **tc**i attribute or not specifying the **tc**i attribute for the "other" term.

150

!ERROR 150! Program counter is odd or uninitialized

Cause: You tried to run the processor from the current PC, but the value of the current PC is odd.

Action: Modify the PC to an even value. The processor expects even word alignment of opcodes.



!ERROR 150! Program counter is located in guarded memory

150

!ERROR 150! Program counter is located in guarded memory

Cause: You tried to run but the emulator detected that the program counter is located in guarded memory. This error will only be generated if the MMU is disabled; otherwise, you will see an asynchronous error indicating access to guarded memory occurred when the emulator attempted to run the target program.

Action: Make sure the program counter is set to an address in RAM or ROM before you attempt to run your program.

151

!ERROR 151! Interrupt stack pointer is odd or uninitialized
!ERROR 151! Master stack pointer is odd or uninitialized

Cause: You are in the monitor and you tried to run, but the emulator detected that your stack pointer is invalid (it detected an odd value).

Action: Use the **reg** command to set the stack pointer to an even value that points at a memory region (emulation or target RAM) that can be used for stack operations before running your program.

151

!ERROR 151! Interrupt stack is located in ROM: <address>
!ERROR 151! Master stack is located in ROM: <address>

Cause: You issued a command to run the target program, but when the emulator attempted to write to one of your stacks, it detected that the stack address is in memory mapped as ROM, and you enabled breaks on writes to ROM.

The monitor exits your target program by executing an RTE instruction. Depending upon whether or not you set the M bit in the SR, the monitor will either place a format \$0 stack frame on the interrupt stack or will place a format \$1 (throwaway) stack frame on the interrupt stack and a format \$0 stack frame on the master stack. Any access violations detected during these writes will abort the exit from the monitor.

Action: Use the **reg** command to set the stack pointer to an even value that points at a memory region (emulation or target RAM) that can be used for stack operations before running your program. Or, you can modify the emulation configuration and respecify the memory map to RAM for the address range containing the interrupt stack and/or the master stack.

!ERROR 151! Interrupt stack is located in guarded memory: <address>

151

!ERROR 151! Interrupt stack is located in guarded memory: <address>**!ERROR 151! Master stack is located in guarded memory: <address>**

Cause: You issued a command to run the target program, but when the emulator attempted to write to one of your stacks, it detected that the stack address is in memory mapped as guarded.

The monitor exits to user program by executing an RTE instruction. Depending upon whether or not you set the M bit in the SR, the monitor will either place a format \$0 stack frame on the interrupt stack or will place a format \$1 (throwaway) stack frame on the interrupt stack and a format \$0 stack frame on the master stack. Any access violations detected during these writes will abort the exit from the monitor.

Action: Use the **reg** command to set the stack pointer to an even value that points at a memory region (emulation or target RAM) that can be used for stack operations before running your program. Or, you can modify the emulation configuration and respecify the memory map to RAM for the address range containing the interrupt stack and/or the master stack.

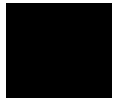
151

!ERROR 151! Interrupt stack is not located in RAM: <address>**!ERROR 151! Master stack is not located in RAM: <address>**

Cause: You issued a command to run the target program. When the emulator attempted to write to one of your stacks, it detected that the stack address is not located in memory which operates as RAM. When the monitor writes out a stack frame to your stack space, the monitor reads it back to verify that it was created correctly. Unless the emulator can verify that the stack frame is located in RAM and was created correctly, the monitor will abort the run.

The monitor exits the target program by executing an RTE instruction. Depending upon whether or not you set the M bit in the SR, the monitor will either place a format \$0 stack frame on the interrupt stack or will place a format \$1 (throwaway) stack frame on the interrupt stack and a format \$0 stack frame on the master stack. Any access violations detected during these writes will abort the exit from the monitor.

Action: Use the **reg** command to set the stack pointer to an even value that points at a memory region (emulation or target RAM) that can be used for stack operations before running your program. Or, you can modify the emulation configuration and respecify the memory map to RAM for the address range containing the interrupt stack and/or the master stack.



!ERROR 154! Hardware breakpoints can only be used in target memory

154 **!ERROR 154! Hardware breakpoints can only be used in target memory**

Cause: You attempted to use the "force hardware" option to set a breakpoint at an address mapped as emulation memory. The "force hardware" option for breakpoints is not available for addresses in emulation memory; it is only available for breakpoints in target memory, typically for setting breakpoints in target ROM.

Action: Delete the "force hardware" option from your command and try to set the breakpoint again.

154 **!ERROR 154! Out of hardware breakpoints**

Cause: You either tried to set a breakpoint in target ROM or use the force hardware option to set a breakpoint in target RAM, and all eight hardware breakpoint resources are already in use.

Action: Review your present set of breakpoints to see if you can delete one or more of the hardware breakpoints that are presently set. No more than eight hardware breakpoints can be set at any one time (one per aligned long word). Only one hardware resource is used if two hardware breakpoints are set in the same long word.

155 **!STATUS 155! Vector table modified for single stepping**

Cause: This status message indicates that you issued the emulator command to single step. The emulator detected that the trace vector was not properly set for stepping so the emulator temporarily modified one or more exception vectors in your vector table. The original values are restored by the emulator after the step completes. This message is only issued one time if you do not change the address or value of the trace vector.



!ERROR 156! Unable to modify trace vector to <value> for single stepping

156

!ERROR 156! Unable to modify trace vector to <value> for single stepping

Cause: You tried to single step, and the emulator detected the trace vector was not set properly and the emulator was unable to modify the vector table because it was not located in emulation memory or target RAM. This usually occurs when the vector table is located in target ROM.

Action: Copy or relocate the vector table in emulation memory or target RAM, or change your ROM image so that it contains the proper value for the trace vector for single stepping. Refer to stepping information in the chapter titled "Using the Emulator" in this manual.

157

!STATUS 157! Disabled mmu/cache while background monitor is selected

Cause: This status message indicates that the MMU and/or cache was enabled in the emulation configuration when you changed the monitor type to background. The background monitor requires the MMU and the cache to be disabled in order to operate properly. Both the MMU and the cache were disabled automatically when you changed to the background monitor.

158

!ERROR 158! Cannot enable mmu/cache while background monitor is selected

Cause: You tried to enable either the MMU or the cache within the emulation configuration after selecting the background monitor. The background monitor requires the MMU and the cache to be disabled in order to operate properly.

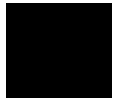
Action: Use the foreground monitor if you want to enable either the MMU or the cache.

160

!ERROR 160! MMU is not enabled via configuration

Cause: You tried to display MMU translations or load the deMMUer but the MMU is disabled within the emulation configuration.

Action: If you wish to use the MMU, enable it in the emulation configuration before attempting to display its translations or load the deMMUer.



!ERROR 160! MMU is not enabled via translation control register

160 **!ERROR 160! MMU is not enabled via translation control register**

Cause: You tried to display the MMU translations or load the deMMUer. While the MMU is enabled within the emulation configuration, the enable bit is not set in the translation control register.

Action: Either enable the MMU in your target system by modifying the TC register, or specify an enabled value for the TC register on the command line when invoking the MMU or deMMUer commands.

161 **!ERROR 161! No translation for alternate function code address spaces**

Cause: You tried to display an MMU translation for an address specified with alternate function codes 0, 3, 4, or 7.

Action: Don't use alternate function codes 0, 3, 4, or 7 when attempting to display an MMU translation for an address. The MMU does not translate addresses in alternate function code space.

162 **!STATUS 162! Display truncated to <number of lines> lines**

Cause: This status message indicates that more lines of MMU translations could have been displayed, but when you requested a display of MMU translations, you limited the number of lines to be displayed.

163 **!ERROR 163! DeMMUer has not been loaded**

Cause: You tried to enable the deMMUer before it had been loaded. The deMMUer can only be enabled after it has been loaded with a set of reverse translation information.

Action: Load the deMMUer from the present translation tables in memory or from a deMMUer file that you have previously saved.



!ERROR 163! Unable to access deMMUer while analysis trace is in process

163

!ERROR 163! Unable to access deMMUer while analysis trace is in process

Cause: You tried to issue a command that requires access to the deMMUer while the analyzer was running a trace. You cannot load, enable or disable the deMMUer while an analysis trace is in process.

Action: Wait for the trace to complete or stop the trace before changing the state of the deMMUer.

170

!STATUS 170! Emulator terminated hung bus cycle: <address> byte read

Cause: This status message will be displayed if the target system fails to provide \overline{TA} or \overline{TEA} bus cycle termination for a particular cycle and the emulator terminates the bus cycle in order to break from execution of the target program to execution within the monitor, or to complete execution of a monitor command (which accessed this memory address). This can happen on any access to target memory or interlocked emulation memory (when using the "lock" attribute in terminal mode).

The emulator will not terminate any hung bus cycles unless you explicitly say break or you execute a monitor command (ie: "m 7000"). The emulator will generate this status message each time it terminates a hung bus cycle. The emulator never attempts to terminate bus cycles in program space (opcode fetches) or for any addresses in the foreground monitor.

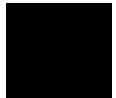
170

!ERROR 170! Target failed to terminate bus cycle: <address> long read

Cause: You attempted to break or reset into the monitor and the target system failed to terminate a bus cycle with \overline{TA} or \overline{TEA} . Normally, the emulator will force bus cycle termination for the target system in order to break into the monitor.

However, the emulator refused to terminate the bus cycle because the address was in program space or it was within the address range of the foreground monitor.

Action: Reset the emulator and target system. If the address is within emulation memory, do not use the lock attribute if the target system does not provide cycle terminations within this address range.



Chapter 12: Emulator Error Messages
!ERROR 171! Request failed; bus grant

- 171 **!ERROR 171! Request failed; bus grant**
- Cause: An attempt was made to execute a monitor command, but an external target system device has monopolized the bus and the monitor is no longer responding.
- Action: Wait until the processor has regained bus control, and then retry the operation or don't let external devices monopolize the bus for extended periods of time.
- 171 **!ERROR 171! Request failed; 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.
- 171 **!ERROR 171! Request failed; slow clock**
- Cause: The target system is providing target power but no clock signal.
- Action: Make sure the clock oscillator is installed correctly.
- 171 **!ERROR 171! Request failed; target reset**
- Cause: During a monitor command, the target system asserted (and continues to assert) the reset signal; the monitor is no longer responding.
- Action: Prevent your target system from asserting the reset signal when you are using monitor commands.
- 171 **!ERROR 171! Request failed; halted**
- 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.

171 **!ERROR 171! Request failed; no bus cycles**

Cause: During a monitor command, some problem caused the processor to stop running bus cycles.

Action: Use the emulation-bus analyzer to determine what caused the problem and try to work around them. If you are using the demo board, make sure the reset flying lead from the probe is connected to the demo board.

171 **!ERROR 171! Request failed; unexpected exception: <vector number>**

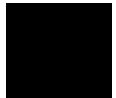
Cause: The monitor was executing a command and some exception occurred that it did not expect. During monitor command execution, the monitor traps all exceptions by using its own stack and vector table. The monitor provides exception handlers for some exceptions, such as access fault, so that it can either recover or issue a detailed error message. The monitor had no exception handler for the exception number shown in this message.

Action: Reset the emulator and try your command again.

172 **!ERROR 172! Target bus error: <address>**

Cause: The monitor attempted to access target system memory or interlocked emulation memory and the target system terminated the bus cycle with TEA.

Action: Retry your command. If the error occurs again and if it is during an attempted access to emulation memory, you can change the interlocked attribute to emulation memory. If the error occurs again on access to target system memory, inspect your target system to understand why it is sending the TEA for the specified address.



!ERROR 172! Address translation error; target bus error: <address>

172 **!ERROR 172! Address translation error; target bus error: <address>**

Cause: The error occurred when the monitor attempted to access memory with the MMU enabled. You requested the monitor to display or modify memory, or you tried to exit the monitor; the memory access generated an access fault resulting from an MMU address translation failure. The target system terminated a tablewalk cycle with TEA (bus error).

Action: Verify that the SRP and URP registers point to the correct location in memory where your address translation tables reside. If this is target memory, you will need to determine why your target system asserts TEA.

172 **!ERROR 172! Address translation error; non-resident page: <address>**

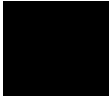
Cause: The error occurred when the monitor attempted to access memory with the MMU enabled. You requested the monitor to display or modify memory, or you tried to exit the monitor; the memory access generated an access fault resulting from an MMU address translation failure. This error indicates that the address does not have a valid translation.

Action: Display the address translation tables for the <address> given in the message. You can display the MMU translations to see if the <address> is within one of the translated ranges. You can display translation tables for the address, and then you can view table details if one of the translation tables seems to be misdirecting the translation of the address.

172 **!ERROR 172! Address translation error; supervisor-only page: <address>**

Cause: The error occurred when the monitor attempted to access memory with the MMU enabled. You requested the monitor to display or modify memory, or you tried to exit the monitor and the memory access generated an access fault resulting from an MMU address translation failure. A user mode access was attempted to a page that is only accessible in supervisor mode.

Action: Try your command again, but be sure to specify access in the supervisor mode.



!ERROR 172! Address translation error; write-protected page: <address>

172

!ERROR 172! Address translation error; write-protected page: <address>

Cause: The error occurred when the monitor attempted to access memory with the MMU enabled. You requested the monitor to display or modify memory, or you tried to exit the monitor and the memory access generated an access fault resulting from an MMU address translation failure.

This error indicates that write access was denied to a write-protected page. NOTE: Except for stacking on exit, any attempts to modify memory in write protected pages using the monitor will succeed as long as the translation tables reside in RAM. The monitor will temporarily clear any write protect flags in your translation tables in order to force the access to be completed. If the monitor is unable to clear the write protect flags because the translation tables are in ROM, you will see this error.

Action: Check the content of the write protected page to see if it has been changed by the attempted write transaction.

173

!ERROR 173! Monitor operation interrupted by target system

Cause: Your attempt to execute a monitor command was aborted when the target system preempted the monitor and did not return control. When the foreground monitor is running and is in its idle state, the monitor can be interrupted by the target system to service target system requirements. If the target system interrupts the monitor and fails to return control to the monitor after it has finished, this error is generated. The emulator does not attempt to regain control when after the monitor has been preempted.

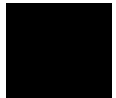
Action: The only way to regain control of your emulation system is to reset the emulation processor. If you do not want the monitor to be preemptable by target system interrupts, you can increase the monitor interrupt priority level. Refer to the chapter that discusses the emulation configuration options.

174

!ERROR 174! No monitor configured

Cause: You configured monitor "none" and you tried to break into the monitor or execute a command that requires use of the monitor.

Action: Either change the configuration to use a monitor, or do not try to issue a command that requires the monitor.



Chapter 12: Emulator Error Messages
!ERROR 175! Coverage not supported

175 **!ERROR 175! Coverage not supported**

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

175 **!ERROR 175! Copy target image not supported**

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

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

176 **!ERROR 176! Update HP64700 system firmware to A.04.00 or newer**

Cause: This error occurred because your system firmware is out of date.

Action: Refer to the chapter titled, "Installing/Updating Emulator Firmware". You must update the firmware to the version number specified in the message, or newer firmware version number. Your system is not usable with its present firmware.

177 **!ERROR 177! Update HP64740 firmware to version A.02.02 or newer**

Cause: This error occurred when you attempted to disassemble a trace and the analyzer firmware was found to be out of date.

Action: Refer to the chapter titled, "Installing/Updating Emulator Firmware". You must update the firmware to the version number specified in the message, or newer firmware version number. Your analyzer is not able to disassemble its trace memory with its present firmware.

!ERROR 178! Unable to run HP64783 performance verification tests

178

!ERROR 178! Unable to run HP64783 performance verification tests

Cause: You entered the **pv** command, but the emulator was unable to start performance verification because the firmware did not identify the probe as being the MC68040.

Action: Make sure the correct emulator probe is connected and that all cables are secured. Make sure that the demo board is connected to the emulator probe, the power cable is connected between the HP 64700 card cage and the demo board, and the reset flying lead is connected between the emulation probe and the demo board.

178

!ERROR 178! Unable to run HP64783 tests without target power

Cause: The demo board does not have proper power connected to it.

Action: Check the connections of the external power cable and the reset flying lead to the demo board.

179

!STATUS 179! HP64783 M68040 firmware not compatible with emulation probe

Cause: The emulation control card is programmed with MC68040 firmware, but the firmware does not identify the probe as being the MC68040.

Action: Make sure that you are using an MC68040 probe, and then make sure the probe cables between the control card and the probe are connected correctly. Refer to the Installation and Service Chapter for proper cable connections.



Chapter 12: Emulator Error Messages
!ERROR 201! Out of system memory

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>**). This 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 an incorrect option to a command. <option>, if printed, indicates the incorrect option.

Action: Use online help by typing **help <command>** or **? <command>**. Reenter the command with the correct syntax. Refer to the chapter titled, "Emulator Commands" 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. Refer to the chapter titled, "Emulator Commands" to determine the expression type; then see the "Expressions" chapter 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>**. Refer to the chapter titled, "Emulator Commands" in this manual for more information.

310 **!ERROR 310! Invalid address: <address>**

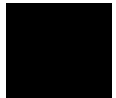
You specified an invalid address value as an argument to a command (other than an analyzer command). For example, you may have specified digits that don't correspond to the base specified, or you forgot to precede a hexadecimal letter digit with a number (even zero (0)).

Action: Reenter the command and the address specification. Use online help by typing **help proc**. See the <ADDRESS> and the <EXPRESSION> syntax pages in the "Expressions" chapter 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 the "Expressions" chapter 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).



!ERROR 313! Missing option or operand

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>**. Refer to the chapter titled, "Emulator Commands" in this manual for further information on required syntax.

314 **!ERROR 314! Option conflict: <option>**

Cause: You have entered a command with two options that 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 the chapter titled, "Emulator Commands" in this manual to determine which options can 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.



Chapter 12: Emulator Error Messages
!ERROR 400! Record checksum failure

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**

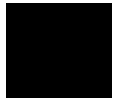
Cause: The monitor is not responding.

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.



!ASYNC_STAT 605! Undefined software breakpoint: <breakpoint address>

605 **!ASYNC_STAT 605! Undefined software breakpoint: <breakpoint address>**

Cause: This status message indicates a breakpoint instruction was executed and the emulator stopped target execution and started running in the monitor. The emulator had no record of a breakpoint being set at this address. This can happen if the MMU relocates a page containing a breakpoint before that breakpoint is executed. In this case, the emulator will have no record of the breakpoint at the relocated address.

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: 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, read "Execution Breakpoint Problems" in the chapter titled, "Using Memory Management".

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 normally used with other messages that further describe the error. It is displayed if the emulator is unable to break to the monitor because the emulation processor is reset, halted, or the monitor is not responding for some reason.

Action: First, look at the emulation prompt and other status messages displayed to determine why the processor is stopped. If reset by the emulation controller, use the **b** command to break to the monitor. If reset by the target 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.

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

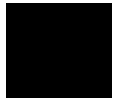
!ASYNC_STAT 615! Software breakpoint: <breakpoint address>

Cause: This status message indicates that the target program executed a software breakpoint instruction (an execution breakpoint, either in software or provided by one of the eight hardware breakpoint resources). The emulator stopped the target program and began running in the monitor.

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.



Chapter 12: Emulator Error Messages
!ASYNC_STAT 617! CMB trigger break

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 mon=junk** instead of **cf mon=fg**.

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>**

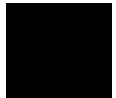
Cause: Target condition or system failure while trying to change configuration item.

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 MC68040 emulator does not 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 the chapter titled, "Emulator Commands" in this manual.



!ASYNC_STAT 628! Write to ROM break:<ROM address>

628 **!ASYNC_STAT 628! Write to ROM break:<ROM address>**

Cause: This status message indicates the target program accessed memory mapped as either emulation ROM or target ROM; the emulator interrupted target execution and began running in the monitor. This only occurs if you enabled breaks on writes to ROM. When the MMU is enabled, the address displayed in this message will be physical, as denoted by the trailing "a" after the function code.

628 **!ASYNC_STAT 628! Guarded mem break: <guarded memory address>**

Cause: This status message indicates that the target program accessed memory mapped as guarded and the emulator interrupted target execution and began running in the monitor. When the MMU is enabled, the address displayed in this message will be physical, as denoted by the trailing "a" after the function code.

628 **!ASYNC_STAT 628! 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.

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 is likely that the emulator was unable to break to the monitor to perform the register modification.

634 **!ERROR 634! Display register failed: <register>**

Cause: The emulator was unable to display the register you requested.

!ERROR 637! Register class cannot be modified: <register class>

Action: To resolve this, you must look at the other status messages displayed. It is likely that the emulator was unable to break to the monitor to perform the register display.

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 the chapter titled, "Emulator Commands" in 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**

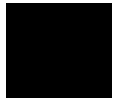
Cause: You have entered a **rst -m** command and the emulator is unable to break into the monitor.

Action: This message is accompanied by other error messages. Look at those messages for clarification of the problem and actions to take.

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.



!ERROR 653! Break condition configuration aborted

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 disabled the software breakpoint feature. Breakpoints are enabled by default. Then you attempted to set a breakpoint, or you attempted to single step with the foreground monitor (either the built-in or custom foreground monitor).

Action: Re-enable the software breakpoint feature and try again.

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>**.

665 **!ERROR 665! Enable breakpoint failed: <address>**

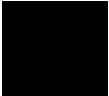
Cause: System failure or target condition.

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

666 **!ERROR 666! Disable breakpoint failed: <address>**

Cause: System failure or target condition.

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



667 **!ERROR 667! Breakpoint code already exists: <address>**

Cause: 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**.

Action: Remove the breakpoints from your program code and use **bp** to insert breakpoints.

668 **!ERROR 668! Breakpoint not added: <address>**

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

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

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.

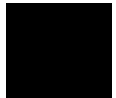
671 **!ERROR 671! Breakpoint disable aborted**

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.



!ERROR 682! Invalid step count: <count>

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.

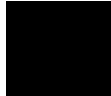
692

!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: The emulator was unable to perform the requested operation on memory mapped to the target system. This message is displayed in conjunction with other error messages that further clarify the problem that occurred. In most cases, the problem results from the emulator's inability to break to the monitor to perform the operation.
- Action: See other error messages to further understand the cause of the error.
- 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.



!ERROR 720! Invalid map term number: <map term number>

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 9**, but the emulator only has eight 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 MC68040 emulator, there are only eight terms. If you had already defined memory types for these terms, then tried to map another term, you would see the above error message.

Action: Either combine map ranges to conserve on the number of terms or delete mapper terms that aren't needed.

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 the "Expressions" chapter 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).



!ERROR 725! Unable to load new memory map; old map reloaded.

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>**

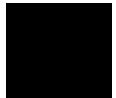
Cause: The only valid memory map attributes for the MC68040 emulator are **dp** (dual-port memory), the **lock** (interlock target system termination signals \overline{TA} and \overline{TEA}), and the **tei** (transfer cache inhibit) attributes. For example, the following command will cause an error: “map 0..100 eram loc2”.

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**.



!ERROR 734! Map range overlaps with term: <term number>

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 the chapter titled, "Expressions" in 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>**

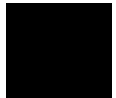
Cause: You tried to delete a macro that did not exist; or you tried to define a new macro with a name containing characters other than letters, digits, or underscores.

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.



!ERROR 814! Command line too complex

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**

Cause: 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>**

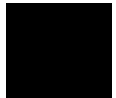
Cause: You specified the date format incorrectly in the **dt** command.

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.



!ERROR 844! Invalid repeat count: <count>

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>**

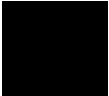
Cause: This error occurs when you enter a global or user symbol name with incorrect syntax.

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 you specify the module name, followed by a colon, and then the symbol name (for example **module:loc_sym**). Make sure 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**

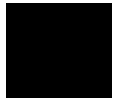
Cause: This error occurs because the system is out of memory.

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.
- 1002 **!STATUS 1002! Analyzer SIMMs are not all the same size; using smallest size**
- Cause: Plug-in SIMMs are used to expand the trace depth of the deep analyzer to 64k or 256k states. Four SIMMs, all of the same size must be used. If they are not all the same size, the smallest SIMM size in the set of four will be used for trace depth.
- Action: No action necessary.
- 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.

!ERROR 1103! Invalid bit range; out of bounds: <sig#>..<sig#>

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>**

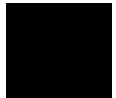
Cause: This error occurs when you attempt to delete an emulation trace label that is currently being used as a qualifier in the emulation trace specification or is currently specified in the emulation trace format.

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.



!ERROR 1130! Illegal base for count display

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.

!ERROR 1139! Illegal width for addr display, mne not specified

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).

1203 **!STATUS 1203! Trigger position changed to accomodate trig1, trig2 delay spec**

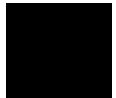
Cause: The terminal interface **tgout** (trigger output) command provides a delay feature that allows for driving of the trig1 and/or trig2 signals a specified number of states after trigger or before trace complete. The setup of this delay feature interacts with the trigger position specification. The trigger position specification may be automatically modified by the deep analyzer in order to make the delay feature work in the expected manner.

Action: You can use the terminal interface command **tp** (trigger position) to examine the new trigger position value.

1207 **!ERROR 1207! Invalid clock channel: <name>**

Cause: Valid clock channels are L, M, and N.

Action: Respecify the command using valid clock channels.



!ERROR 1209! Operator must be “and” or “or”: <expression>

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 the "Expressions" chapter 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 the "Expressions" chapter in this manual.

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 the "Expressions" chapter 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.**

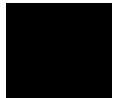
1213

!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 the "Expressions" chapter 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 the "Expressions" chapter 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 the "Expressions" chapter 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.



Chapter 12: Emulator Error Messages
!ERROR 1221! Range resource in use

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**.

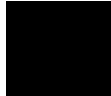
Action: Be sure that the sequence term you enter is either between existing sequence terms or after the last sequence term.

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 the chapter titled, "Using the Analyzer" 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 the chapter titled, "Using the Analyzer" for more information.



!ERROR 1241! Invalid qualifier resource or operator: <expression>

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 the chapter titled, "Using the Analyzer" 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.

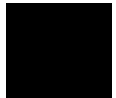
1249

!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 the "Expressions" chapter 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.
- 1254 **!ERROR 1254! Incompatible signal out events: <Incompatible Event Name>**
- Cause: The terminal interface **tgout** (trigger output) command may be used to drive the trig1 and/or trig2 signals to the emulator in response to several different events. The events are trigger recognition, measurement complete, finding a specified expression, delay after trigger recognition, and delay before measurement complete. Some of these events may be ORed together, but a delay specification may not be ORed with trigger recognition or measurement complete events.
- Action: Examine your **tgout** specification and modify it to remove ORing of delay specifications with trigger recognition or measurement complete events.



!ERROR 1255! Trig1, trig2 delay spec out of bounds: <Entered Numeric Value>

1255 **!ERROR 1255! Trig1, trig2 delay spec out of bounds: <Entered Numeric Value>**

Cause: The terminal interface **tgout** (trigger output) command provides a delay feature that allows for driving of the trig1 and/or trig2 signals a specified number of states after trigger or before trace complete. The delay value must be in the range 0 through "current analyzer depth - 1". The current analyzer depth is controlled by the terminal interface command **tcf**. Note: Use of this delay feature may cause modification of the current trigger position value.

Action: Correct the delay value in your specification so that it is within the range of 0 through "current analyzer depth -1".

1256 **!ERROR 1256! Event "expr" cannot be combined with expression definition**

Cause: The terminal interface **tgout** (trigger output) command may use an arbitrary expression as an event to drive the trig1 and/or trig2 signals to the emulator. This expression can be set up two ways. One way uses two **tgout** commands; the first command defines the signals and type of events, and the second command defines the expression. This is most useful when defining complicated expressions. The other way uses one **tgout** command which defines the expression as the event. This error message indicates that you have tried to combine the two methods.

Action: Reenter your **tgout** command using the correct format for the command. Refer to the **tgout** command description in the chapter titled "Interfaces of the Deep Analyzer" for correct formats for the **tgout** command.

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

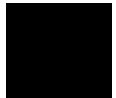
Cause: This error occurs when you attempt to specify the internal trig2 signal as the trace arm condition while the same analyzer's trigger output is currently driving the trig2 signal. This error also occurs if you attempt to specify that the trigger output drive the internal trig2 signal while that 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).



##IL# in trace list Mnemonic column

The notation ##IL# may appear in place of an inverse-assembled instruction in the Mnemonic column of a trace list. This notation indicates the inverse assembler was unable to find the information it needed to complete a trace list line. Probably, the information was not available in the trace memory.

For example, you would see the ##IL# notation in your trace list if you were tracing an Intel processor and you qualified capture of only instruction execution cycles during your trace. The inverse assembler would look for an opcode to associate with each instruction execution, but it would find none. When its search for an opcode timed out, it would place ##IL# in the Mnemonic column of the trace list.

The following trace list was obtained to show the appearance of ##IL# in a trace list Mnemonic column. The example trace list was obtained from an 80186 emulator using its terminal interface.

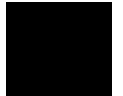
Line	addr,H	8018x mnemonic,H	count,R	seq
26200	81991	##IL#	3.00uS	.
26201	81994	##IL#	3.00uS	.
26202	81996	##IL#	1.00uS	.
26203	8197f	##IL#	3.26uS	.
26204	81984	##IL#	4.98uS	.
26205	81985	##IL#	1.02uS	.
26206	8198a	##IL#	4.00uS	.

The analyzer takes a long time to compose a trace list when the inverse assembler times out before it places each line on screen. Typically, the analyzer takes 2 seconds to place each line on screen when it fails to find the information it needs and places ##IL# on screen, instead.

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 68040 emulator does not use the timing data record types, because it does not support an external analyzer.

No Trigger Record

NO TRIGGER RECORD

10000000

BYTE 1

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

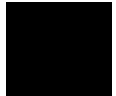
One byte indicating that the most recent trace was halted before any states were

EMPTY TRACE RECORD

01000000

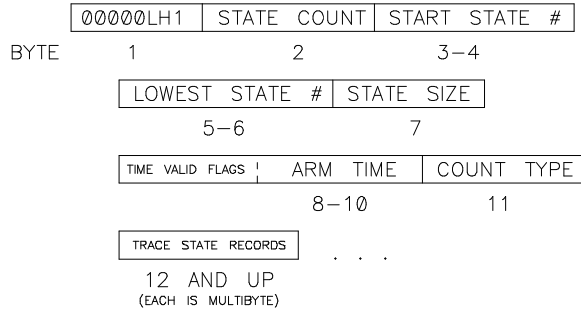
BYTE 1

stored. Therefore, this will be the only record sent.



New State Data Record

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 occurred an unknown amount of time after the trigger
0001h	Arm occurred 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.

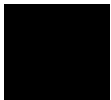
The highest four bits contain status flags as follows:

$$\text{high nibble} = XVS0$$

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.



Chapter 13: Data File Formats
More State Data Record

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.

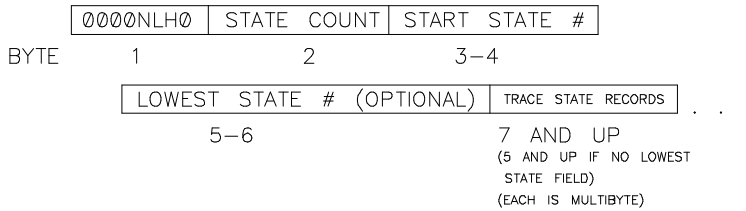
"N" indicates that no count data is available.

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

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

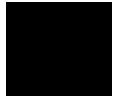
Two bytes containing the starting state number (in the range -1024..1023), most significant byte (MSB) first.

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

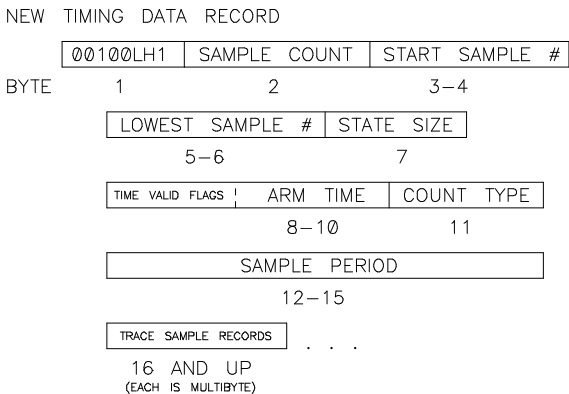
Each of these records is in the trace state format described below. Each record is a variable number of bytes in length. The length is the state size value (described above) plus one byte indicating the reason for storage of this state and an optional two bytes with count data information.



trace 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

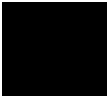


record type = 00100LH1

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.



Chapter 13: Data File Formats
New Timing Data Record

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 occurred an unknown amount of time after the trigger
0001h	Arm occurred 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.

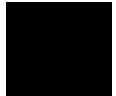
"G" indicates the timing analyzer was set to glitch mode. Each trace sample consists of a two-byte trace data field and a two-byte glitch data field.

sample period

Four bytes containing the number of nanoseconds (ns) between samples.

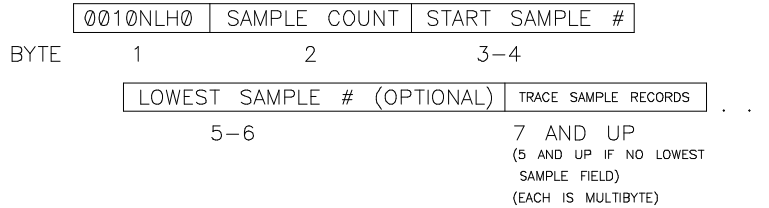
first trace sample..last trace sample

Trace Sample records of the size defined in the sample size field (note relationship to the count type field).



More Timing Data Record

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.

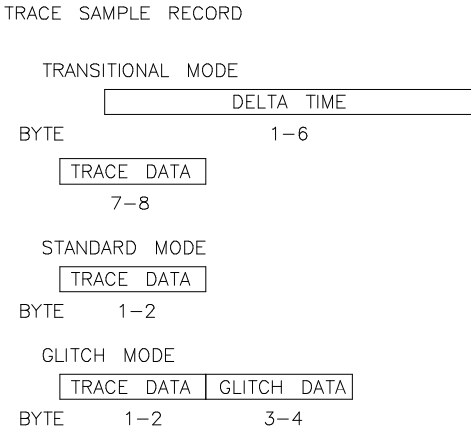
lowest sample

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

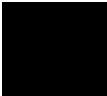


Trace Sample records are variant records which are components of the New Timing Data Record and More Timing Data Record. The structure of the Trace Sample Record depends on the count type field in the Timing Data Records.

Transitional Mode (count type = "T")

delta time

Six bytes of data defining the delta time (elapsed time) since the last transition, in nanoseconds (ns).



Chapter 13: Data File Formats

Trace Sample Records

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 64783 emulator can load an ASCII text file containing symbol definitions.

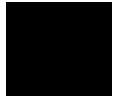
Three types of symbols can be defined: local, global, and user. Only local and global symbols can be loaded from a symbol file; user symbols can only be created with the **sym** command.

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.

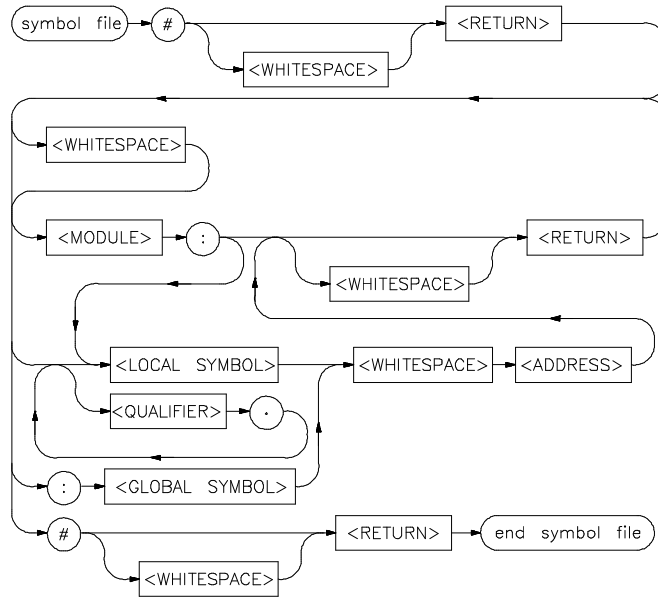
A module is usually a source file name, and symbols are function or procedure names. In a symbol file, any organizational scheme can be used to manage local symbols. While the module name can be equivalent to a source file name, or some other physical or logical entity, it is not necessary. Therefore, 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  
#
```

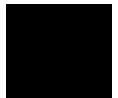
will produce the same result when loaded.

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
```



Symbol file syntax

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
```



Specifications and Characteristics

Processor Compatibility

The HP 64783A/B is compatible with the Motorola MC68040, MC68EC040, and MC68LC040 processors, and with any processors that meet all specifications of the MC68040, MC68EC040, and MC68LC040 processors.

Electrical

Maximum clock speed

The maximum external clock speed of the HP 64783A is 33 MHz, and of the HP 64783B is 40 MHz. The emulator runs without wait states at clock speeds up to 25 MHz. Above 25 MHz, one wait state is required in all bus cycles and between burst transfers.

Motorola JTAG

HP 64783A/B does not support Motorola JTAG. Therefore, no specifications are given for Motorola JTAG in this manual.

HP 64783A/B Maximum Ratings

Characteristic	Symbol	Value	Unit
Supply Voltage	V _{CC}	-0.3 to +5.5	V
Input Voltage	V _{in}	-0.5 to +5.5	V
Maximum Operating Ambient Temperature	T _A	45	°C
Minimum Operating Ambient Temperature	T _A	0	°C
Storage Temperature Range	T _{stg}	-40 to +70	°C

HP 64783A/B Electrical Specifications

HP 64783A/B — DC ELECTRICAL SPECIFICATIONS (V _{CC} =5.0 Vdc ±5%)				
Characteristic	Symbol	Min	Max	Unit
Input High Voltage	V _{IH}	2	V _{CC}	V
Input Low Voltage	V _{IL}	GND	0.8	V
Undershoot		—	0.5	V
<u>Input Leakage Current @ 0.5/2.4 V</u> A _{VEC} , B _{CLK} , B _G , C _{DIS} , M _{DIS} , I _{PLx} , P _{CLK} , R _{STI} , S _{Cx} , T _{BI} , T _{LNx} , T _{CI} , T _{CK} , T _{EA}	I _{IL} I _{IH}	-250 —	— 25	μA
<u>Hi-Z (Off-State) Leakage Current @ 0.5/2.4 V</u> A _n , C _{IOU_T} , D _n , L _{OCK} , L _{OCKE} , S _{IZx} , T _{DO} , T _{Mx} , T _{LNx} , T _{Tx} , U _{PAx} B _B , R _W , T _{IP} , T _S T _A	I _{TSI}	— -50 -100 -200	— 50 100 200	μA
Output High Voltage I _{OH} = -32 mA: A _n , D _n , S _{IZx} , T _{Tx} , U _{PAx} , L _{OCK} , L _{OCKE} , T _{LNx} , C _{IOU_T} , T _{Mx} , P _{STx} , R _{STO} , B _R , M _I , B _G , reset flying lead I _{OH} = -3.2 mA: R _W , T _S , T _{IP} , B _B , T _A , I _{PEND}	V _{OH}			V
Output Low Voltage I _{OL} = 64 mA A _n , D _n , S _{IZx} , T _{Tx} , U _{PAx} , L _{OCK} , L _{OCKE} , T _{LNx} , C _{IOU_T} , T _{Mx} , P _{STx} , R _{STO} , B _R , M _I , B _G , reset flying lead I _{OL} = 24 mA R _W , T _S , T _{IP} , B _B , T _A , I _{PEND}	V _{OL}			V
Capacitance V _{in} =0 V, f=1 MHz	C _{in}	—	25	pF

Chapter 14: Specifications and Characteristics
HP 64783A/B Electrical Specifications

HP 64783A/B — DC ELECTRICAL SPECIFICATIONS (V _{CC} =5.0 Vdc ±5%)				
Characteristic	Symbol	Min	Max	Unit
Supply Current	I _{CC}			
f = 25 MHz		—	1.4	A
f = 33 MHz		—	1.8	A

Notes for HP 64783A/B Electrical Specifications:

BCLK and PCLK have additional input current and capacitance loading because of RC terminations. Refer to their equivalent circuit diagrams for details. The numbers given in the HP 64783A/B Electrical Specifications table do not include the RC terminations.

HP 64783A/B Clock AC Timing Specifications

Num	Characteristic	25 MHz		33 MHz		40 MHz		Unit
		Min	Max	Min	Max	Min	Max	
	Frequency of Operation	16.67	25	16.67	33	20	40	MHz
1	PCLK Cycle Time	20	30	15	30	12.5	25	ns
2	PCLK Rise Time		1.7		1.7	—	1.5	ns
3	PCLK Fall Time		1.6		1.6	—	1.5	ns
4	PCLK Duty Cycle Measured at 1.5 V	47.50	52.50	46.67	53.33	46.0	54.00	%
4a ¹	PCLK Pulse Width High Measured at 1.5 V	9.50	10.50	7	8	5.75	6.75	ns
4b ¹	PCLK Pulse Width Low Measured at 1.5 V	9.50	10.50	7	8	5.75	6.75	ns
5	BCLK Cycle Time	40	60	30	60	25	50	ns
6,7	BCLK Rise and Fall Time	—	4	—	3	—	3	ns
8	BCLK Duty Cycle Measured at 1.5 V	40	60	40	60	40	60	%
8a ¹	BCLK Pulse Width High Measured at 1.5 V	16	24	12	18	10	15	ns
8b ¹	BCLK Pulse Width Low Measured at 1.5 V	16	24	12	18	10	15	ns
9	PCLK, BCLK Frequency Stability	—	1000	—	1000	—	1000	ppm
10	PCLK to BCLK Skew	—	n/a	—	n/a	—	n/a	ns

Notes for Clock AC Timing Specifications:

Chapter 14: Specifications and Characteristics
HP 64783A/B Output AC Timing Specifications

1 Specification value at maximum frequency of operation.

HP 64783A/B Output AC Timing Specifications

Num	Characteristic	25 MHz ¹		33 MHz ¹		40 MHz ¹		Unit
		Min	Max	Min	Max	Min	Max	
11	BCLK to Address $\overline{\text{CIOUT}}$, $\overline{\text{LOCK}}$, $\overline{\text{LOCKE}}$, $\overline{\text{R/W}}$, SIZx , TLNx , TMx , TTx , UPAx Valid	9	25	6.5	22.5	5.25	21	ns
12	BCLK to Output Invalid (Output Hold)	9	—	6.5	—	5.25	—	ns
13	BCLK to $\overline{\text{TS}}$ Valid	9	25	6.5	22.5	5.25	21	ns
14	BCLK to $\overline{\text{TIP}}$ Valid	9	25	6.5	22.5	5.25	22	ns
18	BCLK to Data Out Valid	9	27	6.5	24.5	5.25	23	ns
19	BCLK to Data Out Invalid (Output Hold)	9	—	6.5	—	5.25	—	ns
20	BCLK to Output Low Impedance	3	—	3	—	3	—	ns
21	BCLK to Data-Out High Impedance	9	32	6.5	27	5.25	24.5	ns
26 ²	BCLK to Multiplexed Address Valid	n/a	n/a	n/a	n/a	n/a	n/a	ns
27 ²	BCLK to Multiplexed Address Driven	n/a	—	n/a	—	n/a	—	ns
28 ²	BCLK to Multiplexed Address High Impedance	n/a	n/a	n/a	n/a	n/a	n/a	ns
29 ²	BCLK to Multiplexed Data Driven	n/a	—	n/a	—	n/a	—	ns
30 ²	BCLK to Multiplexed Data Valid	n/a	n/a	n/a	n/a	n/a	n/a	ns

Chapter 14: Specifications and Characteristics
HP 64783A/B Output AC Timing Specifications

Num	Characteristic	25 MHz ¹		33 MHz ¹		40 MHz ¹		Unit
		Min	Max	Min	Max	Min	Max	
38	BCLK to $\overline{\text{Address}}$, $\overline{\text{CIOUT}}$, $\overline{\text{LOCK}}$, $\overline{\text{LOCKE}}$, R/W, $\overline{\text{SIZx}}$, $\overline{\text{TS}}$, $\overline{\text{TLNx}}$, $\overline{\text{TMx}}$, $\overline{\text{TTx}}$, UPAx High Impedance	9	31	6.5	26	5.25	23.5	ns
39	BCLK to $\overline{\text{BB}}$, $\overline{\text{TA}}$, $\overline{\text{TIP}}$ High Impedance	19	31	14	26	11.5	23.5	ns
40	BCLK to $\overline{\text{BR}}$, $\overline{\text{BB}}$ Valid	9	25	6.5	22.5	5.25	21	ns
43	BCLK to $\overline{\text{MI}}$ Valid	9	25	6.5	22.5	5.25	21	ns
48	BCLK to $\overline{\text{TA}}$ Valid	9	25	6.5	22.5	5.25	21	ns
50	BCLK to $\overline{\text{IPEND}}$, $\overline{\text{PSTx}}$, $\overline{\text{RSTO}}$ Valid	9	25	6.5	22.5	5.25	21	ns

Notes:

- 1 Output timing is given for output drivers specified in the DC specs (Refer to the table of HP 64783A/B Electrical Specifications). Large/small buffer mode select has no effect.
- 2 Address multiplex mode is not supported.

HP 64783A/B Input AC Timing Specifications

Num	Characteristic	25 MHz		33 MHz		40 MHz		Unit
		Min	Max	Min	Max	Min	Max	
15	Data-In Valid to BCLK (Setup)	9	—	9	—	8	—	ns
16	BCLK to Data-In Invalid (Hold)	4	—	4	—	3	—	ns
17	BCLK to Data-In High Impedance (Read Followed by Write)	—	49	—	36.5	—	30.25	ns
22a	$\overline{\text{TA}}$ Valid to BCLK (Setup)	15	—	15	—	13	—	ns
22b	$\overline{\text{TEA}}$ Valid to BCLK (Setup)	15	—	15	—	14	—	ns
22c	$\overline{\text{TCI}}$ Valid to BCLK (Setup)	15	—	15	—	14	—	ns
22d	$\overline{\text{TBI}}$ Valid to BCLK (Setup)	15	—	15	—	14	—	ns
23	BCLK to $\overline{\text{TA}}$, $\overline{\text{TEA}}$, $\overline{\text{TCI}}$, $\overline{\text{TBI}}$ Invalid (Hold)	2	—	2	—	2	—	ns
24	$\overline{\text{AVEC}}$ Valid to BCLK (Setup)	10	—	10	—	10	—	ns
25	BCLK to $\overline{\text{AVEC}}$ Invalid (Hold)	2	—	2	—	2	—	ns
31 ¹	DLE Width High	n/a	—	n/a	—	n/a	—	ns
32 ¹	Data-In Valid to DLE (Setup)	n/a	—	n/a	—	n/a	—	ns
33 ¹	DLE to Data-In Invalid (Hold)	n/a	—	n/a	—	n/a	—	ns
34 ¹	BCLK to DLE Hold	n/a	—	n/a	—	n/a	—	ns
35 ¹	DLE High to BCLK	n/a	—	n/a	—	n/a	—	ns

Chapter 14: Specifications and Characteristics
HP 64783A/B Input AC Timing Specifications

Num	Characteristic	25 MHz		33 MHz		40 MHz		Unit
		Min	Max	Min	Max	Min	Max	
36 ¹	Data-In Valid to BCLK (DLE Mode Setup)	n/a	—	n/a	—	n/a	—	ns
37 ¹	BCLK to Data-In Invalid (DLE Mode Hold)	n/a	—	n/a	—	n/a	—	ns
41a	$\overline{\text{BB}}$ Valid to BCLK (Setup)	12	—	12	—	12	—	ns
41b	$\overline{\text{BG}}$ Valid to BCLK (Setup)	12	—	12	—	12	—	ns
41c	$\overline{\text{CDIS}}$, $\overline{\text{MDIS}}$ Valid to BCLK (Setup)	13	—	13	—	13	—	ns
41d	$\overline{\text{IPLx}}$ Valid to BCLK (Setup)	8	—	8	—	8	—	ns
42	BCLK to $\overline{\text{BB}}$, $\overline{\text{BG}}$, $\overline{\text{CDIS}}$, $\overline{\text{IPLx}}$, $\overline{\text{MDIS}}$ Invalid (Hold)	2	—	2	—	2	—	ns
44a	Address Valid to BCLK (Setup)	12	—	12	—	12	—	ns
44b	SIZx Valid to BCLK (Setup)	13	—	13	—	13	—	ns
44c	TTx Valid to BCLK (Setup)	13	—	13	—	13	—	ns
44d	$\overline{\text{R/W}}$ Valid to BCLK (Setup)	10	—	10	—	10	—	ns
44e	SCx Valid to BCLK (Setup)	16	—	16	—	13	—	ns
45	BCLK to Address, SIZx, TTx, $\overline{\text{R/W}}$, SCx Invalid (Hold)	2	—	2	—	2	—	ns
46	$\overline{\text{TS}}$ Valid to BCLK (Setup)	14	—	14	—	12	—	ns
47	BCLK to $\overline{\text{TS}}$ Invalid (Hold)	2	—	2	—	2	—	ns
49	BCLK to $\overline{\text{BB}}$ High Impedance (MC68040 Assumes Bus Mastership)	—	9	—	9	—	9	ns

Chapter 14: Specifications and Characteristics
HP 64783A/B Input AC Timing Specifications

Num	Characteristic	25 MHz		33 MHz		40 MHz		Unit
		Min	Max	Min	Max	Min	Max	
51	$\overline{\text{RSTI}}$ Valid to BCLK	9	—	9	—	9	—	ns
52	BCLK to $\overline{\text{RSTI}}$ Invalid	2	—	2	—	2	—	ns
53 ²	Mode Select Setup to $\overline{\text{RSTI}}$ Negated	n/a	—	n/a	—	n/a	—	ns
54 ²	$\overline{\text{RSTI}}$ Negated to Mode Selects Invalid	n/a	—	n/a	—	n/a	—	ns

Notes:

- 1 Data Latch mode is not supported.
- 2 Mode selects are not used.

Physical

Emulator Dimensions

173 mm height x 325 mm width x 389 mm depth (6.8 in. x 12.8 in. x 15.3 in.)

Emulator Weight

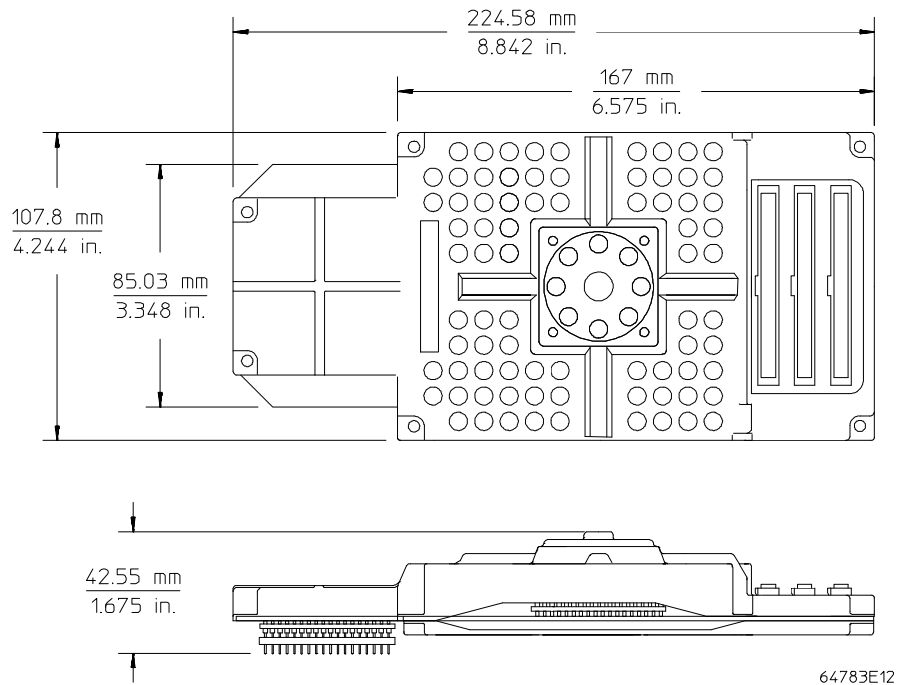
HP 64783A/B, 8.2 kg (18 lb). Any component used in suspending the emulator must be rated for 30 kg (65 lb) capacity.

Probe alone: 0.3 kg (10 oz).

Cable Length

Emulation Control Card to Probe, approximately 914 mm (36 inches).

Probe dimensions



Environmental

Temperature

Operating, 0° to +40° C (+32° to +104° F); nonoperating, -40° C to +60° C (-40° F to +140° F).

Altitude

Operating/nonoperating 4600 m (15 000 ft).

Relative Humidity

15% to 95%.

BNC, labeled TRIGGER IN/OUT

Output Drive

Logic high level with 50-ohm load ≥ 2.0 V. Logic low level with 50-ohm load ≤ 0.4 V.

Input

74HCT132 with 135 ohms to ground in parallel. Maximum input: 5 V above V_{cc} ; 5 V below ground.



Communications

Host Port

25-pin female type “D” subminiature connector.

RS-232-C DCE or DTE to 38.4 kbaud.

RS-422 DCE only to 460.8 kbaud.

CMB Port

9-pin female type “D” subminiature connector.

Part 4

Installation and Service

In This Part

Chapter 15, "Connecting the Emulator to a Target System," tells you how to connect the emulator into an MC68040 target system and overcome the differences between the specifications and characteristics of the target microprocessor and those of the emulator.

Chapter 16, "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.

Chapter 17, "Installing/Updating Emulator Firmware," tells you how to install, update, and verify the firmware of your emulator.

When you finish installation of the emulator, go to part 1 of this manual and perform the Quick Start procedure.



Connecting the Emulator to a Target System

Things you need to know to successfully connect the emulator to a target system and overcome problems you may encounter.

Plugging The Emulator Into A Target System

The following paragraphs help you understand the emulator. Equivalent circuits are shown, followed by a list of devices that you may need to use to overcome mechanical and electrical constraints in your target system.

Understanding an emulator

An emulator is a tool intended for debugging software, and the interactions between software and hardware. Although emulators can help in debugging certain hardware problems, catastrophic problems often require use of other tools, such as a timing analyzers with preprocessors, or oscilloscopes. To effectively use an emulator, you need to understand its capabilities and limitations, and how it interacts with your target system. This chapter discusses limitations and interactions of an emulator, as they relate to your target system.

An emulator is designed to be electrically and functionally equivalent to the processor it emulates, as much as possible. Most MC68040 signals are electrically isolated from their counterparts on the target system connection. This is done for both electrical and functional reasons. Equivalent circuits of each processor signal are shown later in this chapter. The impacts of these circuits are calculated and presented in the emulator specifications listed in the chapter titled "Specifications and Characteristics" in this manual.

In the ideal case, you would use the emulator specifications listed in this manual when designing your target system instead, of the processor specifications. In the typical case, your target system has already been designed and prototyped. A target system that is designed around MC68040 worst case specifications will typically work with the emulator. If certain circuits in your target system do not allow for variations in the MC68040 specifications, compare the relevant emulator specifications to evaluate their impact on your target system. By keeping the differences between emulator specifications and processor specifications in mind while you design your target system, you can save hours of debugging time when you plug the emulator into your target system.

Chapter 15: Connecting the Emulator to a Target System

Plugging The Emulator Into A Target System

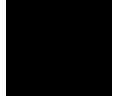
The MC68040 emulator does not switch between large and small buffer modes like the MC68040 processor does. The emulator internally uses the large buffer mode to get optimum timing performance. Since these large drivers can cause problems for systems designed to work with small buffer mode, the emulator buffers all signals from the processor to the target connector. Most of the signals are buffered in ABT logic family parts. These parts are chosen to provide high speed and high current capability while keeping slew rates to an acceptable level for small buffer mode systems. Some control signals are buffered in PALs which have significantly less drive capability than the processor in large mode.

Examine the DC specifications of the emulator to evaluate their differences from processor specifications. Again, you can refer to the equivalent circuit diagrams in this chapter for exact details. Because the emulator does not behave exactly like the processor, you may need to examine signal quality and take appropriate steps to compensate for differences.

The BCLK clock is the most important signal to the emulator because all system timing is derived from this signal. The BCLK clock signal must have clean edges; the duty cycle of this clock is not particularly important. The emulator regenerates an internal BCLK from this signal with a 50% duty cycle. All timing is referenced from the rising edge of BCLK. The PCLK clock is also internally regenerated; therefore, the emulator is not sensitive to this signal.

Both the BCLK and PCLK signals are terminated on the emulator. The terminations are placed on these signals, even though the emulator causes only a short electrical stub, so that accessories such as the flexible cable can be used to connect the emulator probe to your target system. The terminations on these signals can interact with terminations on your target system. Refer to the equivalent circuits in this chapter and adjust terminations in your target system for best results.

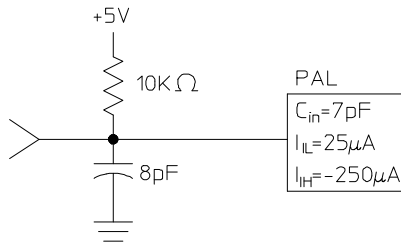
The emulator uses power from the target system to operate the emulation processor and some pullup resistors. Target power is sensed to make sure the emulator does not drive the target system until it is powered up. In addition, the power detection circuit delays release of processor reset for 50 ms after power is in specification to allow the clock circuits to synchronize. Because of the protections designed into the emulator, always power on the emulator before the target system and power off the emulator after the target system.



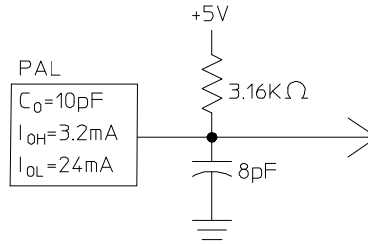
Equivalent circuits

The equivalent circuits shown on this page and the next help you understand connection requirements between the emulator probe and your target system.

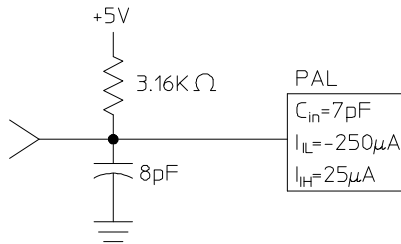
\overline{AVEC} , \overline{MDIS} , \overline{TBI} , \overline{TCI} ,
 $\overline{IPL}(2:0)$, \overline{CDIS}



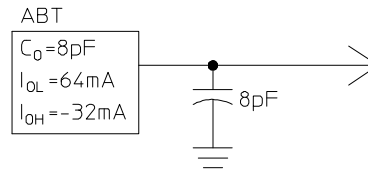
\overline{TIP} , \overline{IPEND}



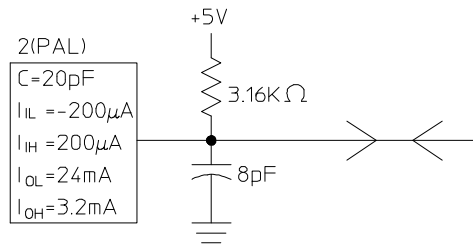
\overline{TEA} , \overline{RSTI}



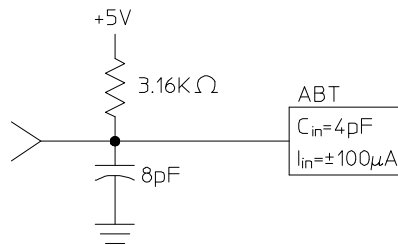
\overline{MI} , \overline{BR} , \overline{RSTO} , $\overline{PST}(3:0)$,
 $\overline{TM}(210)$, \overline{CIOUT} , $\overline{TLN}(1:0)$,
 \overline{LOCK} , \overline{LOCKE} , $\overline{UPA}(1:0)$



\overline{TA}



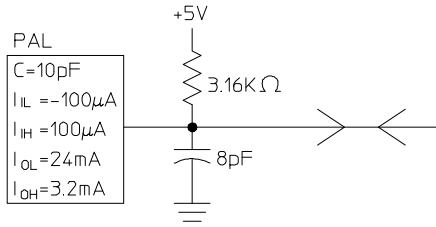
\overline{BG}



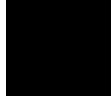
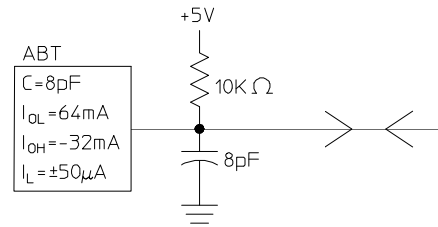
64783B01

Chapter 15: Connecting the Emulator to a Target System
Plugging The Emulator Into A Target System

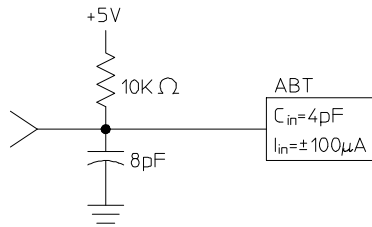
R/ \bar{W} , $\bar{T}\bar{S}$, $\bar{B}\bar{B}$



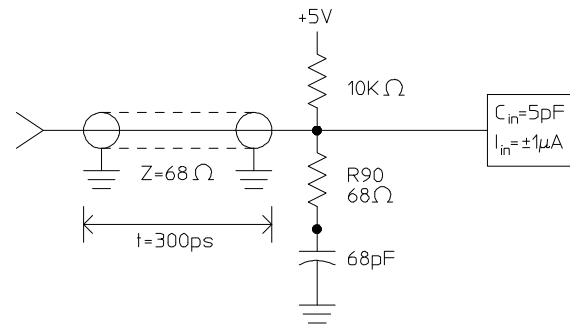
D(31:0), A(31:0),
 TT(1:0), SIZ(1:0)



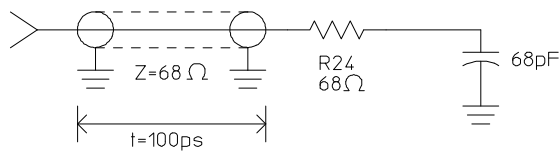
SC(1:0)



BCLK



PCLK

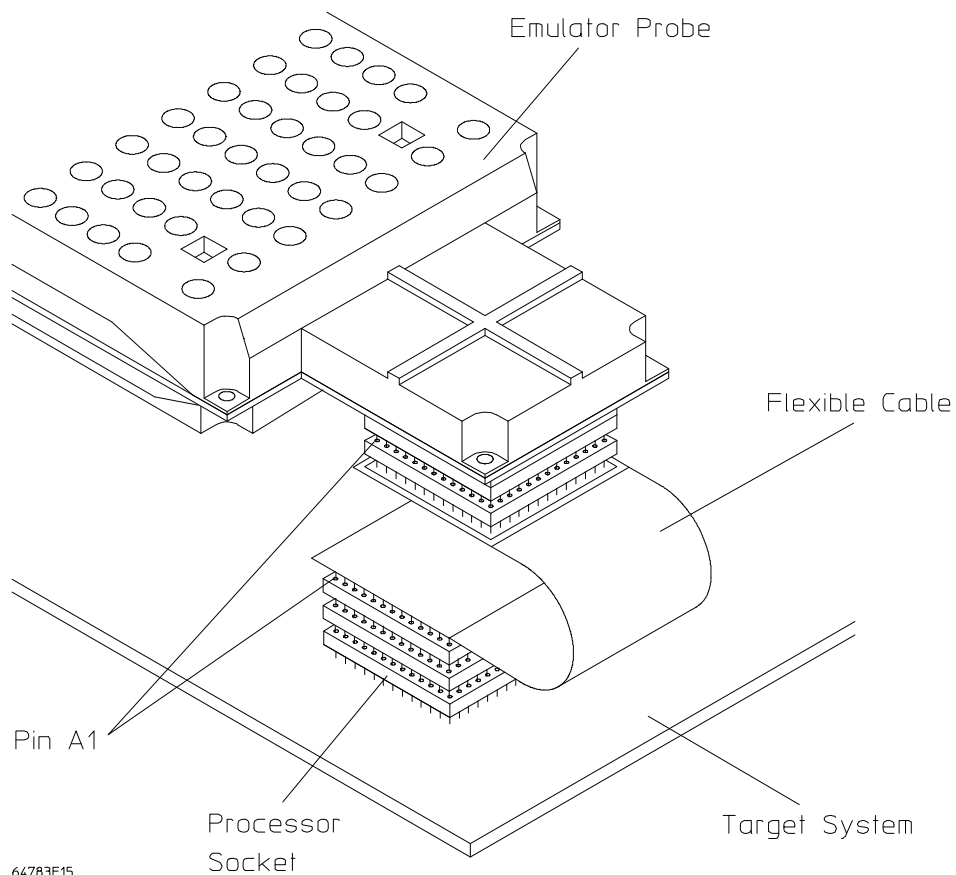


64783B02

Connecting the emulator to the target system

Plugging the emulator into a target system can be difficult because of mechanical constraints. If the mechanical constraints cannot be removed so that the emulator can be plugged directly into the target socket, there are several accessories available to help with the connection. These accessories are:

- Stacking pin protectors.
- PGA rotators, available from Emulation Technology.
- PGA to PGA Flexible Adapter (see below), HP Part Number E3429A.

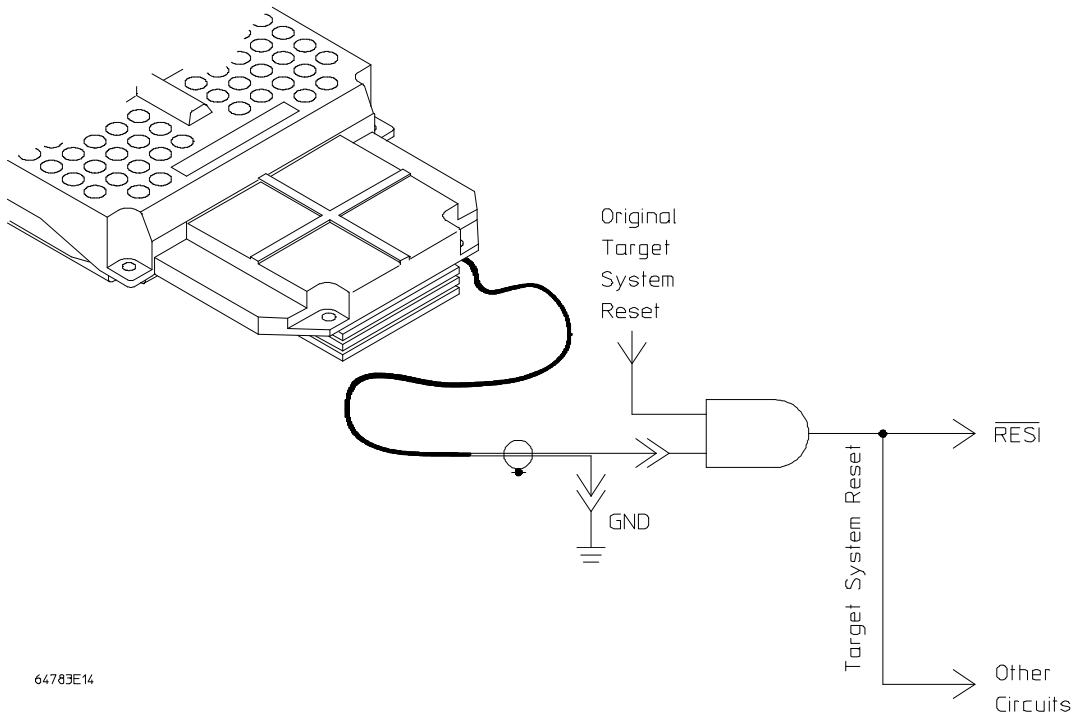


64783E15

Chapter 15: Connecting the Emulator to a Target System Plugging The Emulator Into A Target System

Unfortunately, these accessories have an electrical impact on your target system. The specifications given for the emulator do not include the impact of these accessories. In addition to delays, the accessories can cause problems with signal quality. Only use these accessories as a last resort.

An optional Reset Flying Lead is provided with the emulator. It can be used to reset the target system when the `rst` command is used. The signal is driven low when the emulator is in its reset state ("R>" prompt on screen). In addition, the signal will pulse low when a `r rst` or `rst -m` command is issued, if the emulator is not already in the reset state. The signal carried by the Reset Flying Lead is intended to be used to initialize circuitry in your target system that would normally be reset along with the processor (see below).



The example circuit shows the emulator reset signal being ANDed with a target system reset signal to generate a new target system reset signal. This new signal will reset the processor and other circuits on the target system when either the emulator asserts reset, or the target system generates reset.

Verifying Operation Of The Emulator In Your Target System

When connecting an emulator into a new target system, the step-by-step approach described in the remainder of this chapter will help you get your system running most quickly. This is a logical procedure that starts out with the most simple requirements and moves toward complete functionality, allowing for verification of installation at each step of the way. This not only helps debug problems if they arise, but builds confidence that the emulator is functioning correctly in your target system.

To begin, run the performance verification procedure described in the Installation and Service Chapter in this manual.

Some additional equipment may be required to make measurements of MC68040 signals. It will help to have an oscilloscope and high speed timing analyzer to use during these procedures. A 250-MHz timing analyzer may be fast enough, but faster is better. The oscilloscope should have a single-shot bandwidth greater than 500 MHz. You may also need to cross trigger these instruments from the emulator. If there are no trigger inputs to the timing analyzer, you can probably use a timing channel. The BNC trigger output of the 64700 emulation card cage provides a rising edge TTL signal.

When making measurements, remember that signals need to be probed at the right place for the measurement being made. The emulator specifications are referenced to the target socket connector on the probe. This is where measurements should be made to verify compliance with the specifications. When probing setup and hold times to circuits in the target system, make the appropriate measurements at the circuits. This will keep connection accessories from impacting the true measurements. Always use ground leads to get the most accurate measurements possible.

Running the emulator configured like the processor

This step uses no emulation monitor, or emulation memory, and does not attempt to control any of the processor signals. For this test, the only emulation feature that is operating is the emulation-bus analyzer. The emulation-bus analyzer is passive, like a preprocessor. The main purpose of this step is to determine whether the loading and timing changes of the emulator impact your target system.

If your target system can run a program without the emulator, do this procedure. Otherwise, go to step 2.

- 1 Turn on power to the emulator.
- 2 Check the emulator prompt by pressing <RETURN>.

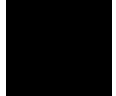
The prompt should be "p>". A prompt of "->" indicates a software compatibility problem. Correct problems indicated in error messages (seen in the emulator error log) or check the software version using the **ver** command for more information.

- 3 Configure the emulator by entering the following commands:

```
cf mon=none
cf cache=en
cf mmu=en
cf ti=en
cf wait=<en,dis>, as appropriate for your target system
```

- 4 Set up the emulation-bus analyzer to capture all MC68040 system cycles.

```
tck -u
tg any
tsto any
tp c
t
```



Chapter 15: Connecting the Emulator to a Target System

Verifying Operation Of The Emulator In Your Target System

5 Execute your program with the command: **r rst**.

This tells the emulator to deassert reset so that the emulator does not interfere with the target system powerup reset.

6 Power on the target system.

7 Verify correct operation.

The target system should run just as if the processor was being used. If your target system performs any I/O, check it to see if your system performs it correctly. If your target system appears to work correctly, allow it to reach its stable operating temperature and test it again.

If the target system appears to work correctly, go to the paragraph titled, "Installing the Monitor", later in this chapter. Otherwise, verify operation of the target system as described next.

To verify operation of the target system

Get the prompt by pressing <RETURN>, or use the command **es** to get more information about the emulator status. If the system is working the prompt will normally be "U>", but there are a few situations where the system will be working properly and the prompt will be something different. If the bus is taken away from the MC68040 often or for long periods of time, the emulator can display the "g>" prompt or alternate between "g>" and "U>". If the MC68040 is running code in its internal cache for long periods of time, the emulator may display the "b>" prompt. The emulator may alternate between any of these prompts during normal operation.

All other prompts usually indicate a problem. Even the "g>" or "b>" prompts can indicate a problem. To understand problems indicated by the prompts, you need to know whether bus cycles were executed, how many bus cycles were executed, what type of bus cycles were executed, and whether the target system is still executing bus cycles. You can tell the difference between these conditions by checking the trace status to see if any bus cycles were captured. The analyzer may have states in its internal pipeline that will not be reported until the trace is halted.

Chapter 15: Connecting the Emulator to a Target System Verifying Operation Of The Emulator In Your Target System

```
b>th;ts
  Emulation trace halted
  --- Emulation Trace Status ---
  User trace halted                <- trace status
  Arm ignored
  Trigger not in memory
  Arm to trigger ?
  States 0 (0) ?..?                <- number of states captured
  Sequence term 2
  Occurrence left 1
b>
```

If the trace status indicates that the trace was halted, look at the number of states collected to decide how many bus cycles were executed. If the status indicates that the user trace was completed, a large number of states were executed. If this is the case, it may help to take another trace to see if bus cycles are still being executed. Again, view the trace status to determine if bus cycles are executing.

If the "p>" prompt remains after target powerup, check:

- mechanical installation of the probe.
- blown fuses.
- target system power supply voltage.

If the prompt is "c>", mechanical installation may be causing the problem, but the most likely cause is a problem with the clock. Check clock quality. Look at the voltage levels, edges, and duty cycle. If the clock looks suspect, compare it to the target system clock without the emulator. If there is a significant difference, you may need to adjust the target system terminations to account for the emulator's termination.

If the prompt is "r>", either the target system never released reset, or the target system reset itself because of some program error condition. If no bus cycles were captured by the analyzer, the target system never released reset. You need to find out which conditions must occur to release reset, and then investigate these conditions to determine why reset isn't being released.

An example of a failure to release reset might be a multichip system where the master chip starts the slave chips after verifying that they are installed in the system by reading checksums from their ROMs. If a checksum is not read correctly, reset to the associated slave chip is not released. If the emulator interfered with the reading of the checksum, then reset would not be released.

Chapter 15: Connecting the Emulator to a Target System

Verifying Operation Of The Emulator In Your Target System

One thing to keep in mind is that the emulator does not trace alternate bus master cycles while it is reset.

If any bus cycles were executed before the reset occurred, then something caused the target system to reassert the reset condition. Usually, this is caused by some type of fault which is detected by the system. This may result from access to a certain address range or because of a watchdog timeout. Refer to "Interpreting the Trace List", later in this chapter, to help you understand what caused the reset.

If the prompt is "b>", and there are no cycles in the trace list, the processor never attempted to run any bus cycles even if other indications show it should have. This could indicate problems with power, clock, or signal transitions, especially the reset signal. Check power supply voltage levels. Make sure the power up is monotonic. Check clock quality. Check that the reset signal meets its required assertion time after power up and clock stabilization. Check signal quality on the reset signal, especially the signal transitions.

If some cycles were captured in the trace list, but no cycles are occurring now, check for setup and hold violations on the processor strobes. All MC68040 signals, except the interrupt lines and reset signal, are synchronous to the clock and have to be valid for all rising edges of BCLK. Check timing inputs to the emulator, such as TA, TEA, and TBI, for setup and hold violations. The "b>" prompt is not a normal condition for the processor when you find no functional reason. It usually indicates that the processor has malfunctioned.

One possible cause of a "b>" prompt is the processor missing the end-of-cycle indication during the cycle of an alternate bus master. The processor monitors the TS signal during alternate bus master activity to see if it needs to intervene in the cycle (snooping). If the processor sees a TS signal but misses the corresponding TA signal, the processor may hang, waiting for this bus cycle to complete, even though the bus was granted to the MC68040 and released.

If bus cycles are occurring, then the "b>" prompt only indicates that bus cycles are infrequent. A type of system that would exhibit this behavior would be an interrupt-driven system. When done processing an interrupt, the system could execute a STOP instruction to wait for the next interrupt. If the interrupts were infrequent a "b>" prompt would be displayed.

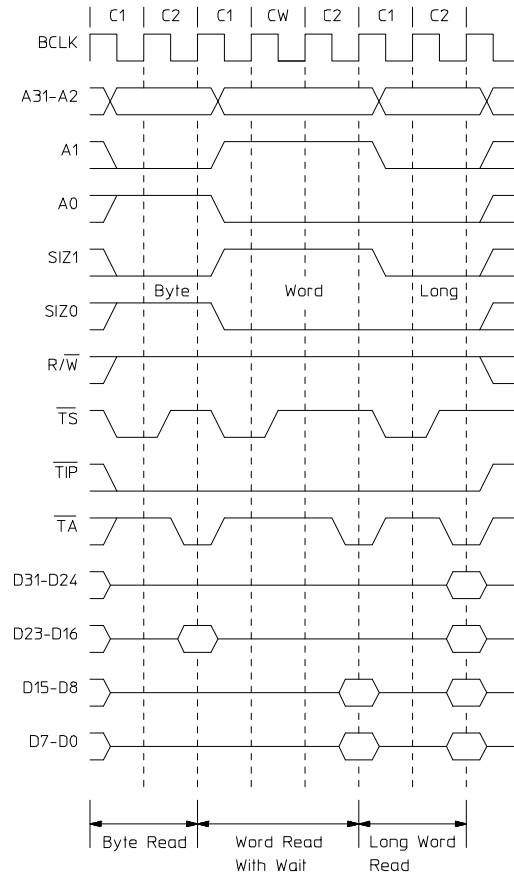
If the prompt is "w>", the emulator has stopped in the middle of a bus cycle. Get the emulation status; it will tell you the address and the type of cycle.

Chapter 15: Connecting the Emulator to a Target System Verifying Operation Of The Emulator In Your Target System

```
w>es
M68040--CPU in wait state; 00badad00@sd long read
w>
```

To troubleshoot the above problem, you need to know if the target system provides bus termination for the address. If the answer is no, then the target program must have run incorrectly. The emulation-bus analyzer will have to be used to investigate further. If the answer is yes, then the reason the bus cycle did not complete must be determined, as described next.

There are many reasons why bus cycle interaction between a target system and an emulator may fail. Usually the cause is that the target system missed the start-of-cycle indication from the emulator, or that the emulator missed the cycle-termination indication from the target system. For a better idea of what is going on, refer to the MC68040 bus cycle diagram, below:



Chapter 15: Connecting the Emulator to a Target System

Verifying Operation Of The Emulator In Your Target System

A basic MC68040 bus cycle starts with the transfer start signal, \overline{TS} . The \overline{TS} signal pulses low for about one clock cycle. Another signal, transfer in progress \overline{TIP} stays low throughout the cycle, but is not necessarily deasserted between cycles. The end of the cycle occurs when the processor samples a transfer acknowledge \overline{TA} and/or a transfer error acknowledge \overline{TEA} on the rising edge of the clock. Because of the nature of these signals, most systems are synchronous to the clock. The typical system will sample \overline{TS} on the rising clock edge and then generate a \overline{TA} signal an integral number of clocks later. Wait states are added to a cycle by delaying when the \overline{TA} is asserted.

If the emulator is configured for wait states (BCLK >25 MHz), then a compatibility problem with the emulator may be stalling the processor.

```
w>cf
  cf cache=en
  cf mmu=en
  cf mon=none
  cf rrt=dis
  cf ti=en
  cf wait=en                                <- configuration for wait states
w>
```

The emulator requires at least one wait state in all bus cycles when it is configured as above. The emulator does not add this wait state, but will not accept a \overline{TA} from the target system until after a wait state has been added. If \overline{TA} is asserted by the target system during the wait state period and is then deasserted before the emulator allows termination, the bus cycle will never complete.

This particular example can be easily duplicated on the demo board by configuring for wait states and interlocking memory to the demo board.

```
cf wait=en
map 0..0ff eram lock
r rst
```

If there is no functional reason why the bus cycle would not complete, check the timing relationships between the various bus cycle control signals. Probably the first measurement you will want to make is to see if the setup time of \overline{TA} to BCLK is within the emulator specification.

If there are no cycles in the trace list, then the processor stopped during the first bus cycle. In this case, it is pretty easy to set up the trace using \overline{TS} as the trigger because the cycle of interest is the first cycle. If there are only a few cycles in the trace list, the same technique can be used if the oscilloscope or timing analyzer has enough depth.

Chapter 15: Connecting the Emulator to a Target System

Verifying Operation Of The Emulator In Your Target System

If there are many cycles in the trace list before the processor stalled, use a different method of triggering. There are a number of different approaches that can be used. The most direct method is to trigger on a condition of TIP low and TA high for a period of time greater than the length of a memory cycle. Another method is to determine if the system always stops at the same address. This address can then be used as the trigger. One drawback to this method is that you may have to probe a large number of signals to get a unique address.

A better way would be to use the emulation-bus analyzer to generate a trigger. Unfortunately, because the cycle never finishes, the emulation-bus analyzer will not capture this address, so something preceding this event must be used as the trigger. Examine the trace list to find a unique event to use as the trigger. Once you have specified the trigger, you need to configure the emulator to drive the trigger out. The real trick to crosstriggering is to correlate the trigger event to the captured data. In this type of measurement, the correlation is easy because the signals of interest stop transitioning shortly after the trigger occurs.

```
tg addr=00badad00
tp c
tgout trig2
bnct -r trig2
t
```

Once you have a trace of the offending cycle, verify that TA is present for a valid rising clock edge, taking into account a wait state if running faster than 25 MHz. If TA looks reasonably correct, verify the setup and hold specifications. If TA occurs but on an invalid clock edge, you may need to make modifications to the target system to ensure that there is at least one wait state in target cycles. If TA is not asserted at all, it could be an indication that the target system missed the TS. Set up your oscilloscope or logic analyzer to make a measurement on your cycle start circuitry to determine why the target system did not respond to the cycle.

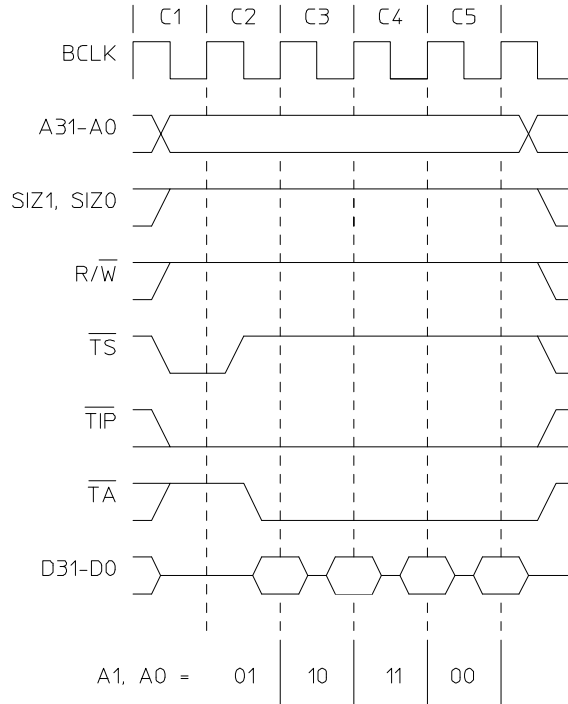
If the cycle where processing stops is part of a burst cycle, as indicated by the line access type in the status display, there are several things to check.

```
w>es
  M68040--CPU in wait state; 000000000@sd line read
w>
```

A burst cycle is shown below. The main characteristic of a burst cycle is that there are four data transfers as part of one cycle. The processor puts out an address and asserts TS only once during the cycle. A burst request is indicated by the SIZ_x signals. The target memory system can inhibit the burst cycle by asserting the TBI signal. If the cycle is inhibited, the timing becomes just like a normal cycle. If the cycle is not inhibited, once TS has been asserted, the process starts sampling TA for

Chapter 15: Connecting the Emulator to a Target System
Verifying Operation Of The Emulator In Your Target System

each data transfer. The cycle is not over until the fourth \overline{TA} is received. When the emulator has wait states enabled, a wait state is required between each of the data transfers in the burst cycle. Evaluating the timing is the same as for a normal cycle.

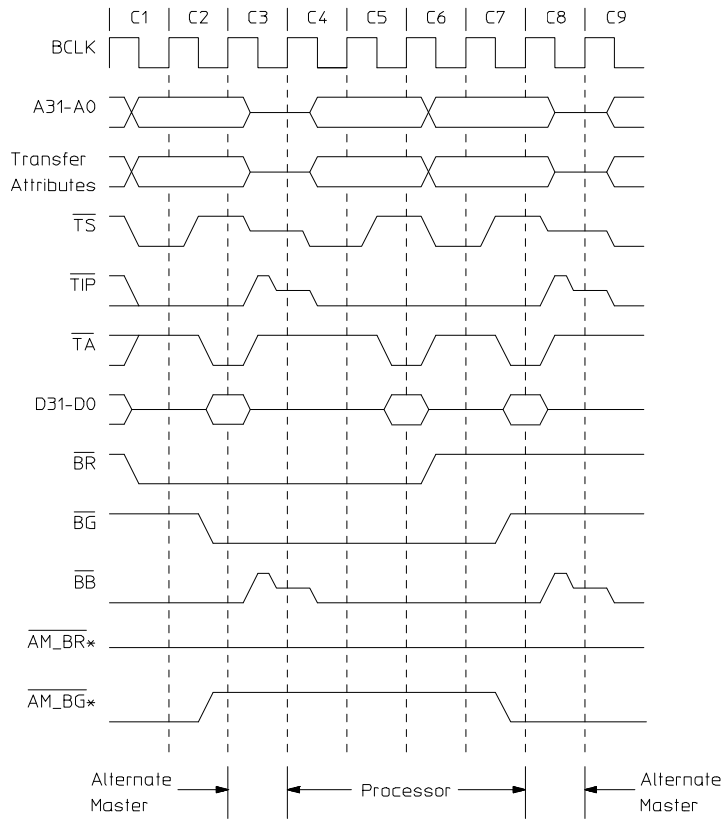


Note: The selected device increments the value of A3 and A2.

64783W03

Chapter 15: Connecting the Emulator to a Target System Verifying Operation Of The Emulator In Your Target System

If the prompt is "g>" and there are no cycles in the trace list, the target system never gave the bus to the processor. Check the bus arbitration signals for proper functionality and timing. Refer to the bus arbitration diagram below. Remember that the analyzer does not trace alternate bus master cycles while the emulator is reset, but it does once the emulator is running.



* AM indicates the alternate bus master.

64783w01

Chapter 15: Connecting the Emulator to a Target System

Verifying Operation Of The Emulator In Your Target System

When trying to determine why the bus is not being granted to the processor, you will need to determine why either the bus arbitration circuitry or an alternate bus master is not behaving correctly. The processor is not the bus master; therefore, it requests the bus with BR and waits for the target system to grant the bus with BG. The processor then waits for the BB line to be deasserted, indicating an idle bus, before taking control of the bus. The processor will not request the bus until after the reset line has been deasserted.

If the bus is requested by the processor, but it is not being granted check the bus arbitration signals BB, BG, and BR. If the bus is granted, but never becomes idle, the alternate bus master may be stuck in the middle of a cycle. Check the cycle strobes TS, TA, and TEA. These strobes do not have to be asserted during alternate master accesses, but if TS is shown to the processor, then TA needs to be shown to end the cycle. While the processor is reset, the only item of concern is signal quality.

If some cycles are shown in the trace list, but no cycles are occurring now, the processor executed some cycles before getting stuck in a DMA cycle. Examine the bus arbitration signals and cycle strobes around where the target system gets stuck. Use the same techniques to set up a trigger as were described for measuring a bus cycle that stops before it is complete.

If there are bus cycles occurring, then the "g>" prompt indicates that a high percentage of the bus activity is by alternate bus masters.

Interpreting the trace list

There are some cases where a problem caused by an errant bus cycle does not show up until many cycles later. The emulation-bus analyzer must be used to track back thru the sequence of events to the faulty bus cycle. Data problems will often behave like this, but there may be other causes.

If the "h>" prompt is shown, indicating a double bus fault, and if there are only two states in the tracelist, this indicates a problem with the fetching of the initial vectors.

```
h>t1
-----
Line   addr,H   68040 Mnemonic
-----
0      00000000 $00000000 sdata long read
1      00000004 $000BADAD  sdata long read
2
h>
```

The first two cycles in the trace list are the initial stack pointer and the initial program counter. The initial program counter must be even or the processor will immediately double bus fault. You should verify that the data captured by the analyzer is what is expected.

If the data for the vectors is wrong, a trace should be set up to check for access problems during the fetch of the initial vectors. If the data is completely incorrect, suspect an address or strobe timing problem. If only a few bits are wrong or if the data in the trace is correct, suspect a data timing problem.

If there are a lot of cycles in the tracelist, you need to start from the end and work backwards to understand what caused the double bus fault. If the trace was completed before the processor stopped, modify the trace specification to "trigger on nothing" so that the last bus cycles that were run can be captured. Wait until the emulator status shows a double bus fault, and then halt the trace.

tg never

reset the target system

es

th

t1 -20

Chapter 15: Connecting the Emulator to a Target System

Verifying Operation Of The Emulator In Your Target System

```

h>t1
-----
Line   addr,H   68040 Mnemonic
-----
-16    00000008 $4AFC0000   sprog long read   <- illegal inst
-15    0000000c $000BADAD   sprog long read
-14    00000010 $000BADAD   sprog long read
-13    00000014 $00000000   sprog long read
-12    00000018 $00000000   sprog long read
-11    000000ee $----0010   sdata word write  <- illegal inst stack
-10    000000ea $----0000   sdata word write
-9     000000ec $0008----   sdata word write
-8     00000010 $000BADAD   sdata long read   <- odd vector
-7     000000e8 $2700----   sdata word write
-6     000000e4 $000BADAC   sdata long write
-5     000000e2 $----200C   sdata word write  <- address error stack
-4     000000de $----0000   sdata word write
-3     000000e0 $0008----   sdata word write
-2     0000000c $000BADAD   sdata long read   <- odd vector
-1     000000dc $2700----   sdata word write
-----
h>

```

A double bus fault occurs when the processor encounters an exception that prevents processing of a previous exception. An example of a double bus fault is shown above. This original exception occurred because the target system tried to execute an illegal instruction. During processing of the illegal instruction exception, the processor encountered another exception.

This exception was an address error caused because the vector supplied for the illegal instruction handler was odd. The double bus fault occurred when the vector supplied for the address error handler was also odd. Other things that can cause a double bus fault are bus errors that occur during exception stacking or vector fetch. Keep in mind that bus errors can happen because the the target system asserts TEA or because of an access violation caused by the MMU.

Once you have found the cause of the double bus fault, you need to determine the root cause of the problem. In some cases, the exception is a normal part of execution, but the subsequent faults indicate a problem. In some cases, the first fault indicates a problem directly, such as when the program has already malfunctioned, and the fault is caused by an unintentional accesses.

At this point, the problem is to find the faulty bus cycle that eventually caused a recognizable problem. The same situation exists if the processor stops execution at an address that should not have been executed, or if a program is simply running code incorrectly.

Chapter 15: Connecting the Emulator to a Target System

Verifying Operation Of The Emulator In Your Target System

There are really only two ways to go about determining what is wrong. One is to try to trace back the terminal error condition to a faulty bus cycle. The other is to start at the beginning of the trace, or at some other known point, and work forward, comparing the trace to the execution that was expected while looking for the point where execution first becomes unexpected. A listing of the program or a tracelist captured by a preprocessor could be used for this comparison.

When you find a suspected bus cycle, set up a trigger on it so that you can make a timing measurement on the cycle. When looking for clues or shortcuts to the problem, keep in mind that a system is usually made up of many different types of memory devices: ROM, EEPROM, SRAM, DRAM, and peripheral ports. Each of these devices may have different timing characteristics. Also, keep in mind that unique characteristics of a bus cycle, such as size, transfer type, number of wait states, and bursting may result in unique timing requirements.

Fixing timing problems

When a timing problem is identified, you must decide how to fix it. First, examine the signal to make sure that signal quality is not affecting the timing. Look for AC or DC drive problems or reflections caused by transmission line problems. If you can find no other solution to the problem, you may have to lower the clock speed.

If the timing problem only occurs during data accesses, another possible solution is to add wait states to the memory access. This assumes that the problem is with the amount of time it takes to access the memories in the system and is not a problem with a setup time to a synchronous circuit. A good indicator of this type of problem is when the data setup time to the emulator is being missed. One point of caution: the emulator, when configured with wait states (**cf wait=en**), does not add a wait state to target accesses. The target system is responsible for adding the wait state.

Another possible solution to data access problems is to use faster memories while using the emulator.

Installing the emulator in a target system without known good software

If you do not have a program in ROM on your target system that you can run to electrically test the emulator, you will need to create a test environment. The initial step of this is to use the emulator's dual-port memory to install a simple program that will run from reset. To do this, proceed as follows:

- 1 Turn on emulator power.
- 2 Check the prompt by pressing <RETURN>. The prompt should be "p>". A "->" prompt indicates a software compatibility problem. Correct problems indicated in error messages or check the version "ver" for more information.
- 3 Configure the emulator by entering the following commands:

```
cf mon=none  
cf cache=en  
cf mmu=en  
cf ti=en  
cf wait=<en,dis>, as appropriate for the target system
```

- 4 Map dual-port memory with the following command:

```
map 0..0fff eram dp,lock
```

This maps a block of emulation memory starting at address 0 so that the reset vectors will be accessed from this block. The block is configured to be interlocked to the target system strobes because all systems must have some memory that responds at address 0 to operate.

- 5 Load a program with the following commands:

```
mo -ax -dl  
m 0=0f00,100  
mo -dw  
m 100=60fe
```

This sets up the reset vectors ISP=0f00 and IPC=100. It then loads the most simple program imaginable: jump to self.

Chapter 15: Connecting the Emulator to a Target System

Verifying Operation Of The Emulator In Your Target System

6 Setup a trace to capture all MC68040 cycles, as follows:

```
tck -u
tg any
tsto any
tp c
t
```

7 Execute **r rst**.

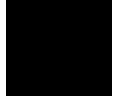
This tells the emulator to deassert reset so that the emulator does not interfere with the target system powerup reset.

8 Power on the target system.

9 Verify correct operation.

The target system should run the same as when the target processor was being used. The first indication of whether or not your target system is working is to see if your program performs any I/O that can verify correct system operation. If your target system appears to work initially, allow it to reach normal operating temperature before concluding that target system operation is as it should be.

If the target system appears to work properly, go ahead to the paragraph titled "Installing a Monitor". If you suspect problems, return to "Verifying System Operation" in the previous paragraphs. Keep in mind that the emulator must receive strobes from the target system for emulation memory accesses to complete. Also, because these cycles are from internal emulation memory, the data on the target system will not be the same as what the processor sees. If you think that there are problems with emulation memory data, check the clock speed configuration; the emulator is designed to give correct data at all speeds of operation.



Installing Emulator Features

Once the emulator is transparently running in the target system, it is time to start adding other emulator features. Dividing the installation of features into two tasks is the easiest way to debug problems. The monitor is the facility that provides the majority of the emulator's features, but some features like the reset circuitry do not require the monitor. The first feature to be installed does not depend on the monitor.

Evaluating the reset facilities

Now is a good time to use the emulator to find out how the emulator reset interacts with your target system. The first question to answer is whether or not the emulator reset command is adequate to reset your target system. Perform the following steps:

- 1 Run your target program by following the procedure in the previous steps.
- 2 Reset the emulation processor and run your program using the emulator commands:

r rst

Note that the "r rst" command pulses the processor reset line.

- 3 Verify correct operation.

If your program does not run correctly after performing the above procedure, your target system has other circuitry besides the processor that must be reset. The emulator only resets the emulation processor when it responds to a reset command. Other circuitry on your target system does not get reset. The following sequence determines if an additional reset circuit is required.

- 1 Run your target program following the procedure in the previous steps.
- 2 Reset the emulation processor and run your target program using these emulator commands:

rst

Reset the target system using whatever facility is available.

r rst

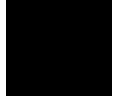
3 Verify correct operation of the target system.

An example of a target system that requires an additional reset circuit is one that normally has RAM starting at address 0, but for the first two bus cycles after reset, maps ROM to this area instead to provide the initial vectors. If this remapping does not occur, the system will attempt to fetch these vectors out of RAM, which will fail.

For systems that require additional circuitry to be initialized by reset, a reset output from the emulation probe (called reset flying lead) is provided. This reset flying lead can be connected into your target circuitry to eliminate the need for an additional step to reset circuitry in your target system. This allows the whole reset procedure to be controlled by the emulator, automatically.

One additional thing to keep in mind is that your target system can initiate a reset without the knowledge of the emulator. A reset that is initiated by your target system will reset the emulator. If the emulator was running your target program at the time of the reset, then when your system releases reset, the emulator will run as if an **r rst** command had been issued. If the emulator was executing in the monitor at the time of the reset, it will return to the monitor when the reset is released.

Another resetting method that may provide more convenience than the first method requires use of the monitor. This method works well for target systems such as those in the example above. This method resets the emulator into the monitor instead of running the target system program immediately. Once in the monitor, the initial stack pointer and initial PC can be loaded into the appropriate registers, and then a run of the target program can be initiated. This method will be illustrated in the next section.



Installing the background monitor

The emulator allows you to choose between use of a background and foreground monitor, but the choice is really predetermined by which of the MC68040 features you will be using.

The background monitor does not support use of the MMU, the caches, or DMA. Therefore, the background monitor is only useful in the most simple systems, or to provide a mechanism for testing target hardware, or to further evaluate the integration of the emulator with your target system.

The background monitor does not show cycles to your target system. It accomplishes this by blocking the TS and TIP signals. Therefore, the background monitor is transparent to your target system. Even though the background monitor does not show its cycles to the target system, the initial vector fetch cycles are shown to the target system and interlocked with the target system strobes. Cycles not shown to the target system are called background cycles. All other cycles are called foreground cycles.

Resetting into the background monitor

There are three ways to initially get into the background monitor. The first of these ways is to enter the monitor from reset. Perform the following command sequence to enter the monitor:

- 1 Reset the emulator and the target system if necessary using any reset procedure you determined to work adequately.
- 2 Configure the emulator by entering the following commands:

```
cf mon=bg  
cf monkaa=none  
cf cache=dis  
cf mmu=dis  
cf ti=en  
cf wait=<en,dis>, as appropriate for the target system
```


Chapter 15: Connecting the Emulator to a Target System

Installing Emulator Features

- 3 Set up a trace to capture all MC68040 cycles, including background monitor cycles, by entering the following commands:

```
tck -ub
tsto any
tg any
t
```

- 4 Execute the command: **rst -m**. This tells the emulator to release reset, but enter the monitor.
- 5 Verify that the emulator is in the monitor.

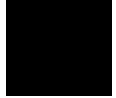
The prompt should be "M>", indicating that operation is in the monitor. There is not much that can go wrong up to this point because everything required has been previously verified.

If you see the following error messages, something went wrong during the initial vector fetches from the target system. Check these cycles for problems.

```
!STATUS 170! Emulator terminated hung bus cycle: 000000000@sd long read
!STATUS 170! Emulator terminated hung bus cycle: 000000004@sd long read
```

If you see a "g>" prompt, the background monitor is not compatible with this type of target system. Go to the paragraph titled "Installing the Foreground Monitor".

If you get the "?>" prompt or something other than the "M>" prompt, this indicates something went wrong with monitor operation. This may indicate problems with the clock or reset signals. Because the emulator provides all control signals for the background monitor, typically problems are with signals that can prevent the processor from running bus cycles.



Dealing with keep-alive circuitry while using the background monitor

Another thing to watch for when using the background monitor is the triggering of a target system keep-alive circuit because monitor bus cycles are hidden. Depending on how a keep-alive circuit operates, the monitor may cause a problem. The symptoms for different keep alive circuits may not show up in the same way.

Keep-alive circuits that monitor accesses on the bus or require a certain address to be accessed probably will fail when you use the background monitor. Keep-alive circuits that make sure bus cycles complete will not fail. If the keep-alive circuit generates a bus error or an interrupt, the monitor will not be affected immediately. If the keep-alive circuit asserts reset instead, monitor operation will be affected immediately, although there may be no apparent symptoms if reset is only asserted temporarily because the monitor will be reentered as soon as reset is deasserted.

If you suspect a problem with a keep-alive circuit, there is a configuration option that can make the background monitor periodically cause a read access to a particular address. If you do need a particular address to be read for the keep-alive function, make sure the address you give will respond with memory strobes when accessed.

cf monkaa=0deadad0

Retry the reset into monitor with this configuration enabled. If there is any sort of problem with the keep-alive access, it will probably show up as a wait state at the keep-alive address. If this happens, check the timing on that particular cycle. The keep-alive address may respond with a bus error without adversely affecting monitor operation.

Testing memory accesses with the background monitor

Once the background monitor looks like it is running properly, you can use it to test accesses to different ranges of memory in your target system. This may be an easier way to diagnose problems than by running a program that accesses each memory range. It is also easy to check accesses of different sizes using the monitor.

```
mo -ax -dl  
m 0badad=12345678
```

When accesses to your target memory do not execute exactly right, the monitor attempts to diagnose these problems and resolve them so the monitor program does not malfunction. However, the monitor does not read back write cycles to check the integrity of the data written. When testing memory accesses, the data should be checked to make sure that it is correct.

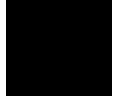
```
M>m 0badad  
0000badad ffd00ff
```

If your target memory does not respond to a bus cycle, the monitor will force termination of the cycle and report this error message:

```
!STATUS 170! Emulator terminated hung bus cycle: 0000badad@sd word read  
!ERROR 700! Target memory access failed
```

Or, if the target system responds with a bus error for this memory access, the monitor will report that information:

```
!ERROR 170! Target bus error: 0000badad@sd  
!ERROR 700! Target memory access failed
```



Running a program from the background monitor

Once you are satisfied that the monitor is working and that memory in your target system can be accessed correctly, you can use the monitor to run your target program. Proceed as follows:

- 1 Reset into the monitor.
- 2 Load a program, if necessary.
- 3 Initialize the initial stack pointer and initial program counter.

```
reg isp=<initial ISP>  
reg pc=<target program starting address>
```

If these values are not known, they can be found by taking a trace of the program running from reset, as was done in the previous sections.

- 4 Take a trace of the program running, using the following commands:

```
tg addr=<long aligned target program starting address>  
t
```

The trigger address must be long aligned because the MC68040 always fetches instructions as long words from long-word boundaries.

- 5 Run the program with the command:

```
r
```

- 6 Verify correct operation of the program.

Assuming that the program ran without the monitor, the stack is most likely the cause of any problems you see. The monitor runs the program by creating a stack in foreground memory at the location indicated by the initial stack pointer. The monitor then initiates an RTE, which starts the target program running. The following trace list is an example showing correct operation:

Chapter 15: Connecting the Emulator to a Target System Installing Emulator Features

Line	addr, H	68040 Mnemonic	
-4	000000f0	\$00----- mon sdata byte read	
-3	000009b4	\$4E714E71 mon sprog long read	
-2	000000ec	\$000a007C sdata long read	<-unstack
-1	000000e8	\$27000000 sdata long read	<-unstack
0	00000008	\$000060FE sprog long read	<-target program
1	0000000c	\$000BADAD sprog long read	

If the monitor detects problems with the stack pointer (the stack pointer must be even), or if the monitor has a problem accessing the stack memory, an error message is issued. Additionally, the monitor checks to make sure that the stack has been written correctly before exiting. Problems are indicated by the error messages listed below.

From this point on, most of the problems will be discussed from a functional point of view instead of a parametric point of view. If any of the functional problems discussed below identify a problem that looks parametric, use the debugging techniques of the previous procedures to isolate the problem.

```
!ERROR 151! Interrupt stack pointer is odd or uninitialized
!ERROR 610! Unable to run
```

This message indicates that the stack pointer is invalid. Only word-aligned stack pointers are allowed with the emulator. If this error is seen, the run will not be attempted.

```
!ERROR 170! Target bus error: 0000000e8@sd
!ERROR 610! Unable to run
```

This message indicates a bus error occurred during the stack write. This behavior could be caused by putting the stack in a memory range that responded with bus error for all accesses, or bus error on write accesses. Or, it could be caused by putting the stack where nothing responds, and the bus error is the result of a timeout. Keep in mind that the stack grows down from the initial stack pointer.

```
!STATUS 170! Emulator terminated hung bus cycle: 0000000e8@sd long write
!ERROR 610! Unable to run
```

This message indicates that the stack is in an address range that did not respond with a memory strobe. Make sure that the stack is placed in valid memory.

```
!ERROR 151! Interrupt stack is not located in RAM: 0000000e8@sd
!ERROR 610! Unable to run
```

Chapter 15: Connecting the Emulator to a Target System

Installing Emulator Features

This message indicates that the stack memory was not writeable. Check to make sure that the stack is placed in RAM.

If the target program appears to start at the wrong address, or if there is some other problem, the stack can be decoded to see if the correct information is present there. The stack above is interpreted as follows: The initial stack pointer is defined to point to the next available stack location. Therefore the exit stack starts four words below the initial stack pointer.

```
ISP-8 -> Status register = 2700
ISP-6 -> Program Counter = 0000000a
ISP-2 -> Vector Offset = 007C
```

The monitor is always exited using the FOUR WORD STACK frame, and the monitor always uses 07C as the vector offset. When running a program from the monitor after entering from reset, the powerup status word of 2700 is used. Therefore, the only difference you will see in this stack frame will be because of different initial program counter values.

The procedure of setting the initial stack pointer and initial program counter can be automated by using the initial vectors configuration question to define these values.

cf rv=<initial ISP>,<initial PC>

Once this configuration has been set up, the following reset sequence may be useful on systems that remap memory to provide reset vectors similar to the example in the "Evaluating the Reset Facility" section.

rst -m
r

Breaking into the background monitor

The next thing to try with the background monitor is to see if you can break into it from your target program. The emulator uses a nonmaskable interrupt (interrupt 7) to break into the monitor. The interrupt is generated in such a way as to not interfere with any interrupts pending in your target system. The resulting interrupt acknowledge cycle is not shown to the target system. The associated stacking is in foreground memory at the location determined by the interrupt stack pointer. If the target system program is running in Master mode, there will also be stacking on the master stack.

A vector fetch occurs sometime during or after stacking; it is also shown to the target system. The emulator provides the data for this vector fetch to correctly run the background monitor. After stacking and the vector fetch are completed, the emulator transitions into the background monitor. The background monitor may access foreground memory during its operation.

While the emulator is in the background monitor, no target interrupts are serviced. The interrupt signals from the target system are ignored while in the background monitor. The emulator will not respond to these signals in any way while in the monitor. If the signals are still present when the monitor is exited, they will be serviced according to normal interrupt priorities.

Entry into the background monitor can be traced by using the following trigger specification:

```
tck -ub
tp c
tg stat=11xxxxxxx1x111xy
t
b
```

Line	addr,H	68040 Mnemonic	
-2	00000008	\$60FE0000	sprog long read
-1	0000000c	\$000BADAD	sprog long read
0	fffffff	\$-----FF mon	int7 ack <-acknowledge
1	000000ee	\$----007C	sdata word write <-stack format
2	000000ea	\$----0000	sdata word write <-stack PC high
3	000000ec	\$0008----	sdata word write <-stack PC low
4	0000007c	\$0000069C	sdata long read <-vector fetch
5	000000e8	\$2700----	sdata word write <-stack SR
6	00000698	\$0012FFFF	mon sprog long read <-monitor
7	0000069c	\$11FC001F	mon sprog long read

Chapter 15: Connecting the Emulator to a Target System

Installing Emulator Features

If you have problems trying to break into the monitor, the most likely causes are the values of the stack pointers, or the vector base register does not point to valid memory. Any bus errors that occur during monitor entry will cause the break to fail. If any stacking or vector fetch cycles are not terminated, the monitor will terminate them by force. If this happens, the PC and SR may be displayed incorrectly by the monitor. The same problem can result from stack memory that is not writable. Neither condition will inhibit entry into the monitor, but the target state will be corrupted.

Exiting the background monitor

If the procedures described in the preceding paragraphs gave satisfactory results, you should be able to resume execution of the target program. You may want to take a trace of the monitor exit procedure to verify that it is completed correctly.

r

If the target system and emulator do not work correctly after exiting the background monitor, the problem may be because your target system is real-time sensitive. If interrupts that needed to be serviced to keep the target system running were delayed by the monitor, things such as data overrun could cause problems in the target system. If you suspect such a problem, use the foreground monitor.

Software breakpoint entry into the background monitor

The background monitor can also be entered via a software breakpoint. The emulator will respond to any software breakpoint instruction in the code if breakpoints are enabled, regardless of whether the breakpoint was inserted by the emulator or not. Breakpoints are enabled by the following command.

bc -e bp

Set breakpoints only on the initial word of an instruction; otherwise, they will not be executed, and might alter an instruction, unintentionally. The emulator can place a breakpoint using one of two methods. By default, the emulator will attempt to modify memory to insert a breakpoint instruction at the address specified. If the memory at the address specified is ROM or cannot be modified for some other reason, special hardware resources on the emulator will interject a breakpoint instruction when that address is fetched.

b

bp <instruction address>

If you suspect a problem occurred during the setting of the breakpoint, you can use the analyzer to watch the breakpoint being set. The easiest way to do this is to store-qualify your trace on the address where you are setting the breakpoint. The trace list will only contain a cycle or two, but you can see what happened when the emulator accessed this address.

tg any

tsto addr=<instruction address>

b

bp <instruction address>

Line	addr,H	68040 Mnemonic	
0	00000008	\$FFFF----	sdata word read
1	00000008	\$FFFF----	sdata word read
2	00000008	\$FFFF----	sdata word read
3	00000008	\$484F----	sdata word write <- breakpoint write
4	00000008	\$FFFF----	sdata word read <- verify
5	00000008	\$FFFF----	sdata word read
6			

Chapter 15: Connecting the Emulator to a Target System

Installing Emulator Features

When a software breakpoint instruction is executed, the processor initiates a breakpoint-acknowledge cycle. This cycle signals the start of an entry into the monitor. From this point on, stacking and the vector fetch proceed the same as for a break entry. Unlike the interrupt-acknowledge cycle, the breakpoint-acknowledge cycle is shown to the target system.

```
tsto any
tg stat=11xxxxxxx1x000xy
t
r 8
```

Line	addr,H	68040 Mnemonic	
-4	00000008	\$484F0000	sprog long read <-bkpt fetch
-3	0000000c	\$000BADAD	sprog long read
-2	00000010	\$000BADAD	sprog long read
-1	00000014	\$00000000	sprog long read
0	00000000	\$41-----	bkpt ack (buserror) <-acknowledge
1	000000ee	\$----0010	sdata word write <-stack format
2	000000ea	\$----0000	sdata word write <-stack PC high
3	000000ec	\$0008---	sdata word write <-stack PC low
4	00000010	\$00000690	sdata long read <-vector fetch
5	000000e8	\$2700----	sdata word write <-stack SR
6	00000690	\$11FC0004 mon	sprog long read <-monitor
7	00000694	\$01186000 mon	sprog long read

The only unique portion of a breakpoint entry is the breakpoint-acknowledge cycle so any problems that you see will probably be related to this cycle. Because the emulator internally responds to this cycle, it is not necessary for the target system to respond to it. If the target system does respond to this cycle with any wait states, the emulator may become out of sync with the target system because the emulator terminates this cycle immediately. If this were to cause a problem, it would show up on the cycle immediately following the breakpoint-acknowledge cycle.

Stepping with the background monitor

The last feature of the background monitor which needs to be evaluated is the single-stepping facility. The emulator uses a combination of the processor trace facility and a nonmaskable interrupt to reenter the monitor after executing exactly one instruction.

```

b
tsto any
tg stat=11xxxxxxx1x111xy
t
s

```

```

000000008@s - BRA.B $00000008
PC = 000000008@s

```

When a step command is issued, the emulator sets the trace bits in the SR and then performs a normal monitor exit. The emulator then forces a break to return to the monitor. A typical trace of a single step is shown below:

Line	addr,H	68040 Mnemonic	
-17	000009b0	\$4E714E71 mon	sprog long read
-16	000000f0	\$00----- mon	sdata byte read
-15	000009b4	\$4E714E71 mon	sprog long read
-14	000000ec	\$0008007C	sdata long read <- unstack
-13	000000e8	\$A7000000	sdata long read <- unstack
-12	00000008	\$60FE0000	sprog long read <- stepped inst
-11	0000000c	\$000BADAD	sprog long read
-10	00000008	\$60FE0000	sprog long read
-9	0000000c	\$000BADAD	sprog long read
-8	000000ec	\$00000008	sdata long write <- trace stack addr
-7	000000ea	\$----2024	sdata word write <- trace stack format
-6	000000e6	\$----0000	sdata word write <- trace stack PC up
-5	000000e8	\$0008----	sdata word write <- trace stack PC low
-4	00000024	\$00000000	sdata long read <- trace vector fetch
-3	000000e4	\$A700----	sdata word write <- trace stack SR
-2	00000000	\$000000F0	sprog long read <- trace prefetch
-1	00000004	\$00000008	sprog long read <- trace prefetch
0	fffffff	\$-----FF mon	int7 ack <- break acknowledge
1	000000e2	\$----007C	sdata word write <- break stack format
2	000000de	\$----0000	sdata word write <- break stack PC up
3	000000e0	\$0000----	sdata word write <- break stack PC low
4	0000007c	\$0000069C	sdata long read <- break vector fetch
5	000000dc	\$2700----	sdata word write <- break stack SR
6	00000698	\$0012FFFF mon	sprog long read <- monitor
7	0000069c	\$11FC001F mon	sprog long read

At the end of the execution of the first target program instruction, the processor takes a trace exception. Stacking for this trace exception commences and at some point, the trace vector is fetched. Once stacking for the trace is complete, the processor prefetches from the address of the trace handler, but these instructions are

Chapter 15: Connecting the Emulator to a Target System

Installing Emulator Features

never executed because the processor immediately starts interrupt processing. The interrupt processing proceeds the same as in a normal break.

Before exiting for a step, the monitor checks to make sure that the trace vector is valid and that it points to accessible memory. If the vector is not even, or if the memory it points to responds with a bus error or hangs, the emulator temporarily modifies the trace vector to point to the start of the vector table. Because the instructions of the trace handler will not be executed, the content of the address locations is not important.

If the emulator modifies the trace vector, the following status message is given:

```
!STATUS 155! Vector table modified for single stepping
```

If the emulator finds it must modify the trace vector for single stepping to complete, but the modification attempt fails, an error message similar to the following is displayed:

```
!ERROR 170! Target bus error: 0ff800024@sd  
!ERROR 156! Unable to modify trace vector to ff800000h for single stepping  
!ERROR 680! Stepping failed
```

If this error occurs, the vector table must be modified so that the trace vector contains an address that points to accessible memory. If the vectors are in ROM, perhaps the memory can be copied into emulation memory where you can modify it.

One way to watch what the emulator is doing during a step, is to set up the analyzer to trace only foreground cycles and to store everything. This lets you watch the emulator check and possibly modify the trace exception vector. Use the following commands:

```
tck -u  
tsto any  
tg any  
t  
s
```

The emulator may experience problems when stepping over instructions that modify the VBR. This is because the check of the trace exception vector is made using the old VBR value, but the actual stepping will use the new value of the VBR. If the new VBR value changes the trace exception vector to something that would require modification, then stepping can fail.

```
!ERROR 680! Stepping failed
```

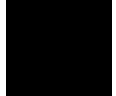
Chapter 15: Connecting the Emulator to a Target System

Installing Emulator Features

When stepping over instructions that cause the processor to take exceptions, the trace list can look very different. Most exceptions preempt the trace exception until after their exception handler runs. Other exceptions (like TRAP, CHK and CHK2) create their stack frame and then take the trace exception. Any exceptions cause the step trace list to look different. In all cases, the monitor is still entered through the interrupt 7 exception.

For all exceptions except TRAP, CHK, and CHK2, the trace stack frame will be missing when the monitor is entered. Instead of using the trace stack frame, the exception stack frame will be used. The emulator detects that and issues an error message that says stepping failed. This error message does not actually indicate a problem with emulator stepping; it just indicates that an exception was hit. The emulator is stopped at the starting address of the exception handler, and stepping can be resumed.

The TRAP, CHK, and CHK2 exceptions will have an additional stack frame when the monitor is entered. The exception stack frame will precede the normal trace and interrupt stack frames. These exceptions do not cause the monitor to issue an error message so multiple steps will not stop on this type of exception.



Installing the foreground monitor

The foreground monitor supports all features of the emulator, but imposes on your target system more than the background monitor. The foreground monitor occupies a 4-Kbyte block in your target memory space. The emulator provides memory for this 4-Kbyte block, but the target system cannot use this address range for anything. The cycles strobes TS and TIP are shown to the target system during foreground monitor cycles. The monitor needs to be placed in an address range where it will not interfere with target system operation.

If the monitor is placed in an address range where the target system responds with a TA, interlock the monitor to the target strobes. The target system must not respond with TEA for this address range. If the monitor is placed in an address range where the target system does not respond with any strobes, do not interlock the monitor. If in doubt, interlock the foreground monitor to the target system. It will be obvious if this is the wrong thing to do because the monitor will stop operating immediately.

If the MMU is being used, the monitor must be placed in an address range that is translated logical=physical, and is writeable for supervisor program and data. If the memory management scheme is dynamic, the monitor page must be resident at all times. In addition, any pages required for stacking or vector fetches must also be resident.

If there is not a suitable address range in which to put the monitor, the system protection schemes may need to be modified to create a place for the monitor. This may be as simple as adding an entry to the MMU tables, or it may require modifying a hardware protection scheme to allow placement of the monitor.

Besides adding special requirements to the placement of the monitor, the MMU impacts many operations of the emulator and processor. When the MMU is on, the emulator can access both physical and logical memory. The emulator also provides commands to examine the MMU tables.

With the MMU on, there are new problems added to the task of connecting the emulator probe into a target system. Besides making sure that the restrictions noted above are complied with, interpreting the trace list becomes more difficult. You also need to keep in mind the distinctions between logical and physical memory accesses when accessing memory. Finally, you need to find out whether you need to load your program before the MMU is running or while it is running.

The foreground monitor, in contrast to the background monitor, allows servicing of interrupts. When the foreground monitor is not busy performing some action, interrupts are allowed. The interrupt routine must return control to the monitor within a reasonable period of time or the monitor may timeout if it attempts to do something. The level of interrupt that can be recognized by the monitor can be controlled through a configuration question:

cf monint=0

Resetting into the foreground monitor

If you have successfully established operation of the background monitor, or if you have decided that you cannot use the background monitor because you need certain MC68040 features, then it is time to evaluate the foreground monitor. The first thing to do is to enter the foreground monitor from reset. Perform the following command sequence to enter the monitor.

- 1 Reset the emulator, and the target system if necessary, using whatever reset procedure you determined to work.
- 2 Configure the emulator, as follows:

```
cf mon=fg  
cf monaddr=addr as appropriate for the target system  
cf monlock=<en,dis> as appropriate for the address mapping  
cf monint=0  
cf cache=en  
cf mmu=en  
cf ti=en  
cf wait=<en,dis> as appropriate for the target system
```

- 3 Set up a trace to capture all MC68040 cycles. Background cycles do not need to be traced to see foreground monitor operation.

```
tg any  
tsto any  
tck -u  
t
```

Chapter 15: Connecting the Emulator to a Target System

Installing Emulator Features

- 4 Execute the command: **rst -m**

This tells the emulator to release reset, but enter the monitor.

- 5 Verify that the emulator is in the monitor.

The prompt should be "M>", indicating correct operation in the monitor.

There is not much that can go wrong up to this point since everything required has been previously verified.

If you get the following error messages, a failure occurred during the initial vector fetches from the target system. Check these cycles for problems.

```
!STATUS 170! Emulator terminated hung bus cycle: 000000000@sd long read
!STATUS 170! Emulator terminated hung bus cycle: 000000004@sd long read
```

If you get a "w>" prompt for a monitor address, you may have incorrectly interlocked the monitor to the target system. If the monitor was correctly interlocked, check to see if there is a timing problem with the target terminations for the monitor address range.

If you get the "b>" prompt or something other than the "M>" prompt, suspect a failure in monitor operation. These prompts may indicate problems with the clock or reset signals. If the monitor is interlocked, it may also indicate that the target system responded with a bus error for a monitor access.

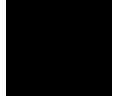
Dealing with keep-alive circuitry by using the custom foreground monitor

As with the background monitor, you may have problems with keep-alive circuitry located in the target system. Because the foreground monitor cycles are shown to the target system, bus cycle activity monitors should not be a problem. Also, because interrupts can be serviced within a reasonable period of time, any keep-alive circuits that depend on interrupts should not be a problem.

Keep-alive circuits that require a certain address to be accessed probably will fail when you are using the foreground monitor. The keep-alive problem will most likely show up immediately when using the foreground monitor. If the monitor is interlocked, it will be affected immediately if a keep-alive circuit causes a bus error. If a keep-alive circuit generates an interrupt or a reset, it should also be immediately obvious. If reset is only temporarily asserted, it may not be so obvious because the emulator will return to the monitor when it is released.

If you suspect a problem with a keep-alive circuit, try using the custom foreground monitor. This monitor can be customized to take the required actions to satisfy a keep-alive circuit. See the chapter on configuring the emulator for information on using the custom foreground monitor. Retry your reset into the monitor with the customized foreground monitor.

If keep-alive circuits cannot be accommodated by using the available emulator features, you may need to disable them for emulation.



Testing memory access with the foreground monitor

Once the foreground monitor looks like it is running properly, you can use it to test accesses to different ranges of memory in your target system. This may be an easier way to diagnose problems than by running a program that accesses each memory range. It is also easy to check accesses of different sizes using the monitor.

mo -ax -dl
m 0badad=12345678

When accesses to your target memory are not performed exactly right, the monitor attempts to diagnose these problems and resolve them so the monitor program does not malfunction. However, the monitor does not read back write cycles to check the integrity of the data written. When testing memory accesses, check the data to make sure it is correct.

```
M>m 0badad
 0000badad ffd00ff
```

If your target memory does not respond to a bus cycle, the monitor will force termination of the cycle and report this error message.

```
!STATUS 170! Emulator terminated hung bus cycle: 0000badad@sd word read
!ERROR 700! Target memory access failed
```

Or, if the target system responds with a bus error for this memory access, the monitor will report that information.

```
!ERROR 170! Target bus error: 0000badad@sd
!ERROR 700! Target memory access failed
```

Running a program from the foreground monitor

Once you are satisfied that the monitor is working and that memory in your target system can be accessed correctly, you can use the monitor to run your target program. Use the following procedure:

- 1 Reset into the monitor.
- 2 Load a program, if necessary.
- 3 Initialize the initial stack pointer and initial program counter.

```
reg isp=<initial ISP>  
reg pc=<starting address of target program>
```

If you do not know these values, you can find them by taking a trace of the program running from reset as done in the previous sections.

- 4 Take a trace of the program as it is running, using the following commands:

```
tg addr=<long aligned starting address of target program>  
t
```

The trigger address must be long aligned because the MC68040 always fetches instructions as long words from long-word boundaries.

- 5 Run the program with the command:

```
r
```

- 6 Verify correct operation of the program.

Assuming that the program ran without the monitor, the stack is most likely the cause of any problems that you see. The monitor runs the program by creating a stack in memory at the location indicated by the initial stack pointer. The monitor then initiates an RTE, which starts the target program running. The following trace list shows an example of correct operation:

Chapter 15: Connecting the Emulator to a Target System

Installing Emulator Features

Line	addr,H	68040 Mnemonic	
-4	000010f0	\$00-----	log sdata byte read
-3	00001e74	\$4E714E71	log sprog long read
-2	0000f0ec	\$000a007C	log sdata long read <-unstack
-1	0000f0e8	\$27000000	log sdata long read <-unstack
0	00000008	\$000060FE	log sprog long read <-target program
1	0000000c	\$000BADAD	log sprog long read

If the monitor detects problems with the stack pointer (the stack pointer must be even), or if the monitor has a problem accessing the stack memory, an error message is issued. Additionally, the monitor checks to make sure that the stack has been written correctly before exiting. Problems are indicated by the following error messages:

```
!ERROR 151! Interrupt stack pointer is odd or uninitialized
!ERROR 610! Unable to run
```

This message indicates that the stack pointer is invalid. Only word aligned stack pointers are allowed with the emulator. The run is not attempted.

```
!ERROR 170! Target bus error: 00000f0e8@sd
!ERROR 610! Unable to run
```

This message indicates a bus error occurred during the stack write. This behavior can be caused if the stack is in a memory range that responds with bus error for all accesses or for write accesses. Or, this behavior can be caused by putting the stack where the target system fails to respond immediately; the bus error is the result of a timeout. Keep in mind that the stack grows down from the initial stack pointer.

```
!STATUS 170! Emulator terminated hung bus cycle: 00000f0e8@sd long write
!ERROR 610! Unable to run
```

This indicates that the stack is in an address range that did not respond with a memory strobe. Make sure that the stack is placed in valid memory.

```
!ERROR 151! Interrupt stack is not located in RAM: 00000f0e8@sd
!ERROR 610! Unable to run
```

This indicates that the stack memory was not writeable. Check to make sure that the stack is placed in RAM.

If the target program appears to start at the wrong address, or if there is some other problem, the stack can be decoded to see if the correct information is present. The stack above is interpreted as follows: The initial stack pointer is defined to point to

Chapter 15: Connecting the Emulator to a Target System

Installing Emulator Features

the next available stack location. Therefore, the exit stack starts four words below the initial stack pointer.

```
ISP-8 - Status register = 2700
ISP-6 - Program Counter = 0000000a
ISP-2 - Vector Offset = 007C
```

The monitor is always exited using the FOUR WORD STACK frame, and the monitor always uses 07C as the vector offset. When running a program from the monitor after entering from reset, the powerup status word of 2700 is used. Therefore, the only difference you will see in this stack frame will be because of different initial program counters.

The procedure for setting the initial stack pointer and initial program counter can be automated by using the initial vectors configuration question to define these values.

cf rv=<initial ISP>,<target program starting address>

Once this configuration has been set up, the following reset sequence may be useful on systems that remap memory to provide reset vectors.

```
rst -m
r
```

Breaking into the foreground monitor

The next thing to try with the foreground monitor is to see if you can break into it from your target program. The emulator uses a nonmaskable interrupt (interrupt 7) to break into the monitor. The interrupt is generated in such a way as to not interfere with any interrupts pending in your target system. The resulting interrupt acknowledge cycle is not shown to the target system. The associated stacking is in foreground memory at the location determined by the interrupt stack pointer. If the target system program is running in Master mode, there will also be stacking on the master stack.

A vector fetch occurs sometime during or after stacking. The emulator provides the data for this vector fetch to correctly run the foreground monitor. While the emulator is transitioning into the foreground monitor, interrupts are temporarily blocked. Once in the monitor the interrupt mask level is lowered to the greater of the "monint" configuration setting or the target program mask level.

Chapter 15: Connecting the Emulator to a Target System

Installing Emulator Features

Entry into the foreground monitor can be traced by using the following trigger specification. The interrupt acknowledge signal is not shown to the target system and is also not shown to the analyzer unless background cycles are being traced.

```
tck -ub
tp c
tg stat=11xxxxxxx1x111xy
t
b
```

Line	addr,H	68040 Mnemonic	
-2	00000008	\$60FE0000 phy sprog long read	
-1	0000000c	\$00000000 phy sprog long read	
0	fffffff	\$-----FF mon int7 ack	<- acknowledge
1	00000200	\$0000040B mmu twalk data long read	<- twalk stack
2	00000400	\$0000060B mmu twalk data long read	
3	0000063c	\$0000F01B mmu twalk data long read	
4	0000f0ee	\$----007C phy sdata word write	<- stack format
5	0000f0ea	\$----0000 phy sdata word write	<- stack PC high
6	0000f0ec	\$0008---- phy sdata word write	<-stack PC low
7	00000200	\$0000040B mmu twalk data long read	<- twalk vector
8	00000400	\$0000060B mmu twalk data long read	
9	00000600	\$0000009F mmu twalk data long read	
10	0000007c	\$000016C2 phy sdata long read	<- vector fetch
11	0000f0e8	\$2700---- phy sdata word write	<- stack SR
12	00000200	\$0000040B twalk prog long read	<- twalk monitor
13	00000400	\$0000060B twalk prog long read	
14	00000600	\$0000101b twalk prog long read	
15	000016c0	\$4E732F0D phy sprog long read	<- monitor
16	000016c4	\$4BFAFB10 phy sprog long read	

If you have problems trying to break into the monitor, the most likely causes are that the stack pointers or vector base register do not point to valid memory. Any exceptions during monitor entry will cause the break to fail. Access errors during stacking or vector fetches are the most common causes of failures. The target system can respond with a bus error, or if the MMU is running, the MMU can signal an access error. The MMU will signal an error if a translation is not available, if a bus error occurs during translation lookup, or if a write protection error occurs.

The break will also fail if accesses to the monitor cause an exception. This includes bus errors and access errors signaled by the MMU. It is possible for the monitor to execute correctly until the MMU is enabled, and then have problems. Keep in mind that the monitor must be translated logical=physical and located in address space that is not write protected.

If any stacking or vector-fetch cycles are not terminated, the monitor will terminate them by force. If this happens, the PC and SR may be displayed incorrectly by the monitor. The same problem can result from stack memory that is not writeable. Neither condition will prevent entry into the monitor, but you will not be able to resume execution in the target program.

Exiting the foreground monitor

If the tests of the preceding paragraphs operate correctly, you should be able to resume execution of the target program. You may want to take a trace of the monitor exit to verify that everything is working correctly. Use the run command:

r

Software breakpoint entry into the foreground monitor

The foreground monitor can also be entered via a software breakpoint. The emulator will respond to any software breakpoint instruction in the code if breakpoints are enabled, regardless of whether the breakpoint was inserted by the emulator or not. Breakpoints are enabled by the following command:

bc -e bp

Only set breakpoints on the initial word of an instruction; otherwise, they will not be executed, and they may alter an instruction, unintentionally. The emulator can place a breakpoint using two methods. By default, the emulator will attempt to modify memory to insert a breakpoint instruction at the address specified. If the memory at the address specified is ROM or cannot be modified for some other reason, special hardware resources on the emulator will interject a breakpoint instruction when the associated address is fetched. You can tell if a hardware resource was required to support a breakpoint by viewing memory at the breakpoint address. If the BKPT instruction has replaced the normal instruction at that address, a software breakpoint was used. If the normal instruction is still in the

Chapter 15: Connecting the Emulator to a Target System

Installing Emulator Features

breakpoint address, the emulator is using one of its eight hardware resources to implement the breakpoint.

b
bp <program instruction>

If you suspect some kind of problem with the setting of the breakpoint, use the analyzer to watch the setting of the breakpoint. The easiest way to do this is to store-qualify the trace on the address where you are setting the breakpoint. The trace list will only contain a cycle or two, but you can see what happened when the emulator accessed the breakpoint address.

If the MMU is running, you will need to store-qualify the actual physical address being accessed. The address given in the "bp" command must always be treated as a logical address. To find the corresponding physical address, use the MMU translation command. Also, keep in mind that the MMU may cause problems when setting the breakpoint.

mmu -t <logical breakpoint address>

tg any
tsto addr=<physical breakpoint address>

b
bp <logical breakpoint address>

```
Line   addr,H   68040 Mnemonic
-----
0      00000008  $FFFF---- phy sdata word read
1      00000008  $FFFF---- phy sdata word read
2      00000008  $FFFF---- phy sdata word read
3      00000008  $484F---- phy sdata word write  <- breakpoint write
4      00000008  $FFFF---- phy sdata word read  <- verify
5      00000008  $FFFF---- phy sdata word read
6
```


Chapter 15: Connecting the Emulator to a Target System Installing Emulator Features

When a software breakpoint instruction is executed, the processor initiates a breakpoint-acknowledge cycle. This cycle signals the start of an entry into the monitor. From this point on, stacking and the vector fetch proceed the same as for a break entry. Unlike the interrupt-acknowledge cycle, the breakpoint-acknowledge cycle is shown to the target system.

```
tck -u
tsto any
tg stat=11xxxxxxx1x000xy
t
r 8
```

Line	addr,H	68040 Mnemonic	
-4	00000008	\$484F0000	log sprog long read <- bkpt fetch
-3	0000000c	\$00000000	log sprog long read
-2	00000010	\$01000000	log sprog long read
-1	00000014	\$00000000	log sprog long read
0	00000000	\$41-----	bkpt ack (buserror) <- acknowledge
1	00000200	\$0000040B	mmu twalk data long read <- twalk stack
2	00000400	\$0000060B	mmu twalk data long read
3	0000063c	\$0000F01B	mmu twalk data long read
4	0000f0ee	\$----0010	phy sdata word write <- stack format
5	0000f0ea	\$----0000	phy sdata word write <- stack PC high
6	0000f0ec	\$0008----	phy sdata word write <- stack PC low
7	00000200	\$0000040B	mmu twalk data long read <- twalk vector
8	00000400	\$0000060B	mmu twalk data long read
9	00000600	\$0000009F	mmu twalk data long read
10	00000010	\$000016A2	phy sdata long read <- vector fetch
11	0000f0e8	\$2700---	phy sdata word write <- stack SR
12	00000200	\$0000040B	twalk prog long read <- twalk monitor
13	00000400	\$0000060B	twalk prog long read
14	00000600	\$0000101b	twalk prog long read
15	000016a0	\$007E2F0D	phy sprog long read <- monitor
16	000016a4	\$4BFAFA73	phy sprog long read

The only unique part of a breakpoint entry is the breakpoint-acknowledge cycle so any problems will probably be related to this cycle. Because the emulator internally responds to this cycle, it is not necessary for the target system to respond to it. If the target system responds to this cycle with any wait states, the emulator may become out of sync with the target system because the emulator terminates this cycle immediately. If this causes a problem, it will show up on the cycle immediately following the breakpoint-acknowledge cycle.

Stepping with the foreground monitor

The last feature of the foreground monitor that needs to be evaluated is the single-stepping facility. The emulator uses the processor trace facility to reenter the monitor after executing exactly one instruction, unless an exception occurs.

```

b
tsto any
tg stat=11xxxxxxx1x000xy
t
s
000000008@s -          BRA.B    $00000008
PC = 000000008@s

```

When a step command is issued, the emulator sets the trace bits in the SR, and then performs a normal monitor exit. The emulator modifies the trace vector to transfer control to the monitor. A typical trace of a single step is shown below:

Line	addr,H	68040 Mnemonic			
-42	000010f0	\$00-----	log	sdata	byte read
-41	00001e74	\$4E714E71	log	sprog	long read
-40	00000200	\$0000040B	mmu	twalk	data long read
-39	00000400	\$0000060B	mmu	twalk	data long read
-38	0000063c	\$0000F01B	mmu	twalk	data long read
-37	0000f0ec	\$0008007C	log	sdata	long read
-36	0000f0e8	\$A7000000	log	sdata	long read
-35	00000200	\$0000040B	twalk	prog	long read
-34	00000400	\$0000060B	twalk	prog	long read
-33	00000600	\$0000009f	twalk	prog	long read
-32	00000008	\$60FE0000	log	sprog	long read
-31	0000000c	\$00000000	log	sprog	long read
-30	00000008	\$60FE0000	log	sprog	long read
-29	0000000c	\$00000000	log	sprog	long read
-28	0000f0ec	\$00000008	log	sdata	long write
-27	0000f0ea	\$----2024	log	sdata	word write
-26	0000f0e6	\$----0000	log	sdata	word write
-25	0000f0e8	\$0008----	log	sdata	word write
-24	00000200	\$0000040B	mmu	twalk	data long read
-23	00000400	\$0000060B	mmu	twalk	data long read
-22	00000600	\$0000009F	mmu	twalk	data long read
-21	00000024	\$00001680	log	sdata	long read
-20	0000f0e4	\$A700----	log	sdata	word write
-19	00001680	\$2F0D4BFA	log	sprog	long read
-18	00001684	\$FB523ABC	log	sprog	long read
-17	00001688	\$20246000	log	sprog	long read
-16	0000168c	\$00924BFA	log	sprog	long read
-15	0000f0e0	\$00000000	log	sdata	long write
-14	00001718	\$2F256000	log	sprog	long read

Chapter 15: Connecting the Emulator to a Target System Installing Emulator Features

```
-13 000011d6 $----2024 log sdata word write
-12 0000171c $0022083A log sprog long read
-11 00001720 $0002F9D6 log sprog long read
-10 00001724 $67184BFA log sprog long read
-9 00001728 $F9F108D5 log sprog long read
-8 000010f8 $F5----- log sdata byte read
-7 0000172c $0003484F log sprog long read      <- monitor bkpt
-6 00001730 $4E7AD002 log sprog long read
-5 00001734 $4A8D6A06 log sprog long read
-4 00001738 $F4784BFA log sprog long read
-3 00001119 $--03---- log sdata byte read
-2 0000173c $F8C40C3A log sprog long read
-1 00001119 $--0B---- log sdata byte write
0 00000000 $41----- bkpt ack (buserror)      <- acknowledge
1 0000f0de $----0010 log sdata word write
2 0000f0da $----0000 log sdata word write
3 0000f0dc $172E---- log sdata word write
4 00000010 $000016A2 log sdata long read
5 0000f0d8 $2704---- log sdata word write
6 000016a0 $007E2F0D log sprog long read
7 000016a4 $4BFAFA73 log sprog long read
```

At the end of the execution of the first target program instruction, the processor takes a trace exception. Stacking for this trace exception commences and at some point, the modified trace vector is fetched. The monitor internally uses a breakpoint instruction, but it is not part of the entry sequence.

If an error occurs during modification of the trace vector, an error message similar to the following is displayed.

```
!ERROR 170! Target bus error: 0ff800024@sd
!ERROR 156! Unable to modify trace vector to 000001680 for single stepping
!ERROR 680! Stepping failed
```

Chapter 15: Connecting the Emulator to a Target System

Installing Emulator Features

If the emulator does not reenter the monitor after stepping, as indicated by the following error message, there can be a number of explanations. If the emulator steps an instruction that modifies the VBR, the step will fail because the modified trace vector will not be used to reenter the monitor. To get around this problem, the trace vector in the target program can be modified to point to the monitor entry point `<monaddress + 0680>`.

```
!ERROR 680! Stepping failed
```

Stepping will behave differently when executing instructions that cause the processor to take exceptions. Most exceptions preempt the trace exception until after their exception handler runs. Other exceptions (like TRAP, CHK, and CHK2) create their stack frame and then take the trace exception.

For all exceptions except TRAP, CHK, and CHK2, the exception handler will execute before the trace exception is taken to return to the monitor. Exception handlers that are instruction emulators are responsible for emulating the trace behavior as well. If they do not emulate this behavior, stepping may fail because the trace exception will never happen.

The TRAP, CHK, and CHK2 exception handlers do not run before the trace exception is taken. They will have an additional stack frame when the monitor is entered. The exception stack frame will precede the normal trace stack frame.

Installing emulation memory

The last feature of the emulator that you need to integrate is the emulation memory. Emulation memory is intended to overlay ROM in the target system. This allows changes to target programs to be quickly loaded into a system. Emulation memory is not dual ported as is the case with the monitor memories. To display and modify emulation memory, you must use the monitor.

If emulation memory is placed over existing target memory, interlock it to the target memory strobes. This ensures that the target memory control circuits remain in sync with the emulator. If there are no strobes that respond in the address range where emulation memory is placed, then do not interlock. When interlocked, both the TA and TEA signals are sampled.

Chapter 15: Connecting the Emulator to a Target System

Installing Emulator Features

```
!ERROR 170! Target failed to terminate bus cycle: 000000000@sd word read
!STATUS 170! Emulator terminated hung bus cycle: 000000000@sd word read
!ERROR 702! Emulation memory access failed
```

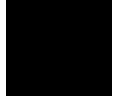
To effectively use emulation memory, the monitor must be able to read and write to it. Read and write accesses to emulation memory are seen by the target system. Emulation memory will not be able to be loaded if it is interlocked and the target system asserts bus error on write cycles or does not terminate the cycle.

```
!ERROR 170! Target bus error: 0000badad@sd
!ERROR 702! Emulation memory access failed
```

If the memory is write protected by the MMU, the monitor will temporarily disable this protection to complete the write. This applies to both emulation memory and target memory. Once emulation memory is mapped, it can be tested by performing accesses from the monitor.

If the MMU is turned on and there are no address translations for the requested emulation memory access, you will see the following error message:

```
!ERROR 170! Address translation error; non-resident page: 000f84000@sd
!ERROR 702! Emulation memory access failed
```





16



Installation and Service

Installation

This chapter shows you how to install emulation and analysis hardware and interface software. It also shows you how to verify installation by starting the emulator/analyzer interface for the first time. These installation tasks are described in the following sections:

- Installing hardware.
- Verifying the installation and performance of the emulator.
- Ensuring software compatibility
- List of replaceable parts.

Installing Hardware

This section describes how to install emulation and analysis hardware and how to connect the emulator probe to the demo target system.

Equipment supplied

The minimum system contains:

- HP 64783A/B 68040/68EC040/68LC040 PGA Emulator Probe (which includes the demo target system).
- HP 64748C Emulation Control card.
- HP 64794 Emulation-Bus Analyzer (deep analyzer) card, or HP 64704A Emulation-Bus Analyzer (1K analyzer) card.
- Ribbon cable.
- HP 64700 Card Cage.

Optional parts are:

- HP 64172A 256-Kbyte Memory Modules for additional memory depth.
- HP 64172B 1-Mbyte Memory Modules for additional memory depth.
- HP 64173A 4-Mbyte Memory Modules for additional memory depth.
- HP 64708A Software Performance Analyzer.

Equipment and tools needed

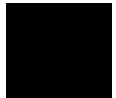
In order to install and use the MC68040 emulation system, you need:

- Flat-blade screwdriver with shaft at least 5 inches long (13 mm approx).

Installation overview

The steps in the installation process are:

- 1 Install optional memory modules on the deep analyzer card, if desired.
- 2 Connect the HP 64783A/B emulator probe to the HP 64748C emulator control card.
- 3 Install cards into the HP 64700 card cage.
- 4 Install emulation memory modules on the emulator probe.
- 5 Connect the emulator probe to the demo target system.
- 6 Apply power to the HP 64700 Card Cage.



Antistatic precautions

Printed-circuit boards contain electrical components that are easily damaged by small amounts of static electricity. To avoid damage to the emulator boards, follow these guidelines:

- If possible, work at a static-free workstation.
- Handle the boards only by the edges; do not touch components or traces.
- Use a grounding wrist strap that is connected to the HP 64700's chassis.

Caution

If you already have a modular HP 64700A Series Card Cage and want to remove the existing emulator and insert an HP 64783A/B emulator in its place, the HP 64700 Series generic firmware and analyzer firmware may NOT be compatible, and the software will indicate incompatibility. In this event, you must purchase a Flash EPROM board to update the firmware. Instructions for installing this board and programming it from a PC or HP 9000 are provided in the HP 64700A Card Cage Installation/Service manual. Instructions for installing and updating emulator firmware are covered in Chapter 20, "Installing/Updating Emulator Firmware".

Note

If you already have a modular HP 64700 Series Card Cage and want to remove the 1K analyzer and install the deep analyzer in its place, the analyzer firmware will be updated by your installation because the analyzer firmware is contained on the analyzer card.

Checking Hardware Installation

After hardware installation, run a performance test to verify that the emulator is working properly. The performance verification procedure is described under "Verifying the Installation" later in this chapter.

Service Information

Use this chapter when removing and installing hardware, running performance verification, and ordering parts. See the HP 64700 Series Installation/Service Guide for information on system configurations, installing product software, software updates, and ordering parts for the card cage. Turn off power to the card cage before removing or installing hardware.

Step 1. Install optional memory modules on Deep Analyzer card, if desired

Observe antistatic precautions

With no optional memory modules installed on the deep analyzer card, the trace memory depth is 8K. If you are going to use the deep analyzer with this default trace memory depth, skip this step and proceed to Step 2 of this installation procedure.

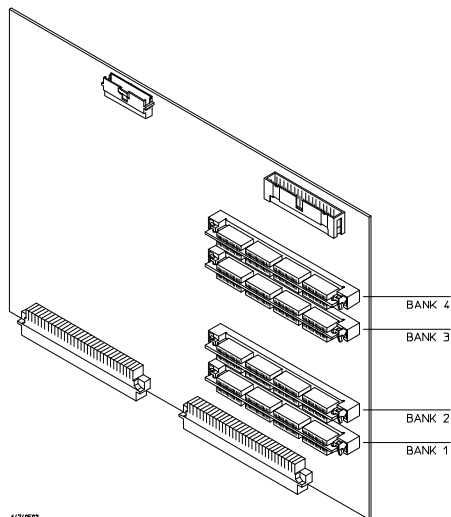
1 Determine placement of the optional memory modules. Two types of modules may be installed: 256-Kbyte (HP 64172A), and 1-Mbyte (HP 64172B). Either module type may be installed in the banks on the analyzer card. Do not use HP 64171A/B or HP 64173A memory modules; they are too slow.

If you install no memory modules, the deep analyzer will have 8K maximum memory depth.

If you install four 256-Kbyte memory modules, the analyzer will have 64K maximum memory depth.

If you install four 1-Mbyte memory modules, the analyzer will have 256K maximum memory depth.

If you install a combination of 256-Kbyte memory modules and 1-Mbyte memory modules, the analyzer will have 64K maximum memory depth. All four connectors must have memory modules installed before the analyzer depth will be increased.



Chapter 16: Installation and Service
Installing Hardware

2 To ensure correct installation of optional memory modules on the deep analyzer card, there is a cutout at one end of the memory modules so they can only be installed the correct way.

To install a memory module:

Align the groove in the memory module with the alignment rib in the connector.

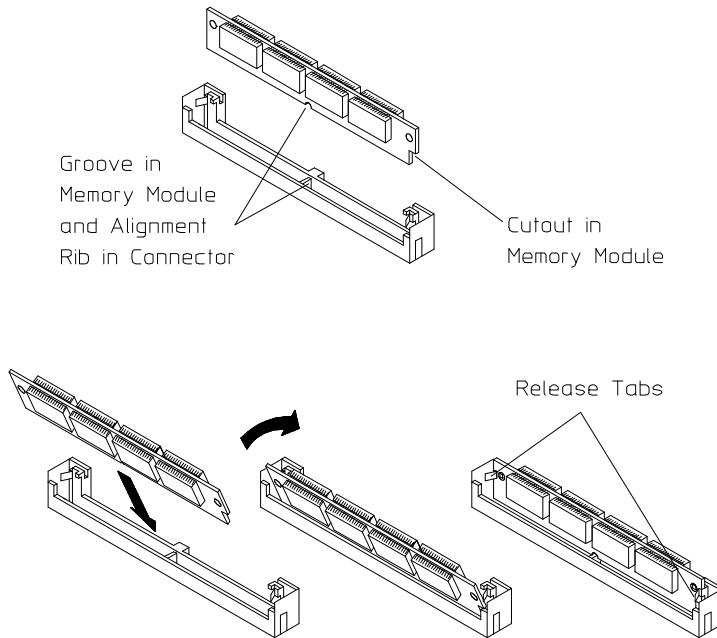
Align the cutout in the memory module with the projection in the connector.

Place the memory module into the connector groove at an angle.

Firmly press the memory module into the connector and make sure it is completely seated.

Rotate the memory module forward so that the pegs on the connector fit into the holes on the memory module.

Make sure the release tabs at each end of the connector snap around the memory module to hold it in place.



64794E03

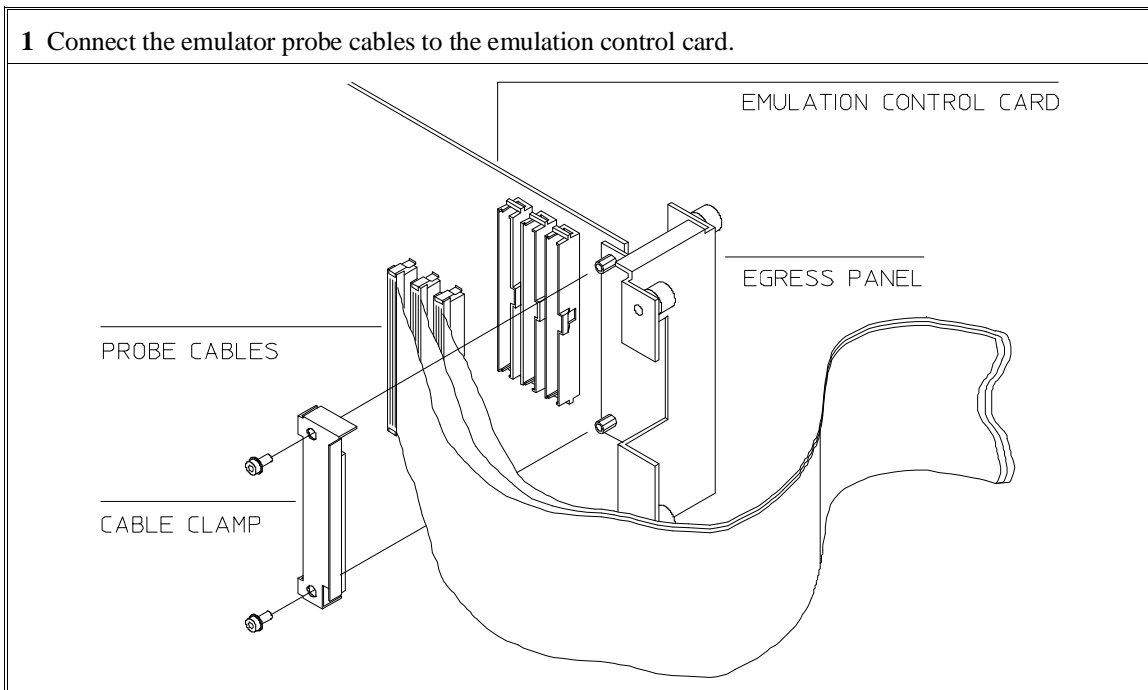
Step 2. Connect the Emulator Probe Cables

Three ribbon cables connect the HP 64748C emulation control card to the HP 64783A/B emulator probe.

The shortest cable connects from J1 of the emulation control card to J3 of the emulator probe. The medium length cable connects from J2 of the emulation control card to J2 of the emulator probe. The longest cable connects from J3 of the emulation control card to J1 of the emulator probe.

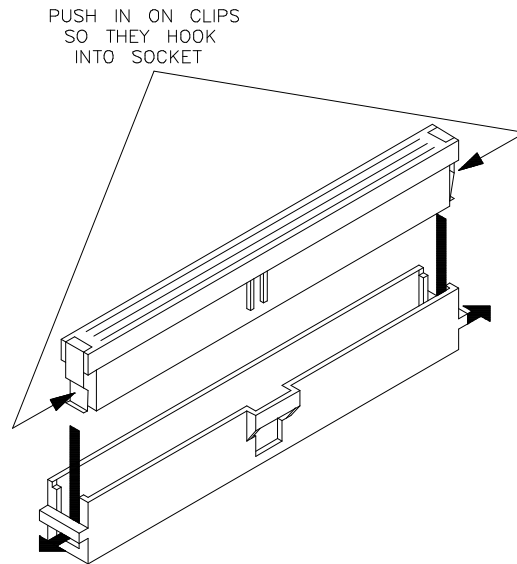
Make sure the cable connectors are seated. There are stainless steel clips on the cable connectors; these must be properly latched inside the sockets. Otherwise, the cables will work loose and you will see erratic operation. See illustration next page (step 2).

1 Connect the emulator probe cables to the emulation control card.

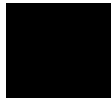
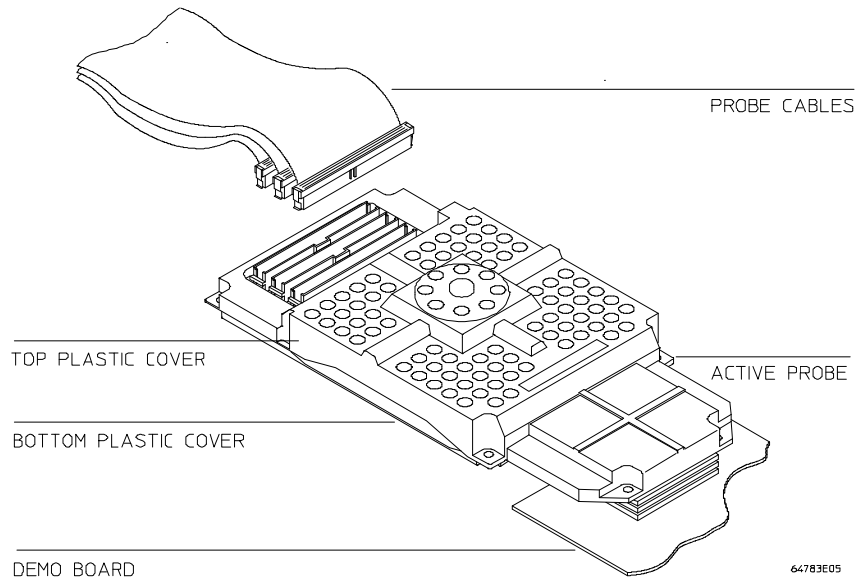


Chapter 16: Installation and Service
Installing Hardware

2 When inserting cable connectors into the sockets, press inward on the connector clips so that they hook into the sockets as shown. The order of connecting cables was given in step 1.



3 Connect the other ends of the cables to the emulator probe. Again, make sure the stainless steel clips on the cable connectors are properly latched within the sockets, as shown in step 2.



Step 3. Install Boards into the HP 64700 Card Cage

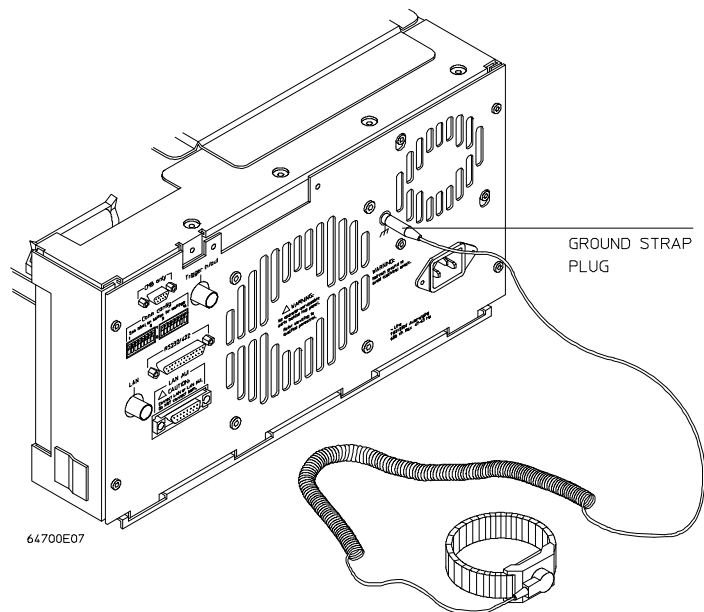
WARNING

Before removing or installing parts in the HP 64700 Card Cage, make sure that the card cage power is off and that the power cord is disconnected.

CAUTION

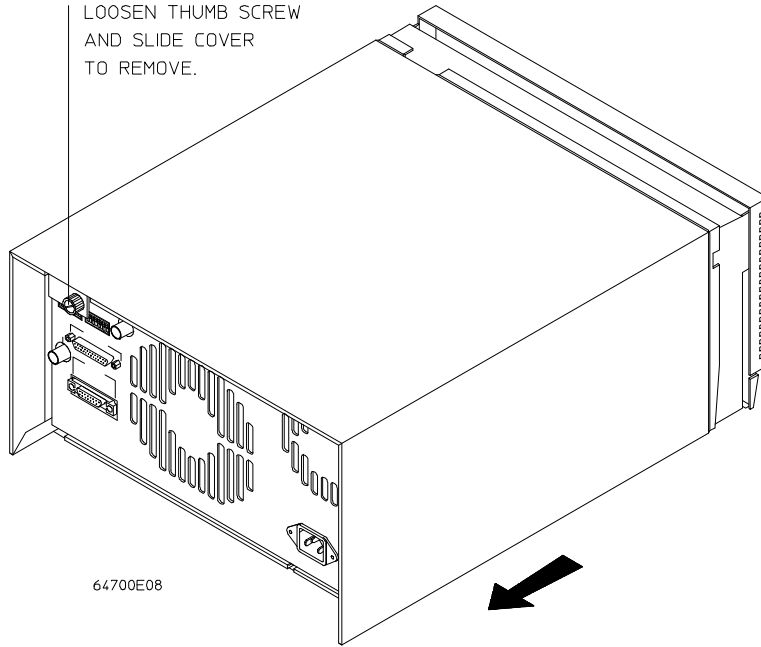
Do NOT stand the HP 64700 Card Cage on the rear panel. You could damage the rear panel ports and connectors.

- 1 Use a ground strap when removing or installing boards into the HP 64700 Card Cage to reduce the risk of damage to the circuit cards from static discharge. A jack on the rear panel of the HP 64700 Card Cage is provided for this purpose.



2 Turn the thumb screw and remove the top cover by sliding the cover toward the rear and up.

LOOSEN THUMB SCREW
AND SLIDE COVER
TO REMOVE.

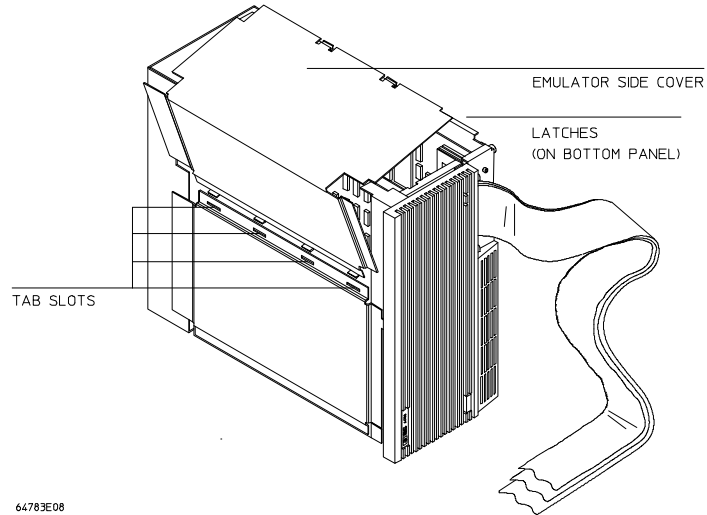


64700E08

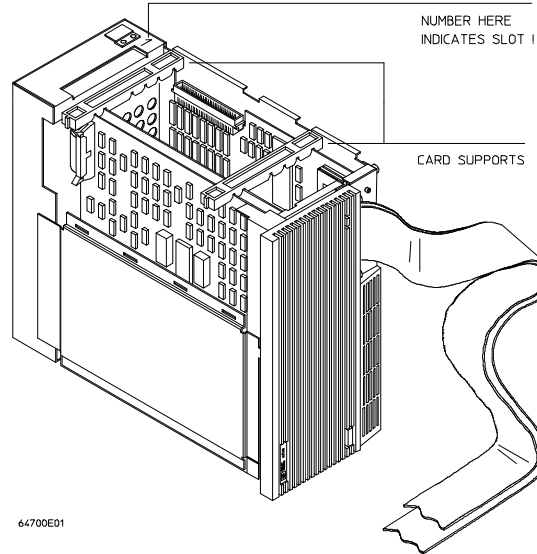


Chapter 16: Installation and Service
Installing Hardware

3 Remove the side cover by unsnapping the two latches and lifting off.

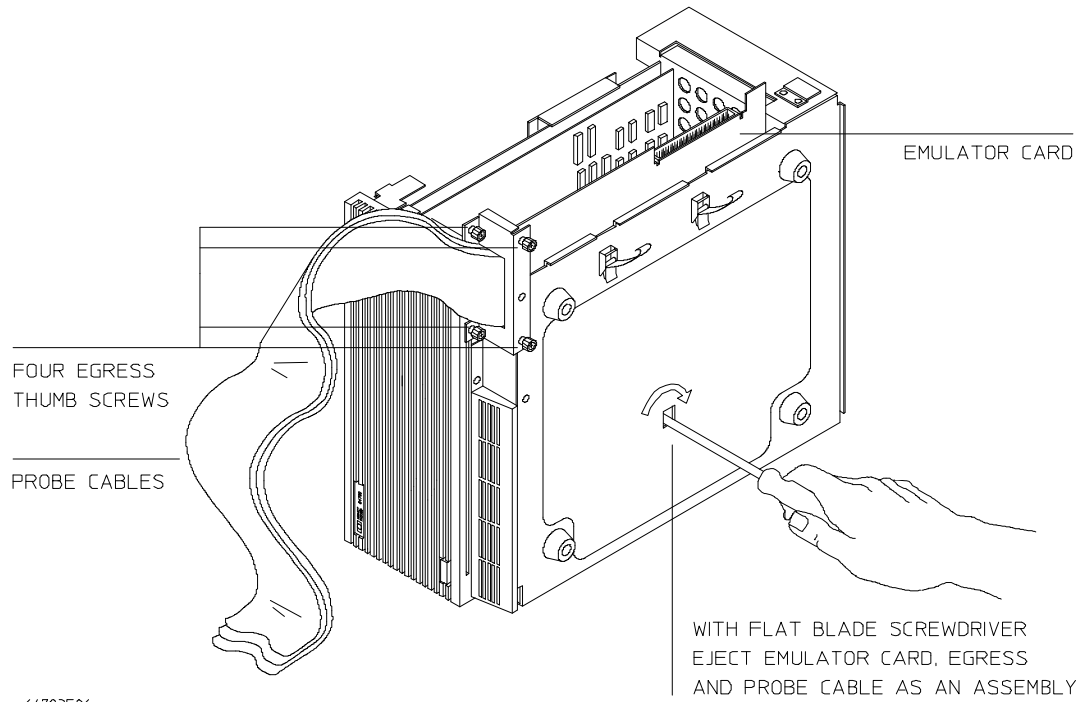


4 Remove the card supports.



5 First, completely loosen the four egress thumb screws.

To remove emulator cards, insert a flat blade screwdriver in the access hole and eject the emulator cards by rotating the screwdriver.

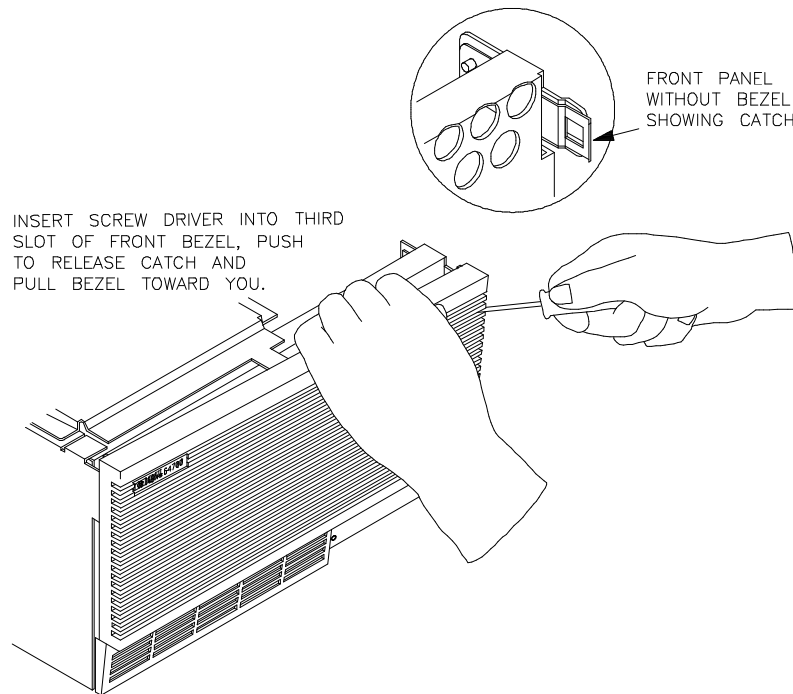


64783E06

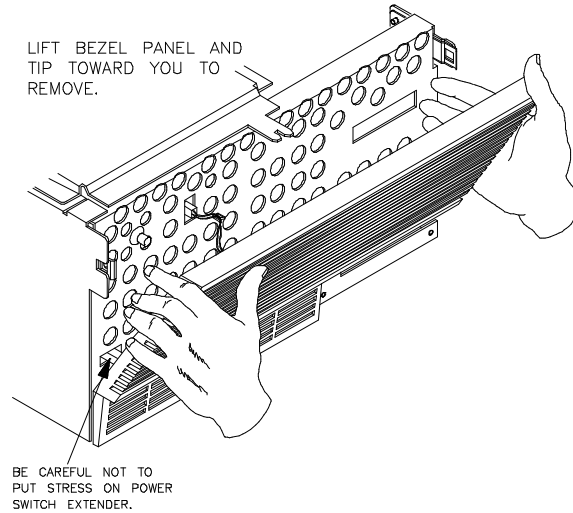
Chapter 16: Installation and Service
Installing Hardware

6 Insert a screw driver into the third slot of the right side of the front bezel, push to release catch, and pull the right side of the bezel about one-half inch away from the front of the HP 64700. Then, do the same thing on the left side of the bezel. When both sides are released, pull the bezel toward you approximately 2 inches.

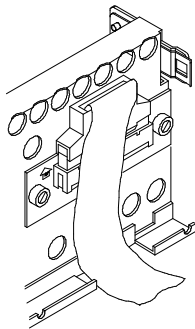
Be careful because the plastic ears are easily broken on the front bezel.



7 Lift the bezel panel to remove. Be careful not to put stress on the power switch extender.

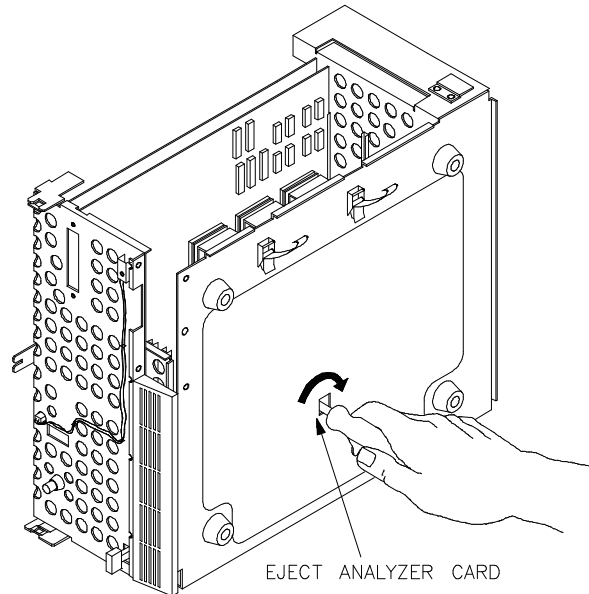


8 If you're removing an existing analyzer card that provides external analysis, remove the right-angle adapter board by turning the thumb screws counterclockwise.



Chapter 16: Installation and Service
Installing Hardware

9 To remove the analyzer card, insert a flat blade screwdriver in the access hole and eject the analyzer card by rotating the screwdriver.

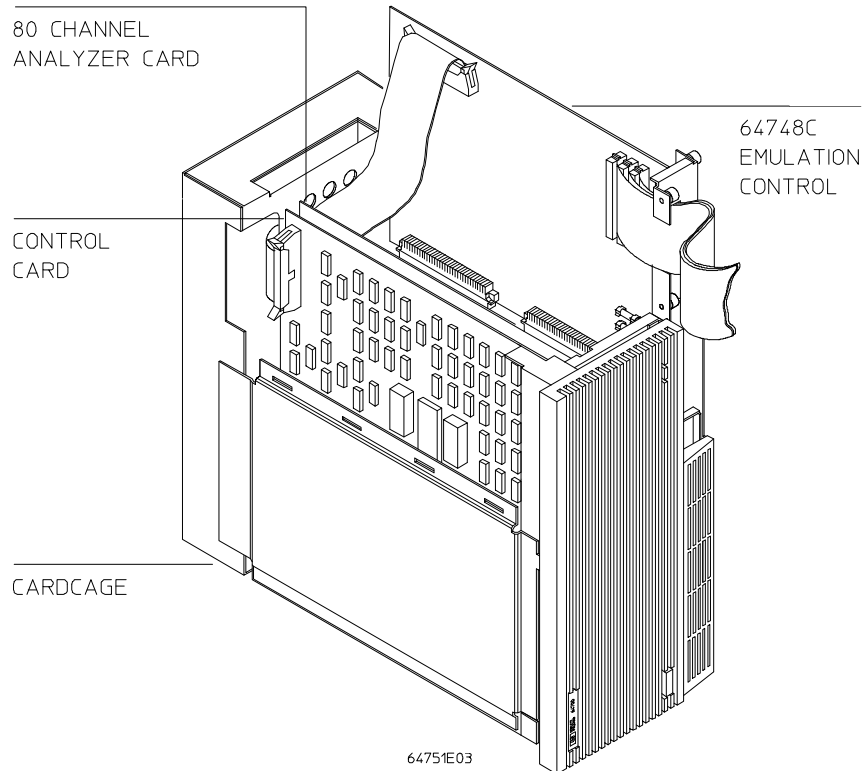


Do not remove the system control board. This board is used in all HP 64700 emulation and analysis systems.

10 Install the analyzer and emulation control cards. The analyzer is installed in the slot next to the system control card. The emulation control card is installed in the second slot from the bottom of the card cage. The software performance analyzer card may occupy any slot between the emulation-bus analyzer and the emulation control card. These cards are identified with labels that show their model numbers and serial numbers. Note that components on the analyzer card face the opposite direction to the other cards.

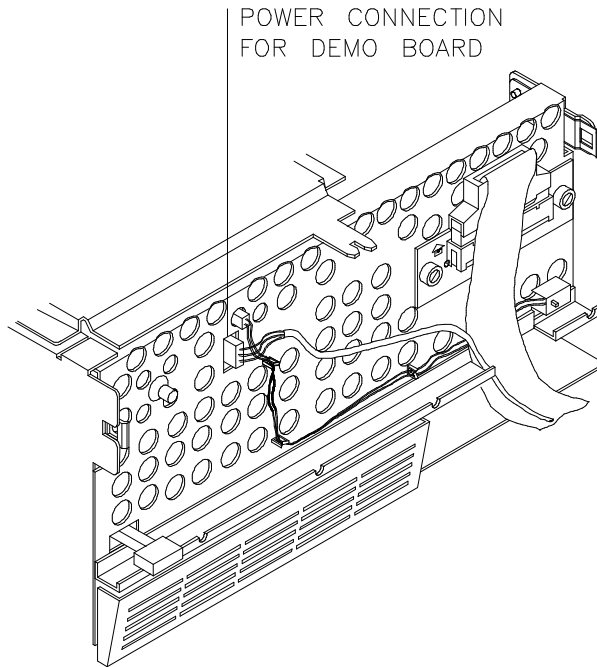
To install a card, insert it into the plastic guides. Make sure the connectors are properly aligned; then, press the card into the mother board socket. Ensure that each card is seated all the way into its socket. If the cards can be removed with your fingers, the cards are NOT seated all the way into the mother board sockets.

Attach the ribbon cable from the emulation control card to the analyzer card, and to the software performance analyzer, if installed. Tighten the thumbscrews that hold the emulation control card to the cardcage frame.

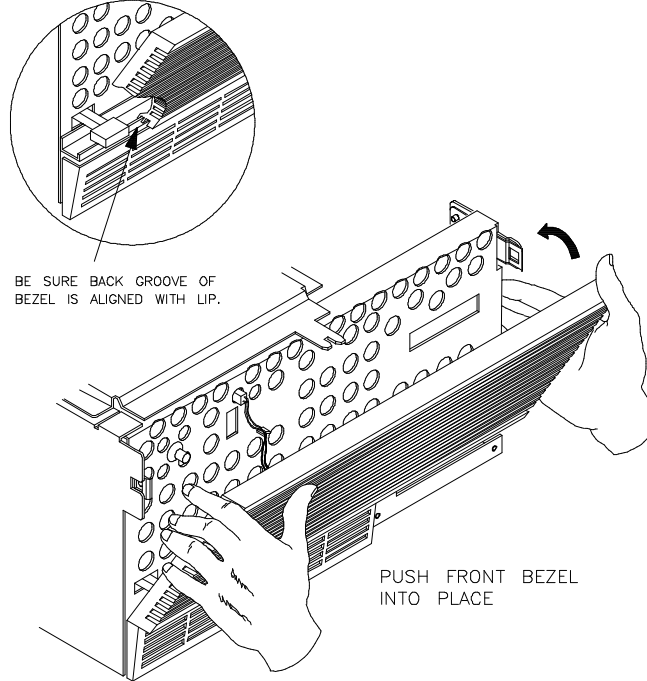


Chapter 16: Installation and Service
Installing Hardware

11 Connect the +5 V power cable to the connector in the HP 64700 front panel.



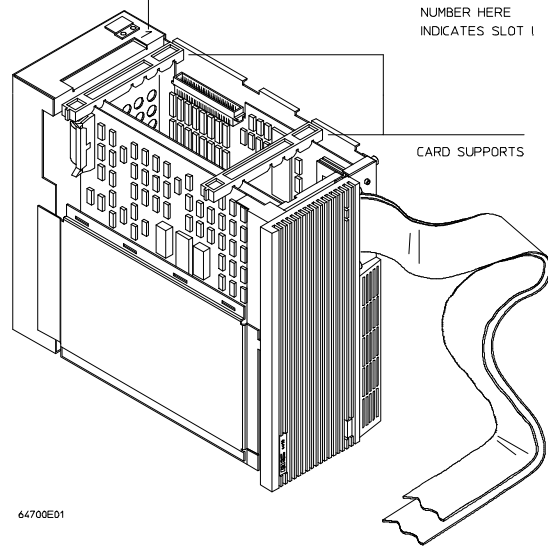
12 To reinstall the front bezel, be sure that the bottom rear groove of the front bezel is aligned with the lip as shown below.



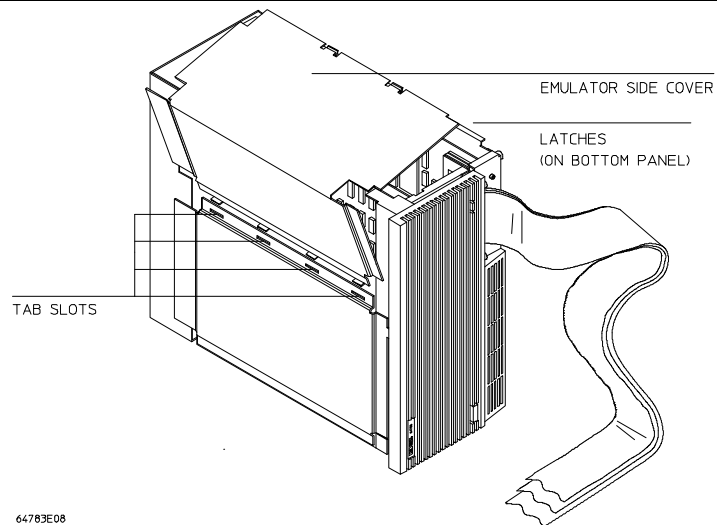
13 If you wish to install the Flash card (used for updating firmware, see the Installing/Updating Emulator Firmware chapter), refer to the diagram above. Install the flash card in any available slot between the HP 64704A and the HP 64748C cards in the cardcage. Insert the flash card in the plastic guides. Make sure the connectors are properly aligned. Then press the card into the mother board sockets. Make sure the card is seated all the way into the sockets.

Chapter 16: Installation and Service
Installing Hardware

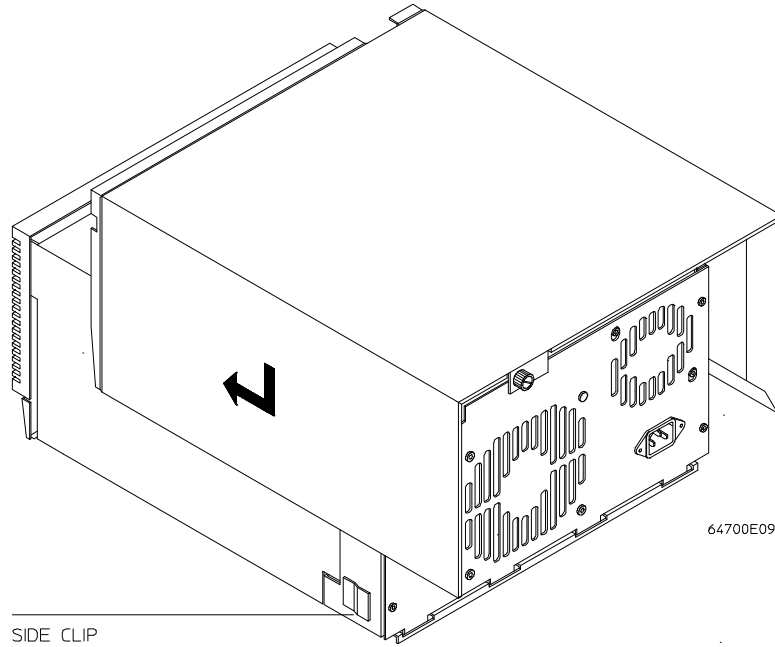
14 Install the card supports.



15 To install the side cover, insert the side cover into the tab slots and fasten the two latches.



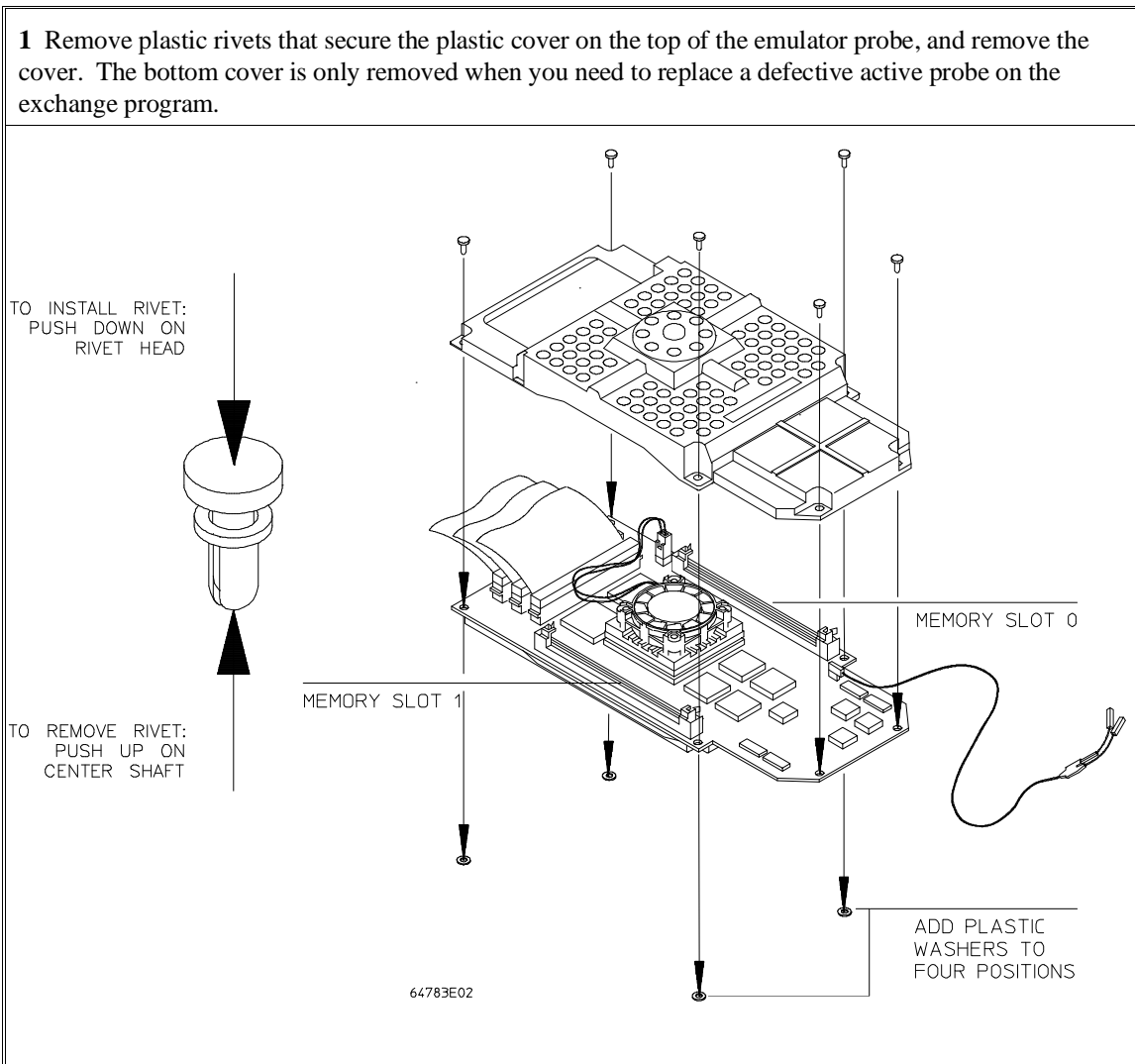
16 Install the top cover in reverse order of its removal, but make sure that the side panels of the top cover are attached to the side clips on the frame.



Step 4. Install emulation memory modules on emulator probe

(Observe antistatic precautions)

1 Remove plastic rivets that secure the plastic cover on the top of the emulator probe, and remove the cover. The bottom cover is only removed when you need to replace a defective active probe on the exchange program.



2 Determine the placement of the emulation memory modules. Three types of modules may be installed: 256 Kbyte (HP 64172A), 1 Mbyte (HP 64172B), and 4 Mbyte (HP 64173A). Any of the emulation memory modules can be installed in either memory slot on the probe. Do not use HP 64171A/B modules; they are too slow.

Memory in memory slot 0 is divided into four equal blocks that can be allocated by the memory mapper. Memory in memory slot 1 is divided into two equal blocks.

If you have only one emulation memory module, place it in memory slot 0.

If you have two memory modules of different sizes, place the memory module with the greatest capacity in memory slot 0 to take advantage of the way memory slot 0 and memory slot 1 are divided by the emulator. For example, if you install a 1-Mbyte module in memory slot 0 and a 256-Kbyte module in memory slot 1, then the emulator will provide four 256-Kbyte blocks of memory in memory slot 0 and two 128-Kbyte blocks of memory in memory slot 1.

Refer to the step called "To assign memory map terms" in the chapter titled "Configuring the Emulator" for details of the combinations of memory module installations.

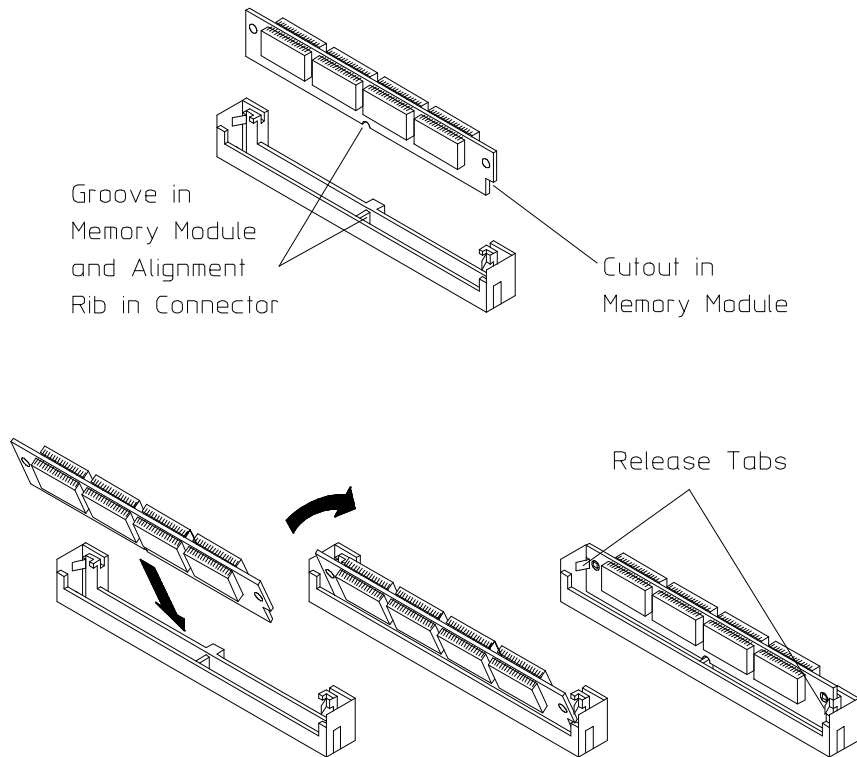


Chapter 16: Installation and Service
Installing Hardware

3 Install emulation memory modules on the emulator probe. There is a cutout at one end of the memory modules so they can only be installed the correct way.

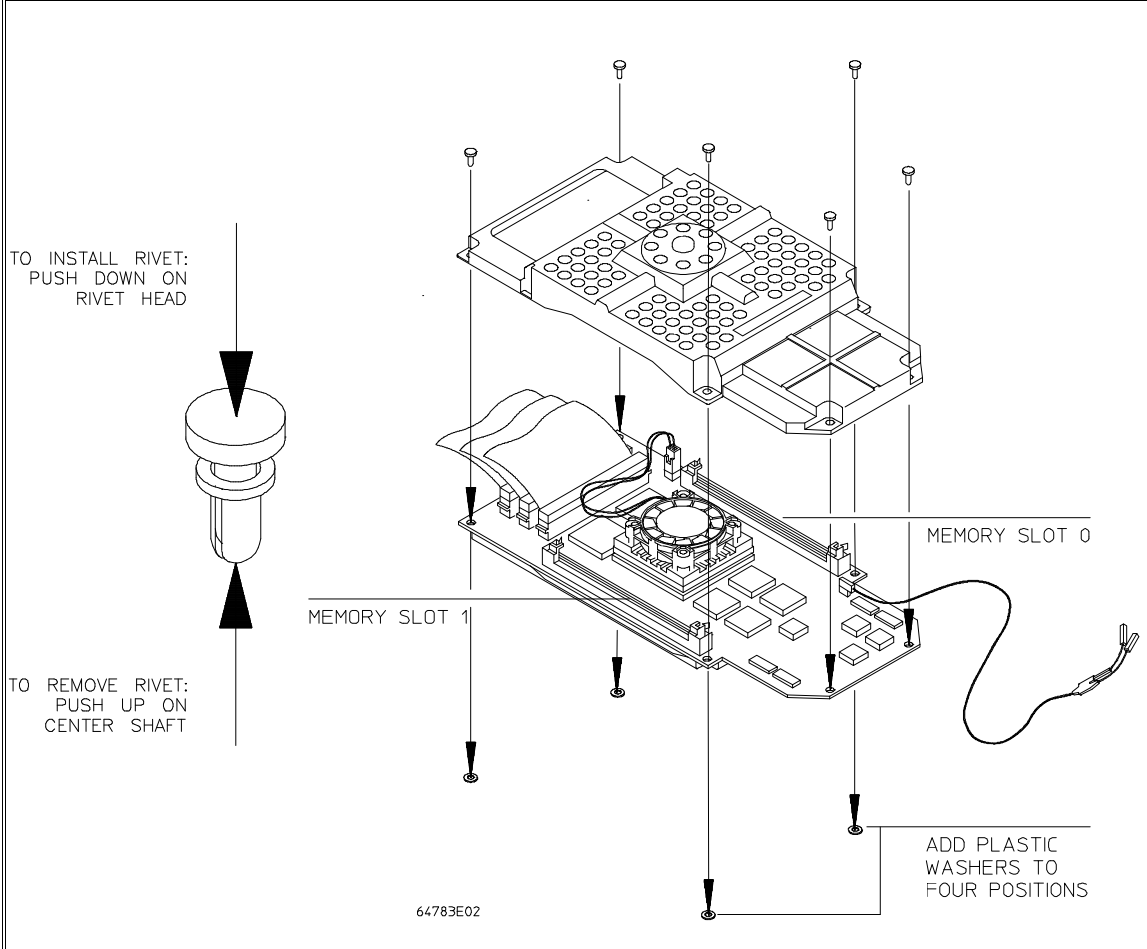
To install a memory module:

- 1** Align the groove in the memory module with the alignment rib in the connector.
- 2** Align the cutout in the memory module with the projection in the connector.
- 3** Place the memory module into the connector groove at an angle.
- 4** Firmly press the memory module into the connector and make sure it is completely seated.
- 5** Rotate the memory module forward so that the pegs on the connector fit into the holes on the memory module.
- 6** Make sure the release tabs at each end of the connector snap around the memory module to hold it in place.



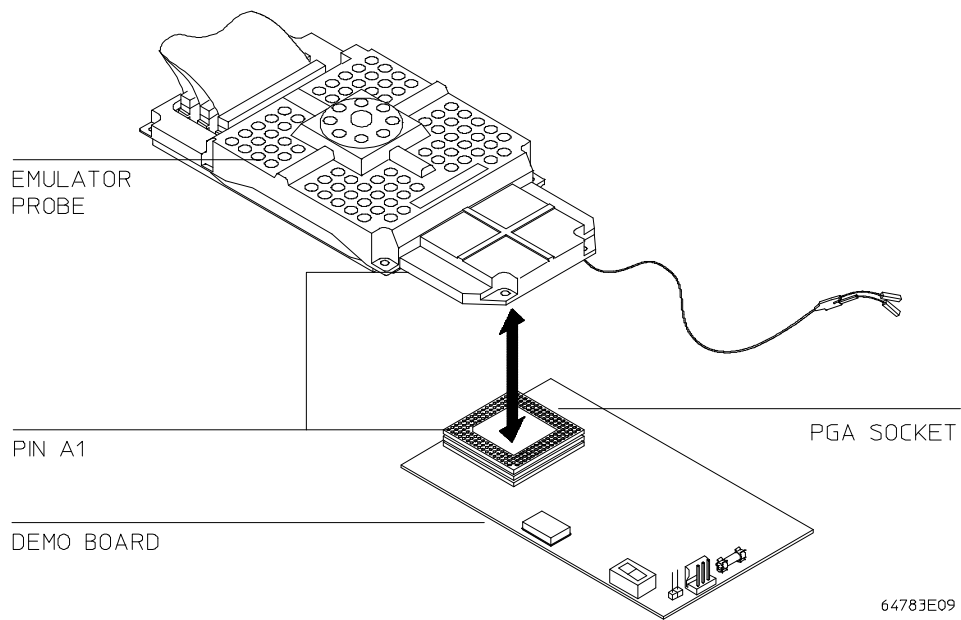
64794E03

4 Replace the plastic cover, and insert new plastic rivets (supplied with the emulator) to secure the cover.

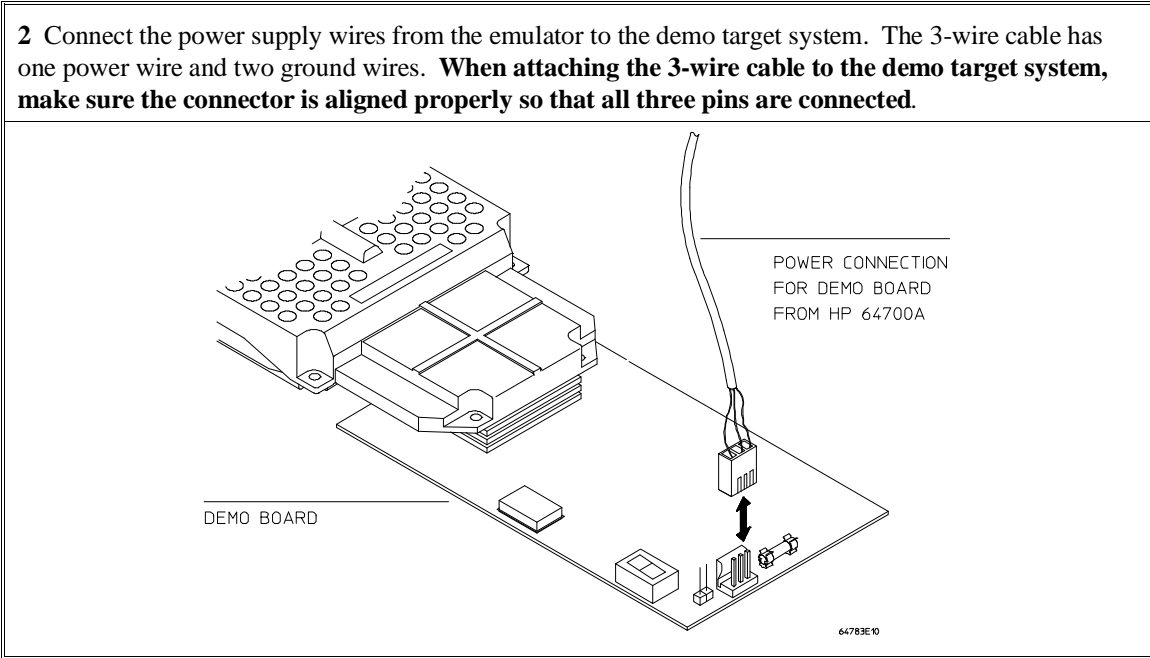


Step 5. Connect the emulator probe to the demo target system

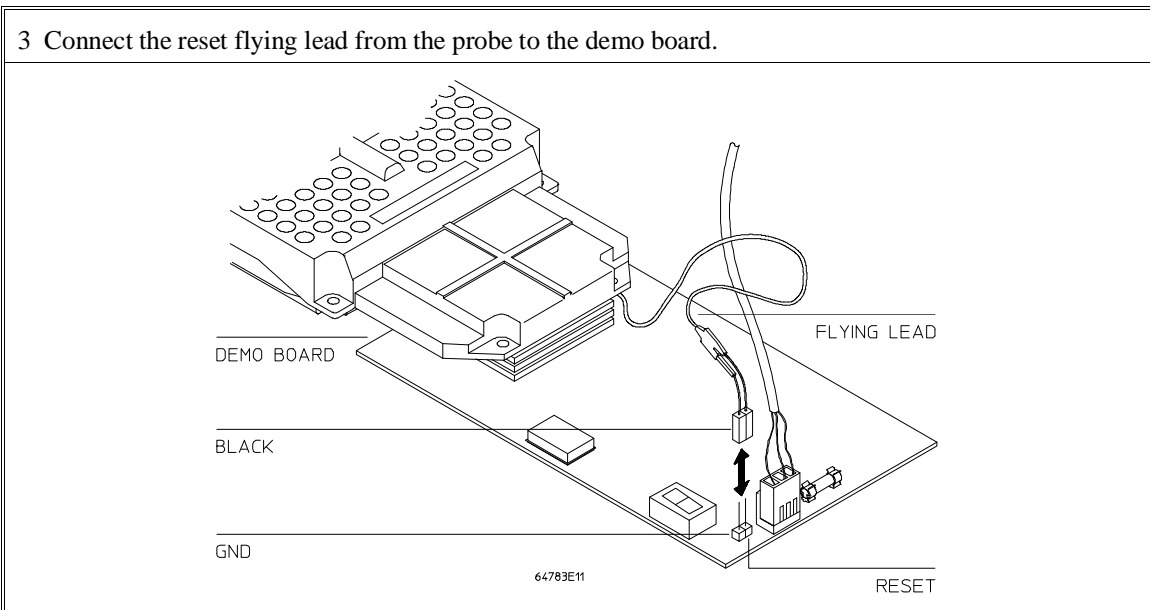
1 With HP 64700 power OFF, connect the emulator probe to the demo target system. When you install the probe into the demo board, be careful not to bend any of the pins. Do not insert the probe of the MC68040 emulator into the demo board socket incorrectly. Be very careful.



2 Connect the power supply wires from the emulator to the demo target system. The 3-wire cable has one power wire and two ground wires. **When attaching the 3-wire cable to the demo target system, make sure the connector is aligned properly so that all three pins are connected.**



3 Connect the reset flying lead from the probe to the demo board.



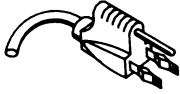


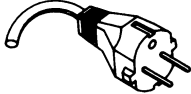
Step 6. Apply power to the HP 64700

The HP 64700B automatically selects the 115 Vac or 220 Vac range. In the 115 Vac range, the HP 64700B will draw a maximum of 345 W and 520 VA. In the 220 Vac range, the HP 64700B will draw a maximum of 335 W and 600 VA.

The HP 64700 is shipped from the factory with a power cord appropriate for your country. You should verify that you have the correct power cable for installation by comparing the power cord you received with the HP 64700 with the drawings under the "Plug Type" column of the following table.


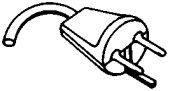

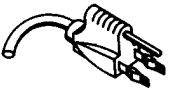
If the cable you received is not appropriate for your electrical power outlet type, contact your Hewlett-Packard sales and service office.

Power Cord Configurations

Plug Type	Cable Part No.	Plug Description	Length in/cm	Color
Opt 903 124V ** 	8120-1378	Straight * NEMA5-15P	90/228	Jade Gray
	8120-1521	90°	90/228	Jade Gray
Opt 900 250V 	8120-1351	Straight * BS136A	90/228	Gray
	8120-1703	90°	90/228	Mint Gray
Opt 901 250V 	8120-1369	Straight * NZSS198/ASC	79/200	Gray
	8120-0696	90°	87/221	Mint Gray
Opt 902 250V 	812001689	Straight * CEE7-Y11	79/200	Mint Gray
	8120-1692	90°	79/200	Mint Gray
	8120-2857	Straight (Shielded)	79/200	Coco Brown
* Part number shown for plug is industry identifier for plug only. Number shown for cable is HP part number for complete cable including plug. ** These cords are included in the CSA certification approval for the equipment.				

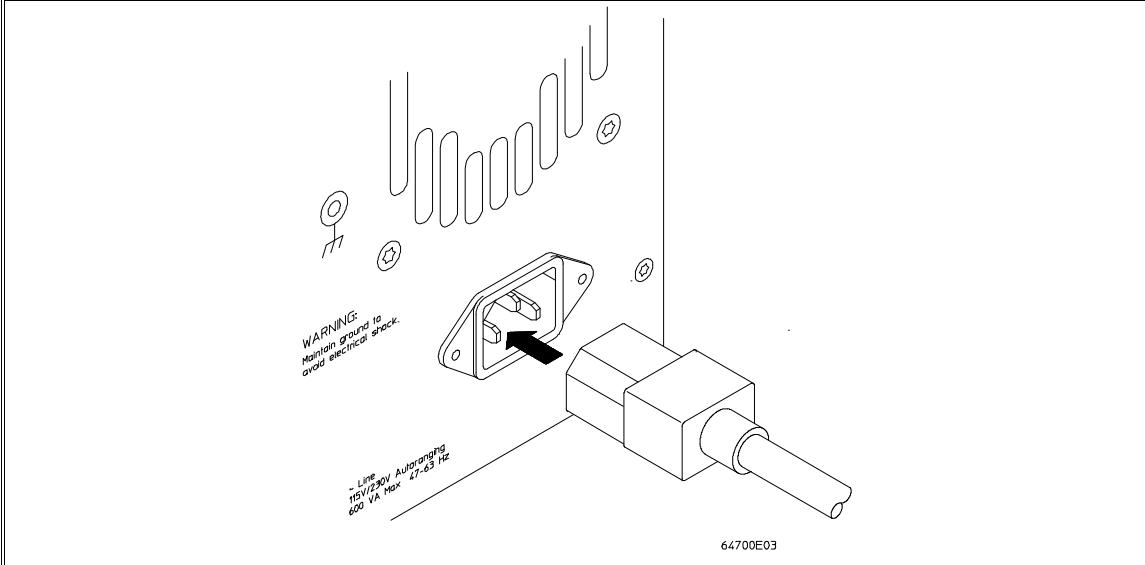


Power Cord Configurations (Cont'd)

Plug Type	Cable Part No.	Plug Description	Length in/cm	Color
Opt 906 250V 	8120-2104	Straight * SEV1011	79/20	Mint Gray
	8120-2296	1959-24507 Type 12 90°	79/200	Mint Gray
Opt 912 220V 	8120-2957	Straight *DHCK107 90°	79/200 79/200	Mint Gray Mint Gray
	8120-4600 8120-4211	Straight SABS164 90°	79/200 79/200	Jade Gray
Opt 917 250V 	8120-4600	Straight SABS164 90°	79/200	Jade Gray
	8120-4211	Straight SABS164 90°	79/200	Jade Gray
Opt 918 100V 	8120-4753	Straight Miti	90/230	Dark Gray
	8120-4754	90°	90/230	Dark Gray
* Part number shown for plug is industry identifier for plug only. Number shown for cable is HP part number for complete cable including plug. ** These cords are included in the CSA certification approval for the equipment.				

1 Connect the power cord and turn on the HP 64700.

The line switch is a push button located at the lower, left-hand corner of the front panel. To turn ON power to the HP 64700, push the line switch button in to the ON (1) position. The power light at the lower, right-hand corner of the front panel will be illuminated.



To verify the performance of the emulator

- 1 If you have a special configuration or session in progress, save it now. This procedure will cause your session to be lost.
- 2 Turn off power to the HP 64700 Card Cage.
- 3 Plug the emulator probe into the Demo Board.
- 4 Connect Demo Board power cable from the Demo Board to the HP 64700 Card Cage front panel. (See the diagrams under "Installing Hardware" in this chapter.)
- 5 Connect the Reset Flying Lead from the Emulation Probe to the Demo Board. (See "Step 4. Connect the emulator probe to the demo target system".)
- 6 Turn on power to the HP 64700 Card Cage.
- 7 Establish communication with the emulator from your host or ASCII terminal and obtain a prompt (such as **R>**).
- 8 Enter: **pv 1** <return>

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**

Chapter 16: Installation and Service Installing Hardware

A message similar to the following should appear:

```
Testing: HP64783 Motorola 68040 Emulator
PASSED:
Number of tests: 1           Number of failures: 0
Testing: HP64740 Compatible (PPN: 64794A) Deep Emulation Analyzer
PASSED:
Number of tests: 1           Number of failures: 0
```

```
Copyright (c) Hewlett-Packard Co. 1987
All Rights Reserved.  Reproduction, adaptation, or translation
without prior
written permission is prohibited, except as allowed under copyright
laws.
```

```
HP64700B Series Emulation System
Version:  B.01.00 20Dec93
Location: Flash
System RAM:1 Mbyte
```

```
HP64783 Motorola 68040 Emulator
HP64740 Compatible (PPN: 64794A) Deep Emulation Analyzer
```

If you have an emulation failure, you can replace the assembly that failed through your local Hewlett-Packard representative, and through the Support Materials Organization (SMO). Refer to the list of replaceable 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.

To verify installation of memory modules in the deep analyzer card or in the emulation probe, type the ver command as follows:

M>ver

```
HP64783 Motorola 68040 Emulator
Version:  A.04.00 22Oct92
Control:  HP64748C ABG Control Board
Speed:    33 MHz
Memory:   260 Kbytes
Bank 0:   HP64172A (20ns) 256 Kbyte Memory Module

HP64740 compatible (PPN: 64794A) Deep Emulation Analyzer
Version:  A.03.00 25Jun93 Unreleased
PC Board: 794-01-A1
Depth:    80ch X 1K states selected, 80ch x 64K states available
Bank 0:   HP64172A (20ns) 256 Kbyte Module
Bank 1:   HP64172A (20ns) 256 Kbyte Module
Bank 2:   HP64172A (20ns) 256 Kbyte Module
Bank 3:   HP64172A (20ns) 256 Kbyte Module
```

What is pv doing to the Emulator?

The performance verification procedures provide a thorough check of the functionality of all of the products installed in the HP 64700 Card Cage. The Test Suite for the HP 64783A/B Emulator consists of the following modules.

Tests available in Emulator Subsystem:

```
test # 1 (ABG68000 RAM)
test # 2 (ABG Type Map)
test # 3 (ABGDeMMU Map)
test # 4 (low DMMU RAM)
test # 5 (up DMMURAM)
test # 6 (68000 side RAM)
test # 7 (Host DPRAM)
test # 8 (Clock Test)
test # 9 (Release tobg)
test #10 (Release to fg)
test #11 (MonTransistion)
test #12 (Break Detection)
test #13(Dual Port RAM)
test #14 (Emul Mem Bank 0)
test #15(Emul Mem Bank 1)
test #16 (Demo Reset)
test #17(Demo Data)
test #18 (Demo Address)
test #19 (DemoStatus)
test #20 (Demo IPL)
test #21 (Demo Cache)
test #22 (Demo DMA)
test #23 (Demo MDIS)
test #24(Demo LED)
test #25 (Analysis Intrfc)
test #26(DeMMUer)
test #27 (CMB)
```

Troubleshooting

The test results for all of these modules are indicated by a simple PASS/FAIL message. The PASS message gives a high level of confidence that all major functions and signals are operating because it includes a loopback test that includes read and write tests to the demo board. The demo board also stimulates inputs to the emulator.

A FAIL message on the other hand indicates that one or more of the tested functions is NOT working. In this event, an HP field representative can either swap assemblies to isolate the failure to an individual board, or replace all the major assemblies shown in the replaceable parts list. The emulation memory modules and plastic cover are not part of the probe assembly. The emulation memory modules must be ordered separately and the plastic covers should be removed from the probe assembly before replacing the probe assembly.

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 HP 64700B Card Cage has Flash EPROM for the assemblies that may be installed.

The latest versions of firmware for the host controller card, and analyzer card, along with a program called *progflash*, are part of the B1471 software for the HP 9000 workstation and Sun SPARCsystems and the HP 64700 Option 006 software for PCs.

When you load all your new versions of software onto your host computer, you are now ready to load the new version of firmware from your host computer to the assemblies that are in the HP 64700 Card Cage.

To load the new firmware, you use the *progflash* command. The *progflash* command must be run from a PC or workstation; it displays a list of card cages and subassemblies in each card cage on your system. From these lists, you can select

Chapter 16: Installation and Service
Installing Hardware

which product to update. For information on using the *progflash* command, and updating your HP 64700 Series firmware, refer to the chapter titled "Installing/Updating Emulator Firmware" in this manual.



Parts List

What is an Exchange Part?

Exchange parts are shown on the parts list. A defective part can be returned to HP for repair in exchange for a rebuilt part.

Probe (exchange)

The Probe for the HP 64783A is not interchangeable with the Probe for the HP 64783B. Make sure you order the Probe replacement part number that is compatible with your emulator.

To replace the Probe on the exchange program, you must remove certain parts, and return only that part considered an exchange part. When returning the Probe, you must remove the:

- cable assembly.
- reset flying lead.
- 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:

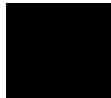
- ribbon cable that connects the Emulation Control Card to the analyzer card.
- cable assembly.
- egress panel.

Chapter 16: Installation and Service
Parts List

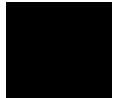
Main Assembly		
Component Part	New	Exchange
HP 64783A/B Probe and Demo Board		
68040 Emulator Firmware Floppy	64783-18000	
64700 SW UTIL	64700-18006	
MC68040 Probe Board for HP 64783A	64783-66504	64783-69504
MC68040 Probe Board for HP 64783B	64783-66505	64783-69505
(Order the following parts separately:)		
Top Plastic Cover	64783-04101	
Bottom Plastic Cover	64783-04102	
Plastic Rivets Kit (rivets and washers)	64748-68700	
Reset Flying Lead	64762-61602	
HP 64783A Demo Board for HP 64783A/B	64783-66502	
(Order the following part separately:)		
External Power Cable	5181-0201	
HP 64748C Emulation Control Card Subassembly		
Egress Panel	64748-00205	
Bracket (used with Egress Panel)	64748-01201	
Spacer, Hex M3X6	0515-1146	
Screw, Machine M3X8	0515-0372	
Cable-100 36"	64748-61601	
Cable-100 37"	64748-61602	
Cable-100 38"	64748-61603	
Cable Clamp	64744-01201	
Rubber Strip	64744-81001	
Emulation Control Card (without external cable or egress panel)	64748-66515	64748-69515
Wrist strap	9300-1405	
HP 64172A 256 Kbyte SRAM Module	64172A	64172-69501
HP 64172B 1 Mbyte SRAM Module	64172B	64172-69502
HP 64173A 4 Mbyte SRAM Module	64173A	64173-69501
HP 64794A Emulation-Bus Analyzer (deep) card	64794-66502	64794-69502

Chapter 16: Installation and Service
Parts List

Main Assembly		
Component Part	New	Exchange
34-pin ribbon cable	64708-61601	
Analyzer Card HP 64740 with 1K memory depth	64740-66526	64740-69526
34-pin ribbon cable	64772-61602	







**Installing/Updating Emulator
Firmware**

Installing/Updating Emulator Firmware

If you ordered the HP 64783A/B MC68040 emulator probe and the HP 64748C emulation control card together, the control card contains the correct firmware for the HP 64783A/B.

However, if you ordered the HP 64783A/B and the HP 64748C separately, or if you are using a HP 64748C that has been previously used with a different emulator probe, you must download the firmware for the HP 64783A/B into the emulation control card.

The firmware, and the program that downloads it into the control card, are included with the MC68040 emulator probe on the following MS-DOS format floppy disks:

- 68040 EMULATION FIRMWARE 64783
- 64700 SW UTIL

(The MC68040 emulator firmware is also included with the MC68040 PC Interface software, and the firmware update utility can also be ordered as product HP 64700S/AX Option 006.)

The steps to install or update the emulator firmware are:

- 1 Install the firmware update utility and the 64783 emulator firmware.
- 2 Run "progflash" to update emulator firmware.

The instructions in this chapter assume you have already connected the HP 64700 card cage to the PC host computer and installed the PC Interface as described in the "Installation" chapter. In other words, the following instructions assume you have already made the necessary changes to the CONFIG.SYS, AUTOEXEC.BAT, and \HP64700\TABLES\64700TAB files.

Step 1: Install the firmware update utility

Your HP Vectra PC or IBM PC AT compatible computer must have MS-DOS 3.1 or greater and a fixed disk drive. The firmware update utility and the 64783 firmware require about 300 Kbytes of disk space.

- 1 Insert the 64700 SW UTIL disk into drive A.
- 2 Change MS-DOS prompt to drive A: by typing "A:" at the MS-DOS prompt.

For example:

```
C> A: <RETURN>
A>
```

- 3 Type "INSTALL" at the MS-DOS prompt.

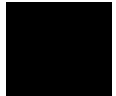
For example:

```
A> INSTALL <RETURN>
```

After confirming that you want to continue with the installation, the install program will give you the option of changing the default drive and/or subdirectory where the software will reside. The defaults are:

```
Drive = C:
Directory Path = C:\HP64700
```

Follow the remaining instructions to install the firmware update utility and the 64783 firmware.



Step 2: Run "progflash" to update emulator firmware

- Enter the `PROGFLAS [-V] [EMUL_NAME] [PRODUCTS ...]` command.

The `PROGFLAS` command downloads code from files on the host computer into Flash EPROM memory in the HP 64700.

The `-V` option means "verbose". It causes progress status messages to be displayed during operation.

The `EMUL_NAME` option is the logical emulator name as specified in the `\HP64700\TABLES\64700TAB` file.

The `PRODUCTS` option names the products whose firmware is to be updated.

If you enter the `PROGFLAS` command without options, it becomes interactive. If you don't include the `EMUL_NAME` option, `PROGFLAS` displays the logical names in the `\HP64700\TABLES\64700TAB` file and asks you to choose one. If you don't include the `PRODUCT` option, `PROGFLAS` displays the products which have firmware update files on the system and asks you to choose one. (In the interactive mode, only one product at a time can be updated.) You can abort the interactive `PROGFLAS` command by pressing `<CTRL>c`.

`PROGFLAS` will print "Flash programming SUCCEEDED" and return 0 if it is successful; otherwise, it will print "Flash programming FAILED" and return a nonzero (error).

You can verify the update by displaying the firmware version information.

Chapter 17: Installing/Updating Emulator Firmware
Step 2: Run "progflash" to update emulator firmware

Examples

To install or update the HP 64783 emulator firmware in the HP 64700 that is connected to the COM1 port:

```
C> PROGFLAS <RETURN>
```

```
HP64700S006 A.05.00 03Jan94  
64700 SW UTIL
```

```
A Hewlett-Packard Software Product  
Copyright Hewlett-Packard Co. 1991
```

All Rights Reserved. Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under copyright laws.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c) (1) (II) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013. HEWLETT-PACKARD Company, 3000 Hanover St., Palo Alto, CA 94304-1181

Logical Name	Processor
1 EMUL_COM1	m68040
2 EMUL_COM2	m68000

Number of Emulator to Update? (intr (usually cntl C or DEL) to abort)

To update firmware in the HP 64700 that is connected to the COM1 port, enter "1".

```
Product  
1 64783
```

Number of Product to Update? (intr (usually cntl C or DEL) to abort)

To update the HP 64783A/B MC68040 emulator firmware, enter "1".

Enable progress messages? [y/n] (y)

To enable status messages, enter "y".

Chapter 17: Installing/Updating Emulator Firmware

Step 2: Run "progflash" to update emulator firmware

```
Checking System firmware revision...
Mainframe is a 64700B

Reading configuration from '/hp64700/update/64783.cfg'
ROM identifier address = 2FFFF0H
Required hardware identifier = 1FFFH,1FFCH
Control ROM start address = 280000H
Control ROM size = 40000H
Control ROM width = 16
Programming voltage control address = 2FFFFEH
Programming voltage control value = FFFFH
Programming voltage control mask = 0H

Rebooting HP64700...
Checking Hardware id code...
Erasing Flash ROM
Downloading ROM code: /hp64700/update/64783.X
  Code start 280000H (should equal control ROM start)
  Code size 29A3EH (must be less than control ROM size)
Finishing up...

Rebooting HP64700...
Flash programming SUCCEEDED
C>
```

You could perform the same update as in the previous example with the following command:

```
C> PROGFLAS -V EMUL_COM1 64783 <RETURN>
```

Glossary

Absolute Count

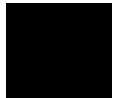
a count in the trace list count column that indicates the total count accumulated between the displayed state and the trigger state. For example, an absolute time count shown beside trace memory line number 100 indicates the elapsed time between capture of the trigger state and capture of state 100.

Absolute File

a file consisting of machine-readable instructions in which absolute addresses are used to store instructions, data, or both. These are the files that are generated by the compiler/assembler/linker and are loaded into HP 64700 Series emulators.

Access Breakpoint

a break from execution of your target program to execution of the emulation monitor when the emulator detects an access violation, such as an attempt to write to ROM or guarded memory space. The same effect can be obtained for an emulation break due to trigger recognition within the emulation-bus analyzer, or due to a signal from an external device supplied over the CMBT or the rear-panel BNC. Access breakpoints do not obtain immediate transfer to the monitor program. Several instruction cycles may be executed after the access violation occurs before execution begins in the monitor. Refer to the chapter on Using the Emulator in this manual for details of how to use breakpoints, and effects of their use on execution of your target program. Also, refer to Execution Breakpoints in this glossary.



Analyzer

an instrument that captures activity of signals synchronously with a clock signal. An emulation-bus analyzer captures emulator bus cycle information. An external analyzer captures activity on signals external to the emulator. No external analyzer is supported by the MC68040 emulator because all analysis bits are used by the emulation-bus analyzer.

Analyzer Clock Speed

the bus cycle rate of the emulation processor. If the emulation processor is running at 21 MHz and the fastest bus cycle requires three clocks, then the analyzer clock speed (bus cycle rate) is $21/3 = 7$ MHz.

Arm Condition

a condition that reflects the state of a signal external to the analyzer. The arm condition can be used in branch or storage qualifiers. External signals can be from another analyzer or an instrument connected to the CMB or BNC.

Assembler

a program that translates symbolic instructions into object code.

Background

a memory that parallels (and overlaps) the emulation processor's normal address range. Entry to background can only take place under emulator control, and cannot be reached via your target program.

Background Monitor

a monitor program that operates entirely in the background address space. The background monitor can execute when target program execution is temporarily suspended. The background monitor does not occupy any of the address space that is available to your target program.

BNC Connector

a connector that provides a means for the emulator to drive/receive a trigger signal to/from an external device (such as a logic analyzer, oscilloscope, or HP 64000-UX system).

Breakpoint

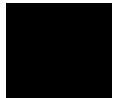
a point at which emulator execution breaks from the target program and begins executing in the monitor. (See also Execution Breakpoint and Access Breakpoint.)

Command File

a file containing a sequence of commands to be executed.

Compatible Mode

The compatible mode of the deep analyzer configures the analyzer to provide the same memory depth as the 1K analyzer: 1024 states deep when the analyzer is not configured to make a count of states or time during a measurement, and 512 states deep when the analyzer is configured to make a count of states or time during a measurement. If the emulator interface you are using along with the deep analyzer requires that you use the compatible mode, the deep analyzer will still be able to provide one of its benefits for your measurement; you will be able to make your counts of states or time at full emulator clock speed.



Compiler

a program that translates high-level language source code into object code, or produces an assembly language program with subsequent translation into object code by an assembler. Compilers typically generate a program listing which may list errors displayed during the translation process.

Counter Overflow

when the counter reaches maximum count and begins a new count from zero. The counter of the deep analyzer simply counts continuously once a trace begins; it increments its count every 20 ns, and reaches maximum count in about 22.9 minutes (22 minutes and 54 seconds). The deep analyzer sets a flag in memory and stores it along with the first state that is captured after the counter overflow occurs (first state captured after the counter begins again at zero).

Configuration File

a file in which configuration information is stored. Typically, configuration files can be modified and re-loaded to configure instruments (such as an emulator) or programs (such as the PC Interface).

Coordinated Measurement

a synchronized measurement made between the emulator and analyzer, between emulation-bus analyzer and external analyzer, or between multiple emulators or analyzers. For example, a coordinated measurement is made when two or more HP 64700 emulators/analyzers start executing together, or break into background monitors at the same time.

Coordinated Measurement Bus (CMB)

the bus that is used for communication between multiple HP 64700 Series emulators/analyzers or between HP 64700 emulators/analyzers and an HP 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.

Deep Analyzer

In this manual, the term "deep analyzer" refers to the HP 64794 Emulation-Bus Analyzer with deep trace memory.

Downloading

the process of transferring absolute files from a host computer into the emulator.

DTE (Data Terminal Equipment)

a specific RS-232C hardware interface configuration. Typically, DTE is a terminal or printer.

Emulation-bus Analyzer

a system component built into the HP 64700 that captures the emulation processor's address, data, and status information.

Emulation Memory

high-speed memory (RAM) in the emulator that can be used in place of target system memory.

Emulator

a tool that replaces the processor in your target system. The goal of the emulator is to operate just like the processor it replaces. The emulator gives information about the bus cycle operation of the processor and control over target system execution. Using the emulator, you may view contents of processor registers, target system memory, and I/O resources.

Emulator Probe

the assembly that connects the emulator to the target system microprocessor socket.



Execution Breakpoint

a BKPT instruction placed in your software in RAM, replacing the normal instruction at the RAM address. Breakpoints for code in ROM are stored in emulation hardware and jammed on the emulation bus during the fetch cycle. When the BKPT is executed, emulation immediately transfers from execution of your target program to execution of the emulation monitor. Refer to the chapter on Using the Emulator in this manual for details of how to use execution breakpoints, and effects of their use on execution of your target program. Also, refer to Access Breakpoints in this glossary.

Expression

the information that can fit into a single pattern or a single range (a pattern such as **addr=2105**, **data!=15**, or a range such as **addr=4012..401a**). A complex expression is made up of pattern, range, and arm labels joined together by various operators that define the specific condition. Each of the pattern and range labels must be previously assigned to specific simple expressions using the terminal interface commands: **tpat** and **trng**.

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

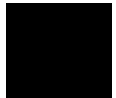
a computer to which an HP 64700 Series emulator can be connected. A host computer may run interface programs which control the emulator. Host computers may also be used to develop programs to be downloaded into the emulator.

Inverse Assembler

a program that translates absolute code into assembly language mnemonics.

Label

a set of one or more analyzer channels. Example, the label "addr" is used to identify the analyzer channels connected to the address bus of the emulation processor.



Linker

a program that combines relocatable object modules into an absolute file which can be loaded into the emulator and executed.

Logical Address Space

the addresses assigned to code during the process of compiling, assembling and linking to generate absolute files. 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.

Overflow

See counter overflow.

Parity Setting

the configuration of the parity switches. Depending on the configuration of the parity output switch and the parity switch, a parity check bit is added to the end of data to make the sum of the total bits either even or odd. A parity check is performed after data has been transferred, and is accomplished by testing a unit of the data for either odd or even parity to determine whether an error has occurred in reading, writing, or transmitting the data.

Path

also referred to as a directory (for example \users\projects).

PC Interface

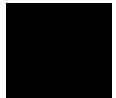
a program that runs on the HP Vectra and IBM PC/AT compatible computers. This is a friendly interface used to operate an HP 64700 Series emulator.

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.

Prestore Qualifier

a specification that must be met by a state before it can be saved in the analyzer prestore memory.

Qualifier

a specification that must be met before an action can be taken by the analyzer. For example, a store qualifier is a specification that must be met by an incoming state before it can be stored in the trace memory. The "arm" condition can be used as an additional qualifier. For example, an external analyzer may be set up to supply a true signal to the rear panel BNC connector on the card cage when it detects a true condition in the target system. Then the analyzer can be set up to store

Glossary
Real-Time Execution

qualify a certain kind of state, but only when the arm signal from the BNC is true.

Real-Time Execution

refers to the emulator configuration in which commands that temporarily interrupt target program execution (for example, display/modify target memory or processor registers) are not allowed.

Relative Count

a count in the trace list count column that shows the count between the present displayed state and the state displayed immediately before it. Relative time count, for example, shows the elapsed time between the previous displayed state and the present state. Note that the count is between displayed states. If your trace list is inverse assembled and/or dequeued, several states may have been captured in memory between the present displayed state and the displayed state immediately before it.

Remote Configuration

the configuration in which an HP 64700 Series emulator is directly connected to a host computer via a single port. Commands are entered (typically from an interface program running on the host computer) and absolute code is downloaded into the emulator through that single port.

RS-232C

a standard serial interface used to connect computers and peripherals.

Sequencer

a state machine in the analyzer that searches for execution of states in a particular order.

Single-step

the execution of one microprocessor instruction. Single-stepping the emulator allows you to view program execution one instruction at a time.

Softkey Interface

the host computer interface program used in the UNIX environment. The Softkey Interface is a friendly interface used to control HP 64700 emulators.

Software Breakpoint

refer to execution breakpoint and access breakpoint in this glossary.

Software Performance Analyzer

an analyzer that measures execution of software modules, interaction between software modules, and usage of data points and I/O ports.

Standalone Configuration

the configuration in which a data terminal is used to control the HP 64700 Series emulator, and the emulator is not connected to a host computer.

stderr

an abbreviation for “standard error output.” Standard error can be directed to various output devices connected to the HP 64700 ports.

stdin

an abbreviation for “standard input.” Standard input is typically defined as your computer keyboard.

stdout

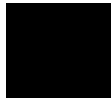
an abbreviation for “standard output.” Standard output can be directed to various output devices connected to the HP 64700 ports.

Step

See Single-step.

Store Qualifier

a specification that must be met by a state before it can be saved in the analyzer trace memory.



Synchronous Execution

the execution of multiple HP 64700 Series emulators/analyzers at the same time (i.e., multiple emulator start/stop).

Syntax

the order in which expressions are structured in command languages. Syntax rules determine which forms of command language syntax are grammatically acceptable.

Target Program

The program you are developing for your product. It is also called user program.

Target System

the circuitry where the emulator probe is connected (typically a microprocessor-based system under development).

Target System Memory

storage that is present in the target system.

Terminal Interface

the command interface present inside the HP 64700 Series emulators that is used when the emulator is connected to a simple data terminal. This interface provides on-line help, command recall, macros, and other features which provide for easy command entry from a terminal.

Trace

a collection of states captured synchronously by the analyzer.

Trigger

the condition that identifies a reference state within an analyzer trace measurement. Trigger also refers to the analyzer signal that becomes active when the trigger condition is found.

Trigger signals called trig1 and trig2 are bidirectional signal lines that can be used to coordinate measurement activity between emulators and

analyzers installed in the instrumentation card cage, and between instruments connected to the BNC on the rear panel of the card cage. For details of how to configure and use trig1 and trig2, refer to the chapter on making coordinated measurements.

Note that there is delay when you use trig1 and/or trig2 for measurement coordination. For example, you may specify that the emulator break to its monitor program when it receives trig1 from the analyzer. Several states may be executed in the emulator between the time the analyzer recognizes its trigger condition, generates trig1, delivers trig1 to the emulator, and the emulator responds to trig1 by breaking to its monitor program.

Uploading

the transfer of emulation or target system memory contents to a host computer.

Unlocked Exit

one of two methods used to leave the high level (Graphical or Softkey) Interface and return to the host computer operating system. An unlocked exit command allows you to exit the high level interface and re-enter later with the default configuration. (See also Locked Exit.) This is not available in the Terminal Interface.

User Program

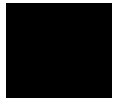
Another name for your target program (the program you are developing for your product).

Viewport

see Window.

Wait States

extra microprocessor clock cycles that increase the total time of a bus cycle. Wait states are typically used when slower memory is implemented.



Glossary
Window

Window

a specified rectangular area of virtual space shown on the display in which data can be observed.

1K Analyzer

the term "1K analyzer" refers to the HP 64704 Emulation-Bus Analyzer with 1K trace memory.

!

when shown in the trace list count column of the terminal interface or the PC interface, the exclamation mark "!" indicates counter overflow.

Index

- A** abbreviated help mode, **280**
- absolute count (in trace list), **354**
- absolute file
 - formats, **269, 284**
 - loading into memory, **284-286**
 - loading via ftp, **65-66**
- absolute, glossary definition of, **621**
- accent grave mark character, **320**
- access mode, **302, 304**
- access to guarded memory, **296**
- accuracy of trigger position, **379**
- active edges (slave clock), **393-394**
- activity, analyzer line, **331-332**
- addition operator, **423**
- address
 - mapping details of a single address, **220**
 - supervisor/user mappings in the MMU, **219**
 - translation details of a single address, **103**
- address, how to long align, **382**
- addresses
 - how they are affected when the MMU is on, **216**
 - logical vs physical explained, **210**
 - physical in trace list, things to check, **203**
- all (analyzer keyword), **356**
- altitude specifications, **515**
- analyzer
 - analyzer initialization, **370-371**
 - clock (master) specification, **150, 342-345**
 - complex configuration, **135, 339**
 - complex configuration pattern qualifier, **381-383**
 - complex configuration range qualifier, **386-388**
 - configuration, **335-341**
 - count qualifier, **338, 340, 346-348**
 - easy configuration, **337**
 - expressions, **425**



expressions in the complex configuration, **339**
halt trace, **364-365**
invalid simple expressions, **429**
labels, **376-377**
line activity, **331-332**
master clock specification, **342-345**
performance verification, **306-308**
prestore qualifier, **338, 340, 384-385**
primary branches (sequencer), **366-369**
secondary branches (sequencer), **349-352**
sequencer, **395-398**
sequencing in the complex configuration, **339**
sequencing in the easy configuration, **337**
slave clocks, **393-394**
start, **330**
state/time counter, **125**
storage qualifiers, **399-401**
storage specification in the complex configuration, **340**
storage specification in the easy configuration, **338**
trace configuration reset, **341**
trace list format, **353-355**
trace sequencer, **128**
tracing background operation, **343**
tracing foreground operation, **343**
trigger condition, **356-358**
trigger in feedback loop, **247, 262**
trigger output, **359-363**
trigger position, **378-380**
analyzer clock speed, glossary definition of, **622**
AND
 (bit-wise) operator, **424**
 (interset logical) operator, **417**
any (analyzer keyword), **356, 384**
arm condition
 analyzer status, **390**
 complex expressions, **416**
 cross-arming, **246, 262**
 specifying, **333-334**
 time until trigger, **391, 489, 494**
arming the analyzer, **333-334**
ASCII strings, displaying on standard output, **271**

architectures of virtual memory, **211-212**

1K analyzer

glossary definition of, **634**

- B**
- b (break) command, **80, 241**
- background operation, tracing, **343**
- bases (number), **420**
 - default for step count, **316**
 - labels in trace list, **353**
- baud rate, communication ports, **323**
- bc (break conditions) command, **242-244**
- binary
 - number base specifier, **411, 420**
 - trace list format, **373, 486**
- bit-wise operators
 - AND, **424**
 - exclusive OR, **424**
 - inclusive OR, **425**
 - merge, **425**
- BKPT (breakpoint vector)
 - generally, **88**
- block (memory mapper)
 - re-assignment of emulation memory, **297**
- BNC trigger signal, **243, 245-247**
- bnc (BNC trigger drivers and receivers) command, **245-247**
- bp (breakpoint modify) command, **92-93, 95-96, 248-252**
- branch not taken in trace list, what it means, **125**
- branch qualifiers (sequencer)
 - primary, **366-369**
 - secondary, **349-352**
- branch taken in trace list, what it means, **125**
- break, **241**
 - to monitor, **80**
 - to monitor on a trigger signal, **166**
- break conditions
 - BNC or CMB trigger signals, **243**
 - software breakpoints, **243**
 - trig1 or trig2 internal signals, **243**
 - write to ROM, **243**
- break in emulator due to trace complete, **167**
- breakpoint
 - to determine whether in software or hardware, **89**

breakpoints
a breakpoint is recognized where none was set, **89**
disabling, **95, 251**
displaying, **96**
enabling, **93, 251**
generally, **88**
inserting, **92, 250**
interlocking target system acknowledge cycles, **191**
removing, **96, 251**

breaks
guarded memory access, **296**
synchronous, **259**

bus cycle
hung bus cycle defined, **81**
bus cycles, slow, **279**

C cables
power, **602**
cables, connecting to the emulator probe, **581**
calculator for expressions, **272**
can't break into monitor example, **235-237**
card cage
applying power, **602**
cautions
antistatic precautions, **578**
apply power to emulator before target system, **99**
protect against static discharge, **98**
rear panel, do not stand HP 64700 on, **584**
turn OFF power before installing emulator probe, **98**
verify pin 1 when installing emulator probe, **98**
cf (emulator configuration) command, **186, 188, 253-256**
channels (analyzer)
demultiplexed slave clock mode, **393**
mixed slave clock mode, **393**
cl (command line control) command, **45, 257-258**
clocks
configuration, **150**
count states or time, **153**
generally, **149**
rate settings, **151**
specifications, **504**
specifying analyzer master, **342-345**

- specifying analyzer slave, **393-394**
- trace user/background code, **149**
- cmb (coord. meas. bus enable/disable) command, **259-260**
- CMB (Coordinated Measurement Bus)
 - enable/disable, **259**
 - specifications, **516**
 - start synchronous execution, **407**
 - trace at /EXECUTE, **402-403**
 - trigger signal, **243, 261-263, 402**
- cmbt (CMB trigger drivers/receivers) command, **261-263**
- column headers in trace list
 - adding new columns, **353**
 - suppressing, **373**
- command files
 - building of, **54**
 - building with a text editor, **55**
 - comments in, **55**
 - editing of, **55**
 - generally, **54**
 - log from a PC host, **56**
 - log on a UNIX host, **57**
 - using on a PC host, **58**
 - using on a UNIX host, **59**
- command line editing
 - commands, **45**
 - installing or removing, **45**
- command processing delays, **50**
- commands
 - b, **80**
 - cf, **255**
 - cp, **77**
 - cu, **58**
 - dmmu, **266-267**
 - echo, **49**
 - help, **46, 280-281**
 - help for group, **280**
 - init, **41**
 - mac, **51-53**
 - macros, **291-294**
 - maximum length of command line, **293**
 - mmu, **299-301**



Index

- rep, **50**
- repeating a group of, **312**
- rst, **84**
- s, **81**
- sym, **325-328**
- tsq, **54**
- w, **50**
- communications (data)
 - initialization, **282**
 - setting parameters, **322-324**
- compatible mode
 - glossary definition of, **623**
- complex analyzer configuration, **339**
- complex expressions, **140**
 - generally, **135**
 - pattern specifications, **381-383**
 - primary branches, **142**
 - range qualifiers, **139**
 - range specification, **386-388**
 - secondary branches, **144**
 - storage qualifiers, **145**
 - trace patterns, **138**
 - trace sequencer, **136-137**
 - trigger terms, **135**
- complex expressions, **418**
- configuration
 - analyzer, **335-341**
 - data communications switches, **323-324**
 - default, **255**
 - emulator, **253-256**
 - items, **254**
 - trace, **128**
- connecting probe to demo target system, **600**
- control (CTRL) characters
 - c, command abort, **307, 312, 316**
 - non-displaying, **271**
- Coordinated Measurement Bus (CMB)
 - generally, **160**
- coordinated measurements
 - enable/disable, **259-260**
- copy

- memory blocks, **77, 264**
- count (occurrence), **153, 337, 340, 356-357, 366, 391**
 - reset if secondary branch taken, **351**
- count (time), **153**
- count qualifier, **338, 340, 346-348**
- counter
 - overflow indication not seen in trace list, **206**
- counter overflow, glossary definition of, **624**
- counter, analyzer tag, **346**
- counts
 - controlled by external analyzer, **168**
 - negative counts in count column, **206**
- cp (copy memory) command, **77, 264**
- cross-triggering, **246, 259, 262**

- crosstriggering emulation-bus analyzers, **163-165**
- cu command, **58**

D

- data communications
 - configuration switches, **323-324**
 - initialization, **282**
 - setting port parameters, **322-324**
 - specifications, **516**
- data cycles
 - monitor access to target memory, **302**
- date
 - displaying emulation system, **47**
 - setting emulation system, **47, 268**
- decimal
 - number base specifier, **411, 420**
- deep analyzer
 - glossary definition of, **625**
- delimiters (string), **271-272, 320**
- delta time
 - binary/hexadecimal trace list, **495, 497**
- deMMUer
 - command options, **224**
 - detailed discussion, **223**
 - how it is loaded by the emulator, **225**
 - how to enable, **224**
 - how to load reverse translations, **224**
 - its reverse translation table, **228-230**

- keeping it up to date, **226**
 - out of resources, things to check, **202**
 - programming in static memory system, **126**
 - resource limitations, **228-230**
 - restrictions when using, **226-227**
 - seeing present reverse translations, **224**
 - deMMUer/MMU chapter, **209**
 - demo target system
 - connecting the emulator probe, **600**
 - DeMorgan's theorem, **418**
 - demultiplexed (slave clock) mode, **393**
 - depth of memory
 - how to obtain different depths, **579**
 - setting in terminal interface, **157**
 - dequeued trace, aligning opcodes/operands, **123**
 - dequeueing of trace lists, **122**
 - disassembled trace, aligning opcodes/operands, **123**
 - disassembly of trace lists, **122, 353, 373**
 - display
 - a single address mapped by MMU, **220**
 - date and time, **47**
 - mode, **304**
 - mode, definition, **302**
 - present MMU mappings, **217-218**
 - divide (integer) operator, **423**
 - dmmu (reversing MMU for analyzer) command, **266-267**
 - download
 - user programs, **284-286**
 - drivers and receivers
 - BNC trigger signal, **245-247**
 - CMB trigger signal, **261-263**
 - See also* trig1 and trig2 internal signals
 - dt (set or display system date/time) command, **47, 268**
 - dual-port emulation memory, **174**
 - dump (upload memory) command, **269-270**
 - dynamic virtual memory systems, **212**
- E**
- easy analyzer configuration, **337**
 - generally, **129**
 - global restart terms, **133**
 - primary branches, **132**
 - reset sequencer, **131**

- sequence terms, **130-131**
- echo (display to standard output) command, **49, 271-273, 420**
- edges (analyzer clock), rising, falling, both, **343**
- edges (analyzer slave clock), active, **393-394**
- emulation break, **241**
- emulation break on analyzer trace complete, **167**
- emulation memory map used by deMMUer, **229**
- emulation memory modules
 - installing, **596**
- emulation monitor, **170**
 - break command, **241**
 - breaks to the, **243**
 - cycles used to access target memory, **302**
 - execute after reset, **313**
 - foreground, loading, **286**
 - running in (emulator status), **279**
 - searching target memory, **320**
- emulation process steps, **38**
- emulation RAM, mapping address ranges, **295**
- emulation ROM, mapping address ranges, **295**
- emulator
 - configuration, generally, **62, 170**
 - displaying status information, **47**
 - error messages, **432**
 - how it loads the deMMUer, **225**
 - in-circuit use of, **97**
 - initialization, **282-283**
 - initialization options, **41**
 - performance verification, **306-308, 606**
 - powerup, **40**
 - probe precautions, **98**
 - prompt, changing the, **305**
 - restrict to real-time runs, **188**
 - status, **279**
- emulator probe
 - connecting the cables, **581**
 - connecting to demo target system, **600**
- enabling the MMU, **100, 214**
- entering commands
 - combining, **42**
 - options, **42**

- repeating, **43**
- equ (equate names to expressions) command, **155-156, 274-278**
- equates, **274-278**
 - defining, **155**
 - deleting, **156**
 - displaying, **156**
- eram, mapper parameter for emulation RAM, **296**
- erom, mapper parameter for emulation ROM, **296**
- error messages, **432**
 - analyzer, **472**
 - emulator, **431-484**
- es (emulator status) command, **47, 279**
- escape (ESC) characters
 - j, edit existing command, **45**
 - k, recall existing command, **45**
- example, can't break into monitor, **235-237**
- exchange part, defined, **611**
- exclamation mark "!"
 - in trace list count column, **205**
- exclusive OR (bit-wise) operator, **424**
- EXECUTE (CMB signal), **260, 390, 402, 407**
- execution breakpoints
 - displaying, **96**
- expression calculator, **272**
- expression, glossary definition of, **626**
- expressions
 - analyzer, complex configuration, **339, 418**
 - analyzer, easy configuration, **337**
 - creating, **108**
 - creating complex expressions, **140**
 - creating, in easy configuration, **129**
 - equating names to, **274-278**
 - operators, **422**
- external timing analyzer
 - glitch mode, **495, 498**
 - standard mode, **495, 498**
 - transitional mode, **495, 497**
- F**
 - fast (F) analyzer clock speed, **341, 344**
 - file formats
 - absolute, **269, 284**
 - Intel hex and Tektronix hex, **63**

- Motorola S-record, **63**
- firmware update utility, installation, **617**
- foreground monitor, **182**
 - mapping 1:1 for use when MMU enabled, **192-194**
 - set interrupt priority, **184**
- foreground operation, tracing, **343**
- formats
 - absolute file, **269, 284**
 - binary trace list, **373, 486**
 - hexadecimal trace list, **373, 486**
 - memory display, **287**
 - trace list, **353-355**
- FPU
 - used with MC68EC040 and MC68LC040, **104**
- ftp
 - loading absolute files, **65-66**
 - loading symbol files, **68-69**
- function codes, **289**
- function codes used in translation tables, **214**
- G**
 - glitch (external timing analyzer) mode, **495, 498**
 - global
 - access and display modes, **302**
 - restart qualifier, **337, 349, 357, 368, 395**
 - storage qualifier, **338, 400**
 - grave mark character, **320**
 - grd, mapper parameter for guarded memory, **296**
 - group (command), **280**
 - guarded memory access, **296**
 - guarded memory access when using MMU, **233**
- H**
 - H,h, hexadecimal number base specifier, **421**
 - halt
 - emulation status, **279**
 - trace, **364-365**
 - trace status, **390**
 - halt of system when using MMU, **234**
 - handshaking (data communications), **323**
 - hardware
 - HP 9000 minimums overview, **576**
 - hardware enable for the MMU, **214**
 - headers in trace list

- adding new columns, **353**
- suppressing, **373**
- help
 - abbreviated mode, **280**
 - on-line help command, **46, 280-281**
- hexadecimal
 - base specifier, **421**
 - number base specifier, **411**
 - trace list format, **373, 486**
- history, trace status, **391**
- HP 9000
 - minimum system requirements overview, **576**
- humidity specifications, **515**
- hung bus cycle, **81**

- I**
 - ##IL#** in trace list Mnemonic column, **484**
 - inclusive OR (bit-wise) operator, **425**
 - information (help), **280-281**
 - init (initialize the emulator) command, **41, 282-283**
 - initialization
 - analyzer, **370-371**
 - emulator, **282-283**
 - installation, **576**
 - hardware, **577-610**
 - of optional memory modules, **579**
 - placing boards in the card cage, **584**
 - verifying installation of memory modules, **607**
 - interlock cycle termination signals, **183**
 - internal signals, trig1 and trig2, **243, 245, 261, 333, 364, 402**
 - interrupt stack pointer presetting, **186**
 - interset operators, **417**
 - intraset operators, **416**
 - inverse values (complex analyzer expressions), **418**
 - IP address, **65**

- J** J clock (analyzer), **344**

- K** K clock (analyzer), **344**

- L** L clock (analyzer), **344**
 - label, glossary definition of, **627**
 - labels (trace)
 - defining analyzer, **113, 376-377**

- deleting analyzer, **114**
 - displaying analyzer, **114**
 - predefined, **376**
 - LAN connection
 - loading absolute files, **65-66**
 - loading symbol files, **68-69**
 - line activity (analyzer), **331-332**
 - list of replaceable parts, **611-613**
 - listing the present MMU mappings, **217-218**
 - load (download user programs) command, **284-286**
 - logical address
 - definition of, **211**
 - viewing the translation details for, **103**
 - logical operators
 - See* operators
 - logical-to-physical mappings, to view, **101-102**
 - logical-to-physical translation (mmu command), **299-301**
 - long-aligned address, how to create, **382**
- M**
- m (memory display/modify) command, **287-290**
 - displaying options, **72**
 - modifying options, **73**
 - search options, **75**
 - M clock (analyzer), **344**
 - mac (macro definition/display) command, **51-53, 291-294**
 - macros, **51**
 - creation of, **51**
 - deletion of, **53**
 - execution of, **52**
 - limitations, **293**
 - list predefined, **51**
 - map (memory mapper) command, **172, 177-178, 295-298**
 - mapping memory, **295-298**
 - mappings, logical-to-physical, to view, **101-102**
 - master clocks (analyzer), **342-345**
 - maximum
 - analyzer clock speed, **343**
 - command line length, **293**
 - mapper terms, **298**
 - sequence levels in easy configuration, **368**
 - sequence terms in easy configuration, **398**
 - software breakpoints, **252**

- MC68EC040 and MC68LC040
 - performance measurements of FPU instructions, **104**
 - special considerations when including an FPU, **104**
 - testing floating-point libraries, **104**
- measurements
 - analyzer, starting, **330**
 - coordinated, **259-260**
- memory, **170, 172**
 - accessing resources, **72**
 - assess mode, **302-304**
 - assign default map, **177**
 - assign map terms, **172**
 - copy blocks of, **77**
 - delete map terms, **178**
 - display mode, **302-304**
 - displaying, **72, 287-290**
 - dual-port, **174**
 - enable one wait state, **178**
 - enable/disable cache, **175**
 - loading programs into, **284-286**
 - mapping, **295-298**
 - modifying, **73, 287-290**
 - preventing storage of sequencer-advance state, **147**
 - processor cache disabling, **189**
 - search, **75, 319-321**
 - sockets on probe, **174**
 - type (list of), **172**
 - upload to host file, **269-270**
- memory depth
 - setting in terminal interface, **157**
- memory management, **209**
- memory management systems supported, **211-212**
- memory map, how it is used by deMMUer, **229**
- memory modules
 - installing on the emulation probe, **596**
 - verifying installation, **607**
- memory modules, how to install, **579**
- merge (bit-wise) operator, **425**
- mixed (slave clock) mode, **393**
- MMU
 - enabled, how it affects the analyzer, **126**

- enabled, using the emulator with, **100**
- how it affects command composition, **216**
- how it is enabled, **214**
- mapping details of a single address, **220**
- mapping monitor 1:1 when MMU enabled, **192-194**
- mappings, how the emulator obtains them, **218**
- mappings, listing the present mappings, **217-218**
- mappings, modifying for monitor, **235-237**
- mappings, obtaining a shorter list of, **218**
- restrictions when using, **215**
- reversing its translations, **266-267**
- special problems discussion, **232**
- where is it located, **213**
- mmu (logical-to-physical translation) command, **299-301**
- MMU/deMMUer chapter, **209**
- mnemonic
 - information in the trace list, **353**
 - memory display mode, **287**
- mnemonic column shows ##IL# notation, **484**
- mo (set access and display modes) command, **302-304**
 - options, **78**
- mode
 - abbreviated help, **280**
 - access, **304**
 - demultiplexed slave clock, **393**
 - display, **304**
 - glitch (external timing analyzer), **495, 498**
 - memory access, **302-304**
 - memory display, **302-304**
 - mixed slave clock, **393**
 - quiet, **285, 316**
 - standard (external timing analyzer), **495, 498**
 - transitional (external timing analyzer), **495, 497**
 - whisper, **316, 389**
- modulo (integer) operator, **423**
- monitor
 - to map 1:1 for use with an enabled MMU, **192-194**
- monitor (emulation), **170, 180**
 - break command, **241**
 - breaks to the, **243**

- configuration of MC68040, **181**
 - cycles used to access target memory, **302**
 - execute after reset, **313**
 - foreground, loading, **286**
 - keep-alive address (MC68EC030), **185**
 - running in (emulator status), **279**
 - searching target memory, **320**
 - set the base address, **182**
 - multiple traces, numbering, **277**
 - multiply (integer) operator, **423**
- N**
- N clock (analyzer), **344**
 - names
 - pattern, **381**
 - values, **274-278**
 - NAND operator, **418**
 - negative counts in trace list count column, **206**
 - never (analyzer keyword), **356**
 - No trace data (message), **375**
 - none (analyzer keyword), **348, 356, 384**
 - NOR (intra-set logical) operator, **416**
 - notation ##IL# in trace list, **484**
 - notes
 - access mode for writing breakpoints, **248**
 - address followed by two periods as a range, **287**
 - address specification, **289, 309**
 - analyzer should not drive and receive same signal, **247, 262**
 - analyzer, "tcq time" only if "tck -s S", **346**
 - arm to trigger time alignment between emulators, **489**
 - asterisk (*) in help command, **280**
 - breakpoint display status checking, **251**
 - date and time are reset when power is cycled, **268**
 - date assumes year is in 20th century, **268**
 - display mode and memory modification, **288**
 - don't care values are not allowed in echo command, **421**
 - emulation memory block re-assignment, **297**
 - equate limits, **274**
 - equates, new values not updated in commands, **277**
 - equates, predefined, **274**
 - init -c or -p cause system memory loss, **282**
 - macros allowed within rep commands, **312**

- macros, predefined, **291**
 - map change requires breakpoint disable, **298**
 - master clock qualifiers: tck -u, tck -b, **343**
 - memory display is not updated, **288**
 - memory map modification causes emulator reset, **298**
 - occurrence counts in complex configuration, **367**
 - operations are on thirty-two-bit signed integers, **272**
 - primary and secondary branch qualifiers satisfied, **351, 368**
 - pv command reinitializes emulator, **306**
 - range not allowed in pattern specifications, **381**
 - range reset when trace configuration reset to easy, **387**
 - run from reset function varies with emulators, **309**
 - rx command enables CMB interaction, **260**
 - search patterns, specifying complex, **320**
 - sequence term count reset, **351**
 - sequencer term 8 default, **350, 368**
 - single open quote, ASCII character, **271, 320**
 - step count must be specified with address, **318**
 - step does not work correctly while CMB enabled, **318**
 - storage qualifiers and the sequencer, **400**
 - storage qualifiers, global, **399**
 - string delimiter character should not be in string, **271**
 - strings should not contain string delimiter character, **320**
 - trace format does not affect information captured, **355**
 - trace list command options, mutually exclusive, **373**
 - trace list from a specific state, **374**
 - trace states, displaying when trigger not found, **364**
 - trig1 and trig2 can both drive/receive BNC, **245**
 - xon toggling with baud rates of 1200 or below, **324**
 - numbering multiple traces, **277**
 - numbers, software version, **404**
 - numeric calculator, **49**
 - numeric search in memory, **319**
- O**
- O,o, octal number base specifier, **420**
 - occurrence count, **337, 340, 356-357, 366, 391**
 - reset if secondary branch taken, **351**
 - octal
 - number base specifier, **411**
 - octal number base specifier, **420**
 - one's complement (unary) operator, **423**
 - opcodes and operands, how to align in trace, **123**

- operators, **422**
 - | (intraset OR), **140**
 - ~ (intraset NOR), **140**
 - AND (interaset AND), **140**
 - combining intraset and interaset, **418**
 - interaset, **417**
 - intraset, **416**
 - OR (interaset OR), **140**
 - precedence, **422**
- OR operator
 - bit-wise, **425**
 - interaset logical, **417**
 - intraset logical, **416**
 - restriction in easy configuration, **429**
- other, mapper parameter for unmapped memory, **296**
- out of deMMUer resources
 - how to avoid this message, **230**
 - things to check, **202**
- overflow, glossary definition of, **628**
- overlap
 - bit-wise merge, **425**
 - trace labels, **377**
- P**
 - p1 - p8, trace pattern labels, **381**
 - parameters, data communications, **322-324**
 - parts list, **611-613**
 - pattern
 - expressions, **420**
 - labels, **416**
 - names, **381**
 - qualifier (complex analyzer config.), **381-383**
 - performance verification, **306-308**
 - physical address
 - definition of, **211**
 - in trace list, things to check, **203**
 - tracing execution in physical space, **127**
 - physical-logical mappings, to view, **101-102**
 - pipeline
 - analyzer architecture, **390**
 - analyzer prestore, **338**
 - po (specify port control) command, **48, 305**
 - polarity, trace labels, **376**

ports (data communications)
 setting parameters, **322-324**

position of trigger state in trace, **378-380**

power applied to the card cage, **602**

power cables
 connecting, **602**
 correct type, **602**

powerup, **40, 99**
 initialization sequence, **282**

precedence, operator, **422**

predefined macros, **291**

predefined trace labels, **376**

prestore qualifier, **338, 340, 384-385**
 defining, **152**

prestore qualifier, glossary definition of, **629**

prevent storage of sequencer-advance states, **147**

primary branches (analyzer sequencer), **142, 366-369**
 define in easy configuration, **132**

print
 to standard output device, **49**

probe
 connecting the cables, **581**
 connecting to demo target system, **600**
 dimensions, **514**
 emulator, **306**
 memory sockets, **174**

problem solving
 can't break to monitor after MMU enabled, **205**
 if deMMUer is out of resources during loading, **202**
 if only physical addresses in trace list, **203**
 if out of resources with less than 8 mappings, **203**
 if the analyzer won't trigger, **199**
 if the demo program won't work, **30**
 if you have trouble mapping memory, **200**
 if you see unexplained states in the trace list, **198**
 if you suspect the emulator is broken, **200**
 if you're having problems with DMA, **201**
 if you're having problems with emulation reset, **202**

problems, a discussion for the MMU, **232**

processor
 cache memories, **171**

cache memory generally, **189**
disable cache memory, **189**
reset from emulator, **84**
run controls, **79**
progflash example, **619**
program counter presetting, **186**
program counter symbol (\$), **309**
programs
 building, **62**
 loading from a PC host, **64**
 loading from a UNIX host, **67**
 simultaneous run on two emulators, **162**
prompt (emulator)
 changing, **305**
 changing , **48**
protocol (transfer), **270, 285, 373, 486**
protocol checking, **285**
pv (performance verification) command, **306-308**

- Q** Q,q, octal number base specifier, **420**
qualifier, glossary definition of, **629**
qualifiers
 analyzer count, **346-348**
 analyzer master clock, **342-345**
 analyzer pattern, **381-383**
 analyzer prestore, **152, 384-385**
 analyzer range, **386-388**
 analyzer storage, **399-401**
 complex storage, **145**
 defining range patterns, **139**
 global restart, **337, 349, 357, 368, 395**
 sequencer primary branch, **366-369**
 sequencer secondary branch, **349-352**
question mark (?)
 break conditions display, **244**
 on-line help command, **280**
quick reference, **46**
quiet mode, **285, 316**
quote marks, **271, 305, 320**
- R** r (run user program) command, **309**
 options, **79**

radix indicators
 binary, **108**
 decimal, **108**
 hexadecimal, **108**
 octal, **108**

range qualifier (complex analyzer config.), **386-388**

ranges, **416**

READY (CMB signal), **260, 407**

recalling commands, **43**
 in reverse, **43**

receivers and drivers
 BNC trigger signal, **245-247**
 CMB trigger signal, **261-263**
 See also trig1 and trig2 internal signals

record checking, **270**

reg (register display/modify) command, **85, 310-311**

registers
 displaying, **85**
 displaying multiple, **86**
 introduction, **85**
 modifying, **86**
 modifying multiple, **87**

relational expressions, **416-417**

relational operators, **349, 367, 400**

relative counts in trace list, **347, 354**

relative, glossary definition of, **630**

rep (repeat commands) command, **44, 50, 312**
 canceling, **50**

reset
 break during, **241**
 breakpoints, **249**
 emulation microprocessor, **313**
 emulator, due to mapper modification, **298**
 init command, **283**
 occurrence count, **351**
 range qualifier and trace configuration, **387**
 run from, **309**
 sequencer, **395**
 system date and time, **268**
 trace configuration, **341**
 trace specification, **370-371**

trace tag counter, **348**
restart (global) qualifier, **337, 349, 357, 368, 395**
reverse translations of MMU (dmmu command), **266-267**
ROM memory changed by processor writes, **91**
ROM, break on writes to, **243**
rotate left/right operator, **423**
RS-422, serial port data communications, **323**
rst (reset emulation processor) command, **313**
options, **84**
run
from a specific address, **79**
from current program counter, **79**
from target system reset, **79**
restrict to real-time, **188**
simultaneously on two emulators, **162**
synchronous, **259**
run from reset command, **171**

S **s (step the emulation processor) command, 316-318**
options, **81**
sample period (external timing analyzer), **495**
secondary branches (analyzer sequencer), **144**
branch default, **144**
semicolon (command separator), **291, 312**
sequence terms
deleting, **396**
inserting, **395**
inserting using easy configuration, **130**
removing using easy configuration, **131**
reset in easy configuration, **131**
sequencer
preventing states from being stored in memory, **147**
preventing storage of sequencer-advance state, **147**
trace start with active term other than term1, **134, 148**
sequencer (analyzer), **395-398**
complex configuration, **339**
define global restart term, **133**
display current settings, **134**
displaying settings in complex configuration, **137**
easy configuration, **337**
primary branches, **366-369**
resetting in complex configuration, **136**

- secondary branch qualifiers, **349-352**
- ser (search memory for values) command, **319-321**
- sets (complex config. trace spec.), **416**
- shift left/right operator, **423**
- short help, **281**
- signals
 - analyzer clocks, **344, 394**
 - analyzer, defining labels for, **376-377**
 - arm, **391**
 - BNC trigger, **243, 245-247**
 - CDIS, **189**
 - CMB /EXECUTE, **260, 313, 390, 402, 407**
 - CMB READY, **260, 407**
 - CMB trigger, **243, 261-263**
 - EXECUTE, **160**
 - internal trig1 and trig2, **243, 333, 364, 402**
 - READY, **160**
 - TA and TEA, **170, 173, 183**
 - TCI, **173**
 - TRIG1, **161**
 - TRIG2, **161**
 - TRIGGER, **160**
 - trigger output, **359-363**
- simple expressions, **129**
- single-step, **81**
- single-step emulation processor, **316-318**
- slave clocks (analyzer), **393-394**
 - demultiplexed mode, **393**
 - mixed mode, **393**
- slow (S) analyzer clock speed, **344, 346**
- slow clock emulator status, **279**
- software
 - ensuring compatibility, **609**
- software breakpoints, **248-252**
 - break condition enable/disable, **243**
 - disabling, **95, 251**
 - displaying, **96**
 - enabling, **93, 251**
 - inserting, **92, 250**
 - maximum number of, **252**
 - pv command effect on, **306**

removing, **96, 251**
software enable for the MMU, **215**
software version numbers, **404**
specifications
altitude, **515**
clock, **504**
CMB, **516**
data communications, **516**
humidity, **515**
probe dimensions, **514**
temperature, **515**
trigger in/out, **515**
weight, **514**
standard (external timing analyzer) mode, **495, 498**
standard command prompt, changing, **48**
states (trace)
maximum with/without count, **348**
prestore, **384**
status, **391**
visible, **391**
static discharge, protecting the emulator probe against, **98**
static memory system, loadig deMMUer, **126**
static virtual memory system, **211**
status
analyzer, **389-392**
emulator, **279**
step, **81**
storage (trace) specification, **399-401**
complex configuration, **340**
easy configuration, **338**
storage qualifier, **111**
store qualifier, glossary definition of, **631**
string delimiters, **271-272, 320**
string search in memory, **320**
stty (set data communications parameters) command, **322-324**
subtraction operator, **423**
switches, data communications configuration, **323-324**
sym (symbol) command, **325-328**
deleting options, **70**
displaying options, **71**
symbol file

- loading via ftp, **68-69**
 - symbol names, creating, **274-278**
 - symbols, **68**
 - ", character string delimiter, **75**
 - \$, program counter, **309**
 - &&, bit-wise merge, **425**
 - *, trace status, **392**
 - @, function code, **411**
 - \, hex codes, **49**
 - ‘, character string delimiter, **75**
 - |, intraset OR, **140, 350, 367, 400**
 - ~, intraset NOR, **140**
 - adding user symbols, **70**
 - displaying, **71**
 - removing, **70**
 - synchronous emulator execution, **407**
 - synchronous runs and breaks, **259**
 - syntax diagrams
 - address variable, **411**
 - system clock, **268, 306**
 - system date/time, **268**
 - system requirements
 - HP 9000 overview, **576**
 - systems, virtual memory explained, **211-212**
- T**
- t (start trace) command, **109, 330**
 - T,t, decimal number base specifier, **420**
 - ta (trace activity display) command, **155, 331-332**
 - table details for a single logical address, **103-105**
 - tag counter (analyzer), **346**
 - ?taken? in trace list, what it means, **124**
 - target system
 - mapping RAM address ranges, **295**
 - mapping ROM address ranges, **295**
 - target system interrupts, disabling
 - interrupts, to disable, **190**
 - tarm (specify arm condition) command, **333-334**
 - tcf (set easy/complex configuration) command, **128, 335-341**
 - tck (specify master clock) command, **149-150, 342-345**
 - tcq (specify count qualifier) command, **125, 153, 346-348**
 - telif (specify secondary branch qualifiers) command, **133, 144, 349-352**

- temperature specifications, **515**
- term other than term 1 active at trace start
 - how to specify, **134, 148**
- terminal interface prompts, **39**
- terms
 - analyzer sequencer, **337, 339, 398**
 - memory mapper, **297**
- tf (specify trace list format) command, **115, 353-355**
- tg (specify trigger condition) command, **356-358**
- tgout (specify signal driven on trigger) command, **359-363**
- th (trace halt) command, **109, 364-365**
 - listing traces, **375**
- tif (specify primary branch qualifiers) command, **132, 142, 366-369**
- time
 - analyzer keyword, **348**
 - display, **47**
 - negative times shown in count column, **206**
 - setting system, **47, 268**
- tinit (trace initialization) command, **370-371**
- tl (trace list display) command, **110, 119-120, 122**
 - options, **117**
- tlb (define labels for analyzer lines) command, **113-114, 376-377**
- tp (trigger position in trace list) command, **378-380**
 - options, **112**
- tpat (complex config. trace patterns) command, **138, 381-383**
- tpq (specify prestore qualifier) command, **152, 384-385**
- trace
 - check signal activity, **155**
 - check user/background code execution, **149**
 - count controlled by external analyzer, **168**
 - display list, **110**
 - display status, **110**
 - pattern defining, **138**
 - preventing storage of sequencer-advance state, **147**
 - signal activity, **155**
 - start measurement, **109**
 - status, **389-392**
 - stop measurement, **109**
- trace configuration
 - easy/complex, **128**
 - reset, **341**

trace labels, **376-377**
 defining, **113**
 deleting, **114**
 displaying, **114**
 predefined, **376**

trace list
 depth, **125**
 disassembly and dequeuing, **122**
 display, **117**
 display of symbols and addresses, **120**
 header suppression, **119, 373**

trace list contains ##IL# notation, **484**

trace list format, **353-355**
 binary/hexadecimal, **486**
 mnemonics, **115**
 modifying, **115**
 relative/absolute, **115**

tram, mapper parameter for target RAM, **296**

transfer memory to host file, **269-270**

transfer, HP 64000 utility, **270, 285, 373, 486**

transitional (external timing analyzer) mode, **495, 497**

translating logical-to-physical (mmu command), **299-301**

translation
 details of a single logical address, **103**
 of a single address through the MMU, **220**
 reversing with the dmmu command, **266-267**
 table details for a single logical address, **103-105**

trig1 and trig2 internal signals, **243, 245, 261, 333, 364, 402**

trigger
 condition, **356-358**
 cross-triggering, **259**
 in/out specifications, **515**
 "not in memory" message, **375**
 position, **112, 378-380**

trigger one analyzer with another, **163-165**

trigger qualifier, **110**

trigger terms
 assigning using complex configuration, **135**

trng (specify complex config. range) command, **139, 386-388**

trom, mapper parameter for target ROM, **296**

troubleshooting, **608**

truth tables for logical operators, **422**
ts (display trace status) command, **110, 389-392**
tsck (specify slave clocks) command, **393-394**
tsq (manipulate trace sequencer) command, **54, 130-131, 134-137, 395-398**
tsto (specify trace storage qualifier) command, **111, 145, 399-401**
two's complement (unary) operator, **423**
tx (trace on CMB /EXECUTE) command, **402-403**

U 219

unary ones's complement operator, **423**
unary two's complement operator, **423**
undefined breakpoint error, **252**
undefined software breakpoint when using MMU, **234**
upload memory to host, **269-270**

V

value expressions, **420**
values, equating with names, **274-278**
variant records, **497**
ver (display software version numbers) command, **49, 404**
verifying installation of memory modules, **607**
verifying performance, **306-308**
version checking, **49**
very fast (VF) analyzer clock speed, **341, 344, 346**
virtual memory (mmu command), **299-301**
virtual memory management, **209**

W

w (wait for specified event) command, **50, 405-406**
wait (in command sequence), **405-406**
wait state
 enable for synchronous/burst accesses, **178**
warnings, power must be OFF during installation, **584**
weight specifications, **514**
whisper mode, **316, 389**
write instructions that change ROM memory, **91**
write to emulation ROM, **296**
write to target ROM, **296**

X

x (start synchronous CMB execution) command, **407**
XOR (bit-wise) operator, **424**

Y

Y,y, binary number base specifier, **420**

Certification and Warranty

Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

Warranty

This Hewlett-Packard system product is warranted against defects in materials and workmanship for a period of 90 days from date of installation. During the warranty period, HP will, at its option, either repair or replace products which prove to be defective.

Warranty service of this product will be performed at Buyer's facility at no charge within HP service travel areas. Outside HP service travel areas, warranty service will be performed at Buyer's facility only upon HP's prior agreement and Buyer shall pay HP's round trip travel expenses. In all other cases, products must be returned to a service facility designated by HP.

For products returned to HP for warranty service, Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country. HP warrants that its software and firmware designated by HP for use with an instrument will execute its programming instructions when properly installed on that instrument. HP does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environment specifications for the product, or improper site preparation or maintenance.

No other warranty is expressed or implied. HP specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.

Exclusive Remedies

The remedies provided herein are buyer's sole and exclusive remedies. HP shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.

Safety

Summary of Safe Procedures

The following general safety precautions must be observed during all phases of operation, service, and repair of this instrument. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the instrument. Hewlett-Packard Company assumes no liability for the customer's failure to comply with these requirements.

Ground The Instrument

To minimize shock hazard, the instrument chassis and cabinet must be connected to an electrical ground. The instrument is equipped with a three-conductor ac power cable. The power cable must either be plugged into an approved three-contact electrical outlet. The power jack and mating plug of the power cable meet International Electrotechnical Commission (IEC) safety standards.

Do Not Operate In An Explosive Atmosphere

Do not operate the instrument in the presence of flammable gases or fumes. Operation of any electrical instrument in such an environment constitutes a definite safety hazard.

Keep Away From Live Circuits

Operating personnel must not remove instrument covers. Component replacement and internal adjustments must be made by qualified maintenance personnel. Do not replace components with the power cable connected. Under certain conditions, dangerous voltages may exist even with the power cable removed. To avoid injuries, always disconnect power and discharge circuits before touching them.

Designed to Meet Requirements of IEC Publication 348

This apparatus has been designed and tested in accordance with IEC Publication 348, safety requirements for electronic measuring apparatus, and has been supplied in a safe condition. The present instruction manual contains some information and warnings which have to be followed by the user to ensure safe operation and to retain the apparatus in safe condition.

Do Not Service Or Adjust Alone

Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

Do Not Substitute Parts Or Modify Instrument

Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification of the instrument. Return the instrument to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.

Dangerous Procedure Warnings

Warnings, such as the example below, precede potentially dangerous procedures throughout this manual. Instructions contained in the warnings must be followed.

Warning

Dangerous voltages, capable of causing death, are present in this instrument. Use extreme caution when handling, testing, and adjusting.

Safety Symbols Used In Manuals

The following is a list of general definitions of safety symbols used on equipment or in manuals:



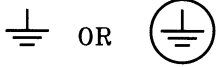
Instruction manual symbol: the product is marked with this symbol when it is necessary for the user to refer to the instruction manual in order to protect against damage to the instrument.



Hot Surface. This symbol means the part or surface is hot and should not be touched.



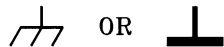
Indicates dangerous voltage (terminals fed from the interior by voltage exceeding 1000 volts must be marked with this symbol).



Protective conductor terminal. For protection against electrical shock in case of a fault. Used with field wiring terminals to indicate the terminal which must be connected to ground before operating the equipment.



Low-noise or noiseless, clean ground (earth) terminal. Used for a signal common, as well as providing protection against electrical shock in case of a fault. A terminal marked with this symbol must be connected to ground in the manner described in the installation (operating) manual before operating the equipment.



Frame or chassis terminal. A connection to the frame (chassis) of the equipment which normally includes all exposed metal structures.



Alternating current (power line).



Direct current (power line).



Alternating or direct current (power line).

Caution

The Caution sign denotes a hazard. It calls your attention to an operating procedure, practice, condition, or similar situation, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product.

Warning

The Warning sign denotes a hazard. It calls your attention to a procedure, practice, condition or the like, which, if not correctly performed, could result in injury or death to personnel.