HP 64752

70732 Emulator Terminal Interface

User's Guide

HP Part No. 64752-97002
Printed in U.S.A.
July 1994

Edition 2

Notice

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

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

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

Printing History

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

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

Edition 1 64752-97000, August 1993

Edition 2 64752-97002, July 1994

Using this Manual

This manual will show you how to use HP 64752A emulators with the Terminal Interface.

This manual will:

- Show you how to use emulation commands by executing them on a sample program and describing their results.
- Show you how to configure the emulator for your development needs. Topics include: restricting the emulator to real-time execution, selecting a target system clock source, and allowing the target system to insert wait states.
- Show you how to use the emulator in-circuit (connected to a demo board/target system).
- Describe the command syntax which is specific to the 70732 emulator.

This manual will not:

- Describe every available option to the emulation commands; this is done in the *HP 64700 Emulators Terminal Interface: User's Reference*.

Organization

Chapter 1 **Introduction to the 70732 Emulator.** This chapter briefly introduces you to the concept of emulation and lists the basic features of the 70732 emulator.

Chapter 2 **Getting Started.** This chapter shows you how to use emulation commands by executing them on a sample program. This chapter describes the sample program and how to: load programs into the emulator, map memory, display and modify memory, display registers, step through programs, run programs, use software breakpoints, search memory for data, and perform coverage tests on emulation memory.

Chapter 3 **Using the Emulator**. This chapter shows you how to: restrict the emulator to real-time execution, use the analyzer, use the instruction cache, and run the emulator from target system reset.

Chapter 4 **In-Circuit Emulation Topics**. This chapter shows you how to: install the emulator probe into a demo board/target system, allow the target system to insert wait states, and use the features which allow you to debug target system ROM.

Appendix A **70732 Emulator Specific Command Syntax**. This appendix describes the command syntax which is specific to the 70732 emulator. Included are: emulator configuration items, display and access modes, register class and name.

Appendix B **Using the Optional Foreground Monitor**. This appendix describes how to use the foreground monitor.

Contents

Illustrations

Tables

1

Introduction to the 70732 Emulator

Introduction

The topics in this chapter include:

- Purpose of the emulator
- Features of the emulator
- Limitations and Restrictions of the emulator

Purpose of the Emulator

The 70732 emulator is designed to replace the 70732 microprocessor in your target system to help you debug/integrate target system software and hardware. The emulator performs just like the processor which it replaces, but at the same time, it gives you information about the bus cycle operation of the processor. The emulator gives you control over target system execution and allows you to view or modify the contents of processor registers, target system memory, and I/O resources.

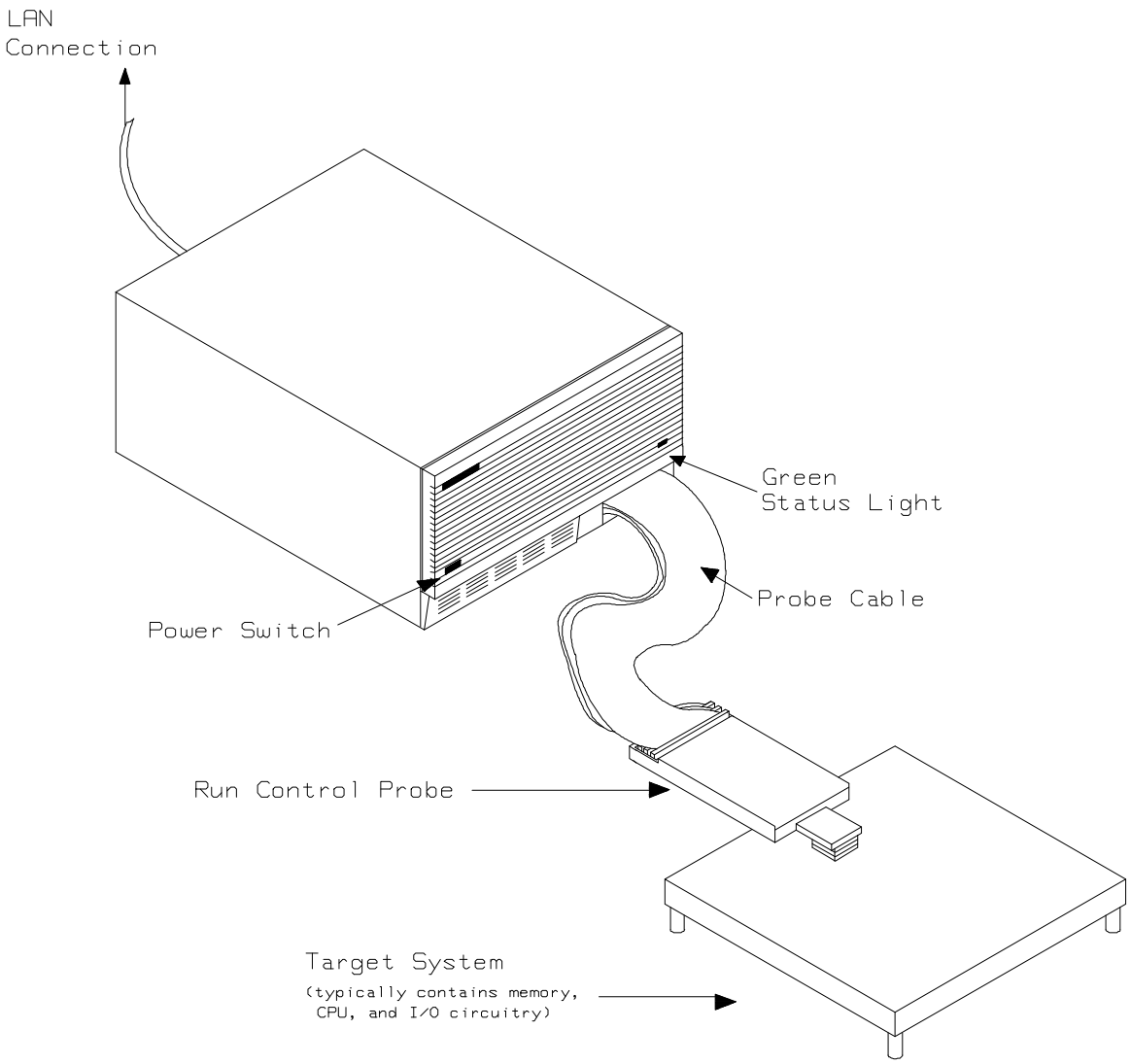


Figure 1-1 HP 64752A Emulatoor for uPD70732

1-2 Intruduction

Features of the 70732 Emulator

This section introduces you to the features of the emulator. The chapters which follow show you how to use these features.

Supported Microprocessors

The 176-pin PGA type of 70732 microprocessor is supported. The HP 64752A emulator probe has a 176-pin PGA connector. When you use 120-pin QFP type microprocessor, you must use with PGA to QFP adapter; refer to the "In-Circuit Emulation Topics" chapter in this manual.

Clock Speeds

The 70732 emulator runs with a target system clock from 8 to 25 MHz.

Emulation memory

The HP 64752A emulator is used with one or two of the following Emulation Memory Module.

- HP 64171A 256K byte Emulation Memory Module(35 ns)
- HP 64171B 1M byte Emulation Memory Module(35 ns)
- HP 64172A 256K byte Emulation Memory Module(20 ns)
- HP 64172B 1M byte Emulation Memory Module(20 ns)

You can define up to 16 memory ranges (at 4K byte boundaries and at least 4k byte in length). You can characterize memory ranges as emulation RAM, emulation ROM, target system RAM, target system ROM, or guarded memory. HP 64172A/B can be accessed with no wait. HP64171A/B can be accessed with no wait when clock speed is less than or equal to 20 MHz, and with one wait when clock speed is greater than 20 MHz. The emulator generates an error message when accesses are made to guarded memory locations. You can also configure the emulator so that writes to memory defined as ROM cause emulator execution to break out of target program execution.

Analysis

The HP 64752A emulator is used with one of the following analyzers which allows you to trace code execution and processor activity.

- HP64704A 80-channel Emulation Bus Analyzer
- HP64794A/C/D Deep Emulation Bus Analyzer

The Emulation Bus Analyzer monitors the emulation processor using an internal analysis bus.

The emulator can dequeu in real-time when analyzer trace execution states and bus states.

The emulator can trace all bus cycles in real-time when analyzer trace only actual bus states.

Registers

You can display or modify the 70732 internal register contents.

Single-Step

You can direct the emulation processor to execute a single instruction or a specified number of instructions.

Breakpoints

You can set up the emulator/analyzer interaction so that when the analyzer finds a specific state, emulator execution will break to the emulation monitor.

You can also define software breakpoints in your program. The emulator uses the BRKRET instruction to provide software breakpoint. When you define a software breakpoint, the emulator places a BRKRET instruction at the specified address; after the BRKRET instruction causes emulator execution to break out of your program, the emulator replaces BRKRET with the original opcode.

Reset Support

The emulator can be reset from the emulation system under your control, or your target system can reset the emulation processor.

Configurable Target System Interface

You can configure the emulator so that it honors target system wait requests when accessing emulation memory.

Foreground or Background Emulation Monitor

The emulation monitor is a program that is executed by the emulation processor. It allows the emulation controller to access target system resources. For example, when you display target system memory, it is the monitor program that executes 70732 instructions which read the target memory locations and send their contents to the emulation controller.

The monitor program can execute in *foreground*, the mode in which the emulator operates as would the target processor. The foreground monitor occupies processor address space and executes as if it were part of the target program.

The monitor program can also execute in *background*. User program execution is suspended so that emulation processor can be used to access target system resources. The background monitor does not occupy any processor address space.

Real-Time Operation

Real-time operation signifies continuous execution of your program without interference from the emulator. (Such interference occurs when the emulator temporarily breaks to the monitor so that it can access register contents or target system memory or I/O.)

When your program is running, the emulator accesses emulation memory by holding emulation microprocessor for 12 clock cycles, not breaking to the monitor. You can restrict the emulator to real-time execution. When the emulator is executing your program under the real-time restriction, commands which display/modify registers, display/modify target system memory or I/O are not allowed.

Coverage

The 70732 emulator does not support coverage test.

Easy Products Upgrades

Because the HP 64700 Series development tools (emulator, analyzer, LAN board) contain programmable parts, it is possible to reprogram the firmware and some of the hardware without disassembling the HP 64700B Card Cage. This means that you'll be able to update product firmware, if desired, without having to call an HP field representative to your site.

Limitations, Restrictions

Reset While in Background Monitor

If you use background monitor, RESET from target system are ignored while in monitor.

User Interrupts While in Background Monitor

If you use the background monitor, NMI from target system are suspended until the emulator goes into foreground operation. Other interrupts are ignored.

Interrupts While Executing Step Command

While stepping user program with the foreground monitor used, interrupts are accepted if they are enabled in the foreground monitor program.

While stepping user program with the background monitor used, interrupts are ignored.

Note 

You should not use step command in case the interrupt handler's punctuality is critical.

Evaluation Chip

Hewlett-Packard makes no warranty of the problem caused by the 70732 Evaluation chip in the emulator.

2

Getting Started

Introduction

This chapter will lead you through a basic, step by step tutorial that shows how to use the HP 64752A emulator for the 70732 microprocessor.

This chapter will:

- Describe the sample program used for this chapter's examples.
- Show you how to use the "help" facility.
- Show you how to use the memory mapper.
- Show you how to enter emulation commands to view execution of the sample program. The commands described in this chapter include:
  - Displaying and modifying memory
  - Stepping
  - Displaying registers
  - Defining macros
  - Searching memory
  - Running
  - Breaking
  - Using software breakpoints
  - Copying memory

Before You Begin

Before beginning the tutorial presented in this chapter, you must have completed the following tasks:

1. Completed hardware installation of the HP64700 emulator in the configuration you intend to use for your work:
   - Standalone configuration
   - Transparent configuration
   - Remote configuration
   - Local Area Network configuration

   References: *HP 64700 Series Installation/Service* manual

2. If you are using the Remote configuration, you must have completed installation and configuration of a terminal emulator program which will allow your host to act as a terminal connected to the emulator. In addition, you must start the terminal emulator program before you can work the examples in this chapter.

3. If you have properly completed steps 1 and 2 above, you should be able to hit <RETURN> (or <ENTER> on some keyboards) and get one of the following command prompts on your terminal screen:

```
U>
R>
M>
```

   If you do not see one of these command prompts, retrace your steps through the hardware and software installation procedures outlined in the manuals above, verifying all connections and procedural steps.

   In any case, you **must** have a command prompt on your terminal screen before proceeding with the tutorial.

A Look at the Sample Program

The sample program used in this chapter is listed in figure 2-1. The program emulates a primitive command interpreter.

```
A-X- 00000000                          1  .file  "cmd_rds.s"
A-X- 00000000                          2
A-X- 00000000                          3  .set    Dest_Size,0x20
A-X- 00000000                          4  .set    Stack_Size,0x100
A-X- 00000000                          5
A-X- 00000000                          6  .text
A-X- 00000000                          7  .align  4
A-X- 00000000                          8
A-X- 00000000                          9  Init:           mov     #Stack+Stack_Size,sp
A-X- 00000000 20BC0000                   -- movhi  0x0,zero,r1
A-X- 00000004 61A00000                   -- movea  0x0,r1,sp
A-X- 00000008                         10                 mov     #Cmd_Input,r4
A-X- 00000008 20BC0000                   -- movhi  0x0,zero,r1
A-X- 0000000C 81A00000                   -- movea  0x0,r1,gp
A-X- 00000010                         11
A-X- 00000010 04DC0000                12  Clear:          st.w    r0,[r4]
A-X- 00000014                         13                 mov     #Msg_Dest,r5
A-X- 00000014 20BC0000                   -- movhi  0x0,zero,r1
A-X- 00000018 A1A00000                   -- movea  0x0,r1,tp
A-X- 0000001C                         14
A-X- 0000001C C4C00000                15  Scan:           ld.b    [r4],r6
A-X- 00000020 C04C                    16                 cmp     0x00,r6
A-X- 00000022 FA85                    17                 je      Scan
A-X- 00000024                         18
A-X- 00000024                         19  Process_Cmd:    cmp     0x41,r6
A-X- 00000024 20A04100                   -- movea  0x41,zero,r1
A-X- 00000028 C10C                       -- cmp    r1,r6
A-X- 0000002A 0E84                    20                 je      Command_A
A-X- 0000002C                         21                 cmp     0x42,r6
A-X- 0000002C 20A04200                   -- movea  0x42,zero,r1
A-X- 00000030 C10C                       -- cmp    r1,r6
A-X- 00000032 1A84                    22                 je      Command_B
A-X- 00000034 00A82C00                23                 jr      Command_I
A-X- 00000038                         24
A-X- 00000038                         25  Command_A:      mov     #Message_A,r7
A-X- 00000038 20BC0000                   -- movhi  0x0,zero,r1
A-X- 0000003C E1A00000                   -- movea  0x0,r1,r7
A-X- 00000040                         26                 mov     #Message_B,r8
A-X- 00000040 20BC0000                   -- movhi  0x0,zero,r1
A-X- 00000044 01A10000                   -- movea  0x0,r1,r8
A-X- 00000048 00A82800                27                 jr      Write_Msg
A-X- 0000004C                         28
A-X- 0000004C                         29  Command_B:      mov     #Message_B,r7
A-X- 0000004C 20BC0000                   -- movhi  0x0,zero,r1
A-X- 00000050 E1A00000                   -- movea  0x0,r1,r7
A-X- 00000054                         30                 mov     #Message_I,r8
A-X- 00000054 20BC0000                   -- movhi  0x0,zero,r1
A-X- 00000058 01A10000                   -- movea  0x0,r1,r8
A-X- 0000005C 00A81400                31                 jr      Write_Msg
A-X- 00000060                         32
A-X- 00000060                         33  Command_I:      mov     #Message_I,r7
A-X- 00000060 20BC0000                   -- movhi  0x0,zero,r1
A-X- 00000064 E1A00000                   -- movea  0x0,r1,r7
A-X- 00000068                         34                 mov     #Message_End,r8
A-X- 00000068 20BC0000                   -- movhi  0x0,zero,r1
A-X- 0000006C 01A10000                   -- movea  0x0,r1,r8
A-X- 00000070                         35
```

Figure 2-1 Sample program listing

```
A-X- 00000070 27C10000             36  Write_Msg:      ld.b    [r7],r9
A-X- 00000074 25D10000             37                  st.b    r9,[r5]
A-X- 00000078 A144                 38                  add     0x01,r5
A-X- 0000007A E144                 39                  add     0x01,r7
A-X- 0000007C 070D                 40                  cmp     r7,r8
A-X- 0000007E F295                 41                  jne     Write_Msg
A-X- 00000080                      42  Fill_Dest:      mov     #Msg_Dest+Dest_Size,r10
A-X- 00000080 20BC0000               -- movhi  0x0,zero,r1
A-X- 00000084 41A10000               -- movea  0x0,r1,r10
A-X- 00000088 05D00000             43                  st.b    r0,[r5]
A-X- 0000008C A144                 44                  add     0x01,r5
A-X- 0000008E 450D                 45                  cmp     r5,r10
A-X- 00000090 F09D                 46                  jge     Fill_Dest
A-X- 00000092 FFAB7EFF             47                  jr      Clear
A-X- 00000096                      48
AW-- 00000000                      49  .data
AW-- 00000000                      50
AW-- 00000000 436F6D6D616E6420     51  Message_A:      .str    "Command A entered "
              4120656E74657265
              6420
AW-- 00000012 436F6D6D616E6420     52  Message_B:      .str    "Command B entered "
              4220656E74657265
              6420
AW-- 00000024 496E76616C696420     53  Message_I:      .str    "Invalid Command "
              436F6D6D616E6420
AW-- 00000034                      54  Message_End:
AW-- 00000034                      55
AW-- 00000000                      56  .bss
AW-- 00000000                      57
AW-- 00000000                      58  .lcomm          Cmd_Input,4,4
AW-- 00000004                      59  .lcomm          Msg_Dest,Dest_Size,4
AW-- 00000024                      60  .lcomm          Stack,Stack_Size,4
AW-- 00000124                      61
```

Figure 2-1 Sample program listing(Cont'd)

```
COMN : !LOAD ?RW V0x500 {
        .bss    = $NOBITS ?AW;
};

DATA : !LOAD ?RW V0x800 {
        .data   = $PROGBITS ?AW;
};

TEXT : !LOAD ?RWX V0x10000 {
        .text   = $PROGBITS ?AX;
};

__tp_TEXT @ %TP_SYMBOL;
__gp_DATA @ %GP_SYMBOL &__tp_TEXT;
```

Figure 2-2 Linker Command File

Data Declarations

The area at DATA segment defines the messages used by the program to respond to various command inputs. These messages are labeled **Message_A**, **Message_B**, and **Message_I**.

Initialization

The program instructions from the **Init** label to the **Clear** label perform initialization. The segment registers are loaded and the stack pointer is set up.

Reading Input

The instruction at the **Clear** label clears any random data or previous commands from the **Cmd_Input** byte. The **Scan** loop continually reads the **Cmd_Input** byte to see if a command is entered (a value other than 0H).

Processing Commands

When a command is entered, the instructions from **Process_Cmd** to **Command_A** determine whether the command was "A", "B", or an invalid command.

If the command input byte is "A" (ASCII 41H), execution is transferred to the instructions at **Command_A**.

If the command input byte is "B" (ASCII 42H), execution is transferred to the instructions at **Command_B**.

If the command input byte is neither "A" nor "B", i.e. an invalid command has been entered, then execution is transferred to the instructions at **Command_I**.

The instructions at **Command_A**, **Command_B**, and **Command_I** load register R6 with the starting location of the message to be displayed and register R7 with the ending location of the appropriate message. Then, execution transfers to **Write_Msg** where the appropriate message is written to the destination location, **Msg_Dest**.

After the message is written, the instructions at **Fill_Dest** fill the remaining destination locations with zeros. (The entire destination area is 20H bytes long.) Then, the program jumps back to read the next command.

The Destination Area

The area at COMN segment declares memory storage for the command input byte, the destination area, and the stack area.

Using the "help" Facility

The HP 64700 Series emulator's Terminal Interface provides an excellent help facility to provide you with quick information about the various commands and their options. From any system prompt, you can enter "**help**" or "**?**" as shown below.

```
R>help
```

```
help  - display help information

  help <group>         - print help for desired group
  help -s <group>      - print short help for desired group
  help <command>       - print help for desired command
  help                 - print this help screen

--- VALID <group> NAMES ---
  gram     - system grammar
  proc     - processor specific grammar

  hidden   - special use commands normally hidden from user
  sys      - system commands
  emul     - emulation commands
  hl       - highlevel commands (hp internal use only)
  trc      - analyzer trace commands
  *        - all command groups
```

Commands are grouped into various classes. To see the commands grouped into a particular class, you can use the help command with that group. Viewing the group help information in short form will cause the commands or the grammar to be listed without any description.

For example, if you want to get some information for group gram, enter "**help gram**". Following help information should be displayed.

```
R>help gram
```

```
gram - system grammar
------------------------------------------------------------------------------
--- SPECIAL CHARACTERS ---
  #  - comment delimiter    ;  - command separator    Ctl C - abort signal
  {} - command grouping     "" - ascii string         ''    - ascii string
  Ctl R - command recall    Ctl B - recall backwards

--- EXPRESSION EVALUATOR ---
  number bases:  t-ten   y-binary   q-octal   o-octal   h-hex
  repetition and time counts default to decimal - all else default to hex
  operators:     ()  ~  *  /  %  +  -  <<  <<<  >>  >>>  &  ^  |  &&

--- PARAMETER SUBSTITUTION ---
  &token& - pseudo-parameter included in macro definition
          - cannot contain any white space between & pairs
          - performs positional substitution when macro is invoked
  Example
     Macro definition:  mac getfile={load -hbs"transfer -t &file&"}
     Macro invocation:  getfile MYFILE.o
     Expanded command:  load -hbs"transfer -t MYFILE.o"
```

Help information exists for each command. Additionally, there is help information for each of the emulator configuration items.

Becoming Familiar with the System Prompts

A number of prompts are used by the HP 64700 Series emulators. Each of them has a different meaning, and contains information about the status of the emulator before and after the commands execute. These prompts may seem cryptic at first, but there are two ways you can find out what a certain prompt means.

Using "help proc" to View Prompt Description

The first way you can find information on the various system prompts is to look at the **proc** help text.

```
R>help proc
```

```
--- Address format ---
32 bit address for memory or I/O addresses

--- Emulation Prompt Status Characters ---
R - emulator in reset state            c - no target system clock
U - running user program               r - target system reset active
M - running monitor program            h - halt or machine fault
W - waiting for CMB to become ready    w - waiting for target ready
T - waiting for target reset           g - bus grant
? - unknown state

--- Analyzer STATUS Field Equates ---
exec    - execution cycle     fetch   - fetch cycle
fetchbr - fetch after branch  data    - data cycle
mem     - memory cycle        io      - io cycle
read    - read cycle          write   - write cycle
byte    - byte cycle          hword   - half word cycle
word    - word cycle          fault   - machine fault acknowledge
halt    - halt acknowledge    hold    - hold acknowledge
buslock - bus lock            wrrom   - write to rom
grd     - guarded memory      fg      - foreground
bg      - background
```

Using the Emulation Status Command (es) for Description of Current Prompt

When using the emulator, you will notice that the prompt changes after entering certain commands. If you are not familiar with a new prompt and would like information about that prompt only, enter the **es** (emulation status) command for more information about the current status.

```
U>es
N70732--Running user program
```

Initializing the Emulator

If you plan to follow this tutorial by entering commands on your emulator as shown in this chapter, verify that no one else is using the emulator. To initialize the emulator, enter the following command:

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

The **init** command with no options causes a limited initialization, also known as a warm start initialization. Warm start initialization does not

affect system configuration. However, the **init** command will reset emulator and analyzer configurations. The **init** command:

- Resets the memory map.
- Resets the emulator configuration items.
- Resets the break conditions.
- Clears software breakpoints.

The **init** command does not:

- Clear any macros.
- Clear any emulation memory locations; mapper terms are deleted, but if you respecify the same mapper terms, you will find that the emulation memory contents are the same.

Other Types of Initialization

There are two options to **init**. The **-p** option specifies a powerup initialization, also known as a cold start initialization. It initializes the emulator, analyzer, system controller, and communications port; additionally, performance verification tests are run.

The **-c** option also specifies a cold start initialization, except that performance verification tests are not run.

Set Up the Proper Emulation Configuration

Emulation configuration is needed to adapting to your specific development. As you have initialized the emulator, the emulation configuration items have default value.

Set Up Emulation Condition

The emulator allows you to set the emulator's configuration setting with the **cf** command. Enter the **help cf** to view the information with the configuration command.

```
R>help cf
```

```
cf - display or set emulation configuration

  cf                - display current settings for all config items
  cf <item>         - display current setting for specified <item>
  cf <item>=<value> - set new <value> for specified <item>
  cf <item> <item>=<value> <item> - set and display can be combined

help cf <item>    - display long help for specified <item>

--- VALID CONFIGURATION <item> NAMES ---
  bussize  - select the data bus width
  cache    - en/dis instruction cache
  coh      - enable/disable restriction to real time runs
  dasms    - en/dis access memory to disassemble trace list
  dbc      - en/dis drive of background cycles to the target system
  hld      - en/dis Target HLDRQ(-) signal
  mon      - selection of a foreground or background monitor
  monloc   - selection of monitor address
  nmi      - en/dis Target NMI(-) signal
  rdy      - en/dis READY(-) interlock
  rrt      - enable/disable restriction to real time runs
  rst      - en/dis Target RESET(-) signal
  szrq     - en/dis Target SZRQ(-) signal
  tradr    - tracing bus address as data
  trfetch  - en/dis tracing fetch cycle
  trmode   - select analyzer mode
  waitb0   - determine if insert wait cycle on bank0
  waitb1   - determine if insert wait cycle on bank1
```

To view the current emulator configuration setting, enter the following command.

```
R>cf
```

```
cf bussize=32
cf cache=en
cf coh=dis
cf dasms=dis
cf dbc=en
cf hld=en
cf mon=bg
cf monloc=0
cf nmi=en
cf rdy=en
cf rrt=dis
cf rst=en
cf szrq=en
cf tradr=dis
cf trfetch=en
cf trmode=exe
cf waitb0=en
cf waitb1=en
```

The individual configuration items won't be explained in this section; refer to the "CONFIG_ITEMS" in the "70732 Emulator Specific Command Syntax" appendix for details.

Set Up Access/Display Modes

To avoid problems later while modifying and displaying memory locations, enter the following command:

```
R>mo -ab -db
```

This sets the access and display modes for memory operation to byte.(if they are left at the default mode of word,the memory modification and display examples will not function correctly.)

Mapping Memory

Depending on the memory module, emulation memory consists of 256K, 512K, 1M, 1.25M or 2M bytes, mappable in 4K byte blocks.

The memory mapper allows you to characterize memory locations. It allows you specify whether a certain range of memory is present in the target system or whether you will be using emulation memory for that address range. You can also specify whether the target system memory is ROM or RAM, and you can specify that emulation memory be treated as ROM or RAM.

Note

Target system devices that take control of the bus (for example, external DMA controllers), cannot access emulation memory.

Blocks of memory can also be characterized as guarded memory. Guarded memory accesses will generate "break to monitor" requests. Writes to ROM will also generate "break to monitor" requests if the **rom** break condition is enabled. Memory is mapped with the **map** command. To view the memory mapping options, enter:

```
M>help map
```

```
map - display or modify the processor memory map

  map                                 - display the current map structure
  map <addr>..<addr> <type>           - define address range as memory type
  map <addr>..<addr> <type> <attr>    - define address range as memory type
                                        with optional memory attribute
  map other <type> <attr>             - define all other ranges as memory type
                                        with optional memory attribute
  map -d <term#>                      - delete specified map term
  map -d *                            - delete all map terms

--- VALID <type> OPTIONS ---
  eram  -  emulation ram
  erom  -  emulation rom
  tram  -  target ram
  trom  -  target rom
  grd   -  guarded memory

--- VALID <attr> OPTIONS ---
  b0_d32 - using bank0 and 32 bit data bus
  b0_d16 - using bank0 and 16 bit data bus
  b1_d32 - using bank1 and 32 bit data bus
  b1_d16 - using bank1 and 16 bit data bus
  b0     - using bank0 and 32 bit data bus
  b1     - using bank1 and 32 bit data bus
```

Enter the **map** command with no options to view the default map structure.

```
M>map
```

```
# remaining number of terms : 16
# remaining emulation memory : 80000h bytes
map other tram
```

Which Memory Locations Should be Mapped?

Typically, assemblers generate relocatable files and linkers combine relocatable files to form the absolute file. A linker load map listing will show what memory locations your program will occupy. One for the sample program is shown below.

```
                ******** LINK EDITOR ALLOCATION MAP ********

OUTPUT          INPUT           VIRTUAL         SIZE            INPUT
SECTION         SECTION         ADDRESS                         FILE

.bss                                            0x00000500      0x00000124
                .bss            0x00000500      0x00000124      cmd_rds.o

.data                                           0x00000800      0x00000034
                .data           0x00000800      0x00000034      cmd_rds.o

.text                                           0x00010000      0x00000096
                .text           0x00010000      0x00000096      cmd_rds.o

.symtab                                         0x00000000      0x00000220
                .symtab         0x00000000      0x00000220      *(nil)*

.strtab                                         0x00000000      0x0000011b
                .strtab         0x00000000      0x0000011b      *(nil)*

.shstrtab                                       0x00000000      0x0000002c
                .shstrtab       0x00000000      0x0000002c      *(nil)*
```

From the load map listing, you can see that the sample program occupies three address ranges. The program area, which contains the opcodes and operands, occupies locations 10000 through 10096 hex. The data area, which contains the ASCII values of the messages the program transfers, occupies locations 800 through 834 hex. The destination area, which contains the command input byte and the locations of the message destination, occupies locations 500 through 624 hex.

Since the program writes to the destination area, the mapper block of destination area should not be characterized as ROM. Enter the following commands to map memory for the sample program and display the memory map.

```
R>map 0..fff eram
R>map 10000..10fff erom
R>map
```

```
# remaining number of terms   : 14
# remaining emulation memory  : 7e000h bytes
map  000000000..000000fff  eram           # term 1
map  000010000..000010fff  erom           # term 2
map  other tram
```

When mapping memory for your target system programs, you should characterize emulation memory locations containing programs and constants (locations which should not be written) as ROM. This will prevent programs and constants from being written over accidentally. Break will occur when instructions or commands attempt to do so(if the **rom** break condition is enabled).

Getting the Sample Program into Emulation Memory

This section assumes you are using the emulator in one of the following three configurations:

1. Connected only to a terminal, which is called the *standalone* configuration. In the standalone configuration, you must modify memory to load the sample program.

2. Connected between a terminal and a host computer, which is called the *transparent* configuration. In the transparent configuration, you can load the sample program by downloading from the "other" port.

3. Connected to a host computer and accessed via a terminal emulation program. This configurations is called *remote* configurations. In the remote configuration, you can load the sample program by downloading from the same port.

Standalone Configuration

If you are operating the emulator in the standalone configuration, the only way to load the sample program into emulation memory is by modifying emulation memory locations with the **m** (memory display/modification) command.

You can enter the sample program into memory with the **m** command as shown below.

```
R>m -dw 10000=0000bc20,0624a061,0000bc20,0500a081
R>m -dw 10010=0000dc04,0000bc20,0504a0a1,0000c0c4
R>m -dw 10020=85fa4cc0,0041a020,840e0cc1,0042a020
R>m -dw 10030=841a0cc1,002ca800,0000bc20,0800a0e1
R>m -dw 10040=0000bc20,0812a101,0028a800,0000bc20
R>m -dw 10050=0812a0e1,0000bc20,0824a101,0014a800
R>m -dw 10060=0000bc20,0824a0e1,0000bc20,0834a101
R>m -dw 10070=0000c127,0000d125,44e144a1,95f20d07
R>m -dw 10080=0000bc20,0524a141,0000d005,0d4544a1
R>m -dw 10090=0abff9df0,ffffff7e
R>m -db 800="Command A entered Command B entered Invalid command "
```

After entering the opcodes and operands, you would typically display memory in mnemonic format to verify that the values entered are correct (see the example below). If any errors exist, you can modify individual locations. Also, you can use the **cp** (copy memory) command if, for example, a byte has been left out, but the locations which follow are correct.

Note

Be careful about using this method to enter programs from the listings of relocatable source files. If source files appear in relocatable sections, the address values of references to locations in other relocatable sections are not resolved until link-time. The correct values of these address operands will not appear in the assembler listing.

Transparent Configuration

If your emulator is connected between a terminal and a host computer, you can download programs into memory using the **load** command with the **-o** (from other port) option. The **load** command will accept absolute files in the following formats:

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

The examples which follow will show you the methods used to download HP absolute files and the other types of absolute files.

HP Absolutes

Downloading HP format absolute files requires the **transfer** protocol. The example below assumes that the **transfer** utility has been installed on the host computer (HP 64884 for HP 9000 Series 500, or HP 64885 for HP 9000 Series 300).

Note

Notice that the transfer command on the host computer is terminated with the <ESCAPE>**g** characters; by default, these are the characters which temporarily suspend the transparent mode to allow the emulator to receive data or commands.

```
R>load -hbo <RETURN> <RETURN>
$ transfer -rtb cmd_rds.X <ESCAPE>g
####
R>
```

Other Supported Absolute Files

The example which follows shows how to download Intel hexadecimal files by the same method (but different **load** options) can be used by load Tektronix hexadecimal and Motorola S-record files as well.

```
R>load -io <RETURN> <RETURN>
$ cat ihexfile <ESCAPE>g
#####
Data records = 00003 Checksum error = 00000
R>
```

Remote Configuration

If the emulator is connected to a host computer, and you are accessing the emulator from the host computer via a terminal emulation program, you can also download files with the **load** command. However, in the remote configuration, files are loaded from the same port that commands are entered from. For example, if you wish to download a Tektronix hexadecimal file from a Vectra personal computer, you would enter the following commands.

```
R>load -t <RETURN>
```

After you have entered the **load** command, exit from the terminal emulation program to the MS-DOS operating system. Then, copy your hexadecimal file to the port connected to the emulator, for example:

```
C:\copy thexfile com1: <RETURN>
```

Now you can return to the terminal emulation program and verify that the file was loaded correctly.

For More Information

For more information on downloading absolute files, refer to the **load** command description in the *HP 64700 Emulators Terminal Interface: User's Reference* manual.

Loading an ASCII Symbol File

The 70732 emulator supports the use of symbolic references in the terminal interface. The symbols can be loaded with a program file, as is the case with Intel OMF files . They can also be loaded from an ASCII text file on a host system.

The symbols used are defined in a file using a text editor, or any other means to create the file. Refer to the *HP64700-Series Emulators Terminal Interface Reference* for information on the format of the file. The file is then transferred to the emulator using the **load** command.

You can create a text file named "cmd_rds.SYM" on your HP-UX host system. The file will look something like as follows.

```
#
cmd_rds:
Init 10000
Clear 10010
Scan 1001c
Process_Cmd 10024
Command_A 10038
Command_B 1004c
Command_I 10060
Write_Msg 10070
Fill_Dest 10080
#
```

Use the "-S" option on the **load** command to transfer the file.

```
R>load -Sos "cat cmd_rds.SYM"
```

The symbols can then be manipulated with the "sym" command, and used in commands at the command line. If the load is not successful, the nature of the error will be reported.

```
R>sym

sym cmd_rds:Init=000010000
sym cmd_rds:Clear=000010010
sym cmd_rds:Scan=00001001c
sym cmd_rds:Process_Cmd=000010024
sym cmd_rds:Command_A=000010038
sym cmd_rds:Command_B=00001004c
sym cmd_rds:Command_I=000010060
sym cmd_rds:Write_Msg=000010070
sym cmd_rds:Fill_Dest=000010080
```

Displaying Memory In Mnemonic Format

Once you have loaded a program into the emulator, you can verify that the program has indeed been loaded by displaying memory in mnemonic format.

```
R>m -dm 10000..10095

000010000  -              MOVHI   0x0000,R0,R1
000010004  -              MOVEA   0x0624,R1,R3
000010008  -              MOVHI   0x0000,R0,R1
00001000c  -              MOVEA   0x0500,R1,R4
000010010  -              ST.W    R0,0x0000[R4]
000010014  -              MOVHI   0x0000,R0,R1
000010018  -              MOVEA   0x0504,R1,R5
00001001c  -              LD.B    0x0000[R4],R6
000010020  -              CMP     0x00,R6
000010022  -              JZ/JE   0x0001001c
000010024  -              MOVEA   0x0041,R0,R1
000010028  -              CMP     R1,R6
00001002a  -              JZ/JE   0x00010038
00001002c  -              MOVEA   0x0042,R0,R1
000010030  -              CMP     R1,R6
000010032  -              JZ/JE   0x0001004c
000010034  -              JR      0x00010060
000010038  -              MOVHI   0x0000,R0,R1
00001003c  -              MOVEA   0x0800,R1,R7
000010040  -              MOVHI   0x0000,R0,R1
000010044  -              MOVEA   0x0812,R1,R8
000010048  -              JR      0x00010070
00001004c  -              MOVHI   0x0000,R0,R1
000010050  -              MOVEA   0x0812,R1,R7
000010054  -              MOVHI   0x0000,R0,R1
000010058  -              MOVEA   0x0824,R1,R8
00001005c  -              JR      0x00010070
000010060  -              MOVHI   0x0000,R0,R1
000010064  -              MOVEA   0x0824,R1,R7
```

```
000010068  -                MOVHI   0x0000,R0,R1
00001006c  -                MOVEA   0x0834,R1,R8
000010070  -                LD.B    0x0000[R7],R9
000010074  -                ST.B    R9,0x0000[R5]
000010078  -                ADD     0x01,R5
00001007a  -                ADD     0x01,R7
00001007c  -                CMP     R7,R8
00001007e  -                JNZ/JNE 0x00010070
000010080  -                MOVHI   0x0000,R0,R1
000010084  -                MOVEA   0x0524,R1,R10
000010088  -                ST.B    R0,0x0000[R5]
00001008c  -                ADD     0x01,R5
00001008e  -                CMP     R5,R10
000010090  -                JGE     0x00010080
000010092  -                JR      0x00010010
```

If you display memory in mnemonic format and do not recognize the instructions listed or see some illegal instructions or opcodes, go back and make sure the memory locations you have typed are mapped properly. If the memory map is not the problem, recheck the linker load map listing to verify that the absolute addresses of the program match with the locations you are trying to display.

If you have loaded symbols with the sample program, the display will include the symbols in the memory display.

```
000010000  cmd_rds:Init     MOVHI   0x0000,R0,R1
000010004  -                MOVEA   0x0624,R1,R3
000010008  -                MOVHI   0x0000,R0,R1
00001000c  -                MOVEA   0x0500,R1,R4
000010010  cmd_rds:Clear    ST.W    R0,0x0000[R4]
000010014  -                MOVHI   0x0000,R0,R1
000010018  -                MOVEA   0x0504,R1,R5
00001001c  cmd_rds:Scan     LD.B    0x0000[R4],R6
000010020  -                CMP     0x00,R6
000010022  -                JZ/JE   0x0001001c
000010024  rds:Process_Cmd  MOVEA   0x0041,R0,R1
000010028  -                CMP     R1,R6
00001002a  -                JZ/JE   0x00010038
00001002c  -                MOVEA   0x0042,R0,R1
000010030  -                CMP     R1,R6
000010032  -                JZ/JE   0x0001004c
000010034  -                JR      0x00010060
000010038  d_rds:Command_A  MOVHI   0x0000,R0,R1
00001003c  -                MOVEA   0x0800,R1,R7
000010040  -                MOVHI   0x0000,R0,R1
000010044  -                MOVEA   0x0812,R1,R8
000010048  -                JR      0x00010070
00001004c  d_rds:Command_B  MOVHI   0x0000,R0,R1
000010050  -                MOVEA   0x0812,R1,R7
000010054  -                MOVHI   0x0000,R0,R1
000010058  -                MOVEA   0x0824,R1,R8
00001005c  -                JR      0x00010070
000010060  d_rds:Command_I  MOVHI   0x0000,R0,R1
000010064  -                MOVEA   0x0824,R1,R7
```

```
000010068  -                MOVHI   0x0000,R0,R1
00001006c  -                MOVEA   0x0834,R1,R8
000010070  d_rds:Write_Msg  LD.B    0x0000[R7],R9
000010074  -                ST.B    R9,0x0000[R5]
000010078  -                ADD     0x01,R5
00001007a  -                ADD     0x01,R7
00001007c  -                CMP     R7,R8
00001007e  -                JNZ/JNE 0x00010070
000010080  d_rds:Fill_Dest  MOVHI   0x0000,R0,R1
000010084  -                MOVEA   0x0524,R1,R10
000010088  -                ST.B    R0,0x0000[R5]
00001008c  -                ADD     0x01,R5
00001008e  -                CMP     R5,R10
000010090  -                JGE     0x00010080
000010092  -                JR      0x00010010
```

Note

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.

Stepping Through the Program

The emulator allows you to execute one instruction or a number of instructions with the **s** (step) command. Enter the **help s** to view the options available with the step command.

```
R>help s
```

```
s - step emulation processor

  s                     - step one from current PC
  s <count>             - step <count> from current PC
  s <count> $           - step <count> from current PC
  s <count> <addr>      - step <count> from <addr>
  s -q <count> <addr>   - step <count> from <addr>, quiet mode
  s -w <count> <addr>   - step <count> from <addr>, whisper mode

--- NOTES ---
  STEPCOUNT MUST BE SPECIFIED IF ADDRESS IS SPECIFIED!
  If <addr> is not specified, default is to step from current PC.
  A <count> of 0 implies step forever.
```

A step count of 0 will cause the stepping to continue "forever" (until some break condition, such as "write to ROM", is encountered, or until you enter <CTRL>**c**). The following command will step from the first address of the sample program.

```
R>s 1 10000

000010000  -                MOVHI   0x0000,R0,R1
PC = 000010004
```

Displaying Registers

The step command shown above executed the "MOVHI 0x0000,R0,R1" instruction. Enter the following command to view the contents of the registers.

```
M>reg *

reg   pc=00010004  psw=00008000
reg   r0=00000000   r1=00000000   r2=00000000   r3=00000000   r4=00000000
reg   r5=00000000   r6=00000000   r7=00000000   r8=00000000   r9=00000000
reg  r10=00000000  r11=00000000  r12=00000000  r13=00000000  r14=00000000
reg  r15=00000000  r16=00000000  r17=00000000  r18=00000000  r19=00000000
reg  r20=00000000  r21=00000000  r22=00000000  r23=00000000  r24=00000000
reg  r25=00000000  r26=00000000  r27=00000000  r28=00000000  r29=00000000
reg  r30=00000000  r31=00000000
```

The register contents are displayed in a "register modify" command format. This allows you to save the output of the **reg** command to a command file which may later be used to restore the register contents. (Refer to the **po** (port options) command description in the *Terminal Interface: User's Reference* for more information on command files.)

Refer to the "REGISTER CLASS and NAME" section in the "70732 Emulator Specific Command Syntax" appendix for more information on the register names and classes.

Combining Commands

More than one command may be entered in a single command line. The commands must be separated by semicolons (;). For example, you could execute the next instruction(s) and display the registers by entering the following.

```
M>s;reg
```

```
000010004  -               MOVEA   0x0624,R1,R3
PC = 000010008
reg   pc=00010008  psw=00008000
reg   r0=00000000   r1=00000000   r2=00000000   r3=00000624   r4=00000000
reg   r5=00000000   r6=00000000   r7=00000000   r8=00000000   r9=00000000
reg  r10=00000000  r11=00000000  r12=00000000  r13=00000000  r14=00000000
reg  r15=00000000  r16=00000000  r17=00000000  r18=00000000  r19=00000000
reg  r20=00000000  r21=00000000  r22=00000000  r23=00000000  r24=00000000
reg  r25=00000000  r26=00000000  r27=00000000  r28=00000000  r29=00000000
reg  r30=00000000  r31=00000000
```

The sample above shows you that "MOVEA 0x0624,R1,R3" is executed by step command.

Using Macros

Suppose you want to continue stepping through the program and displaying registers after each step. You could continue entering **s** command followed by **reg** command, but you may find this tiresome. It is easier to use a macro to perform a sequence of commands which will be entered again and again.

Macros allow you to combine and store commands. For example, to define a macro which will display registers after every step, enter the following command.

```
M>mac st={s;reg}
```

Once the **st** macro has been defined, you can use it as you would use any other command.

```
M>st
```

```
# s ; reg
000010008  -               MOVHI   0x0000,R0,R1
PC = 00001000c
reg   pc=0001000c  psw=00008000
reg   r0=00000000   r1=00000000   r2=00000000   r3=00000624   r4=00000000
reg   r5=00000000   r6=00000000   r7=00000000   r8=00000000   r9=00000000
reg  r10=00000000  r11=00000000  r12=00000000  r13=00000000  r14=00000000
reg  r15=00000000  r16=00000000  r17=00000000  r18=00000000  r19=00000000
reg  r20=00000000  r21=00000000  r22=00000000  r23=00000000  r24=00000000
reg  r25=00000000  r26=00000000  r27=00000000  r28=00000000  r29=00000000
reg  r30=00000000  r31=00000000
```

Command Recall

The command recall feature is yet another, easier way to enter commands again and again. You can press <CTRL>**r** to recall the commands which have just been entered. If you go past the command of interest, you can press <CTRL>**b** to move forward through the list

of saved commands. To continue stepping through the sample program, you could repeatedly press <CTRL>**r** to recall and <RETURN> to execute the **st** macro.

Repeating Commands

The **rep** command is also helpful when entering commands repetitively. You can repeat the execution of macros as well as normal commands. For example, you could enter the following command to cause the **st** macro to be executed four times.

```
M>rep 4 st
```

```
# s ; reg
00001000c  -                MOVEA   0x0500,R1,R4
PC = 000010010
reg   pc=00010010  psw=00008000
reg   r0=00000000   r1=00000000   r2=00000000   r3=00000624   r4=00000500
reg   r5=00000000   r6=00000000   r7=00000000   r8=00000000   r9=00000000
reg  r10=00000000  r11=00000000  r12=00000000  r13=00000000  r14=00000000
reg  r15=00000000  r16=00000000  r17=00000000  r18=00000000  r19=00000000
reg  r20=00000000  r21=00000000  r22=00000000  r23=00000000  r24=00000000
reg  r25=00000000  r26=00000000  r27=00000000  r28=00000000  r29=00000000
reg  r30=00000000  r31=00000000
# s ; reg
000010010  -                ST.W    R0,0x0000[R4]
PC = 000010014
reg   pc=00010014  psw=00008000
reg   r0=00000000   r1=00000000   r2=00000000   r3=00000624   r4=00000500
reg   r5=00000000   r6=00000000   r7=00000000   r8=00000000   r9=00000000
reg  r10=00000000  r11=00000000  r12=00000000  r13=00000000  r14=00000000
reg  r15=00000000  r16=00000000  r17=00000000  r18=00000000  r19=00000000
reg  r20=00000000  r21=00000000  r22=00000000  r23=00000000  r24=00000000
reg  r25=00000000  r26=00000000  r27=00000000  r28=00000000  r29=00000000
reg  r30=00000000  r31=00000000
# s ; reg
000010014  -                MOVHI   0x0000,R0,R1
PC = 000010018
reg   pc=00010018  psw=00008000
reg   r0=00000000   r1=00000000   r2=00000000   r3=00000624   r4=00000500
reg   r5=00000000   r6=00000000   r7=00000000   r8=00000000   r9=00000000
reg  r10=00000000  r11=00000000  r12=00000000  r13=00000000  r14=00000000
reg  r15=00000000  r16=00000000  r17=00000000  r18=00000000  r19=00000000
reg  r20=00000000  r21=00000000  r22=00000000  r23=00000000  r24=00000000
reg  r25=00000000  r26=00000000  r27=00000000  r28=00000000  r29=00000000
reg  r30=00000000  r31=00000000
# s ; reg
000010018  -                MOVEA   0x0504,R1,R5
PC = 00001001c
reg   pc=0001001c  psw=00008001
reg   r0=00000000   r1=00000000   r2=00000000   r3=00000624   r4=00000500
reg   r5=00000504   r6=00000000   r7=00000812   r8=00000812   r9=00000020
reg  r10=00000524  r11=00000000  r12=00000000  r13=00000000  r14=00000000
reg  r15=00000000  r16=00000000  r17=00000000  r18=00000000  r19=00000000
reg  r20=00000000  r21=00000000  r22=00000000  r23=00000000  r24=00000000
reg  r25=00000000  r26=00000000  r27=00000000  r28=00000000  r29=00000000
reg  r30=00000000  r31=00000000
```

Command Line Editing

The terminal interface supports the use of HP-UX **ksh(1)**-like editing of the command line. The default is for the command line editing feature to be disabled to be compatible with earlier versions of the interface. Use the **cl** command to enable command line editing.

```
M>cl -e
```

Refer to "Command Line Editing" in the *HP64700-Series Emulators Terminal Interface Reference* for information on using the command line editing feature.

Modifying Memory

The preceding step and register commands show the sample program is executing Scan loop, where it continually reads the command input byte to check if a command had been entered. Use the **m** (memory) command to modify the command input byte.

```
M>m 500=41
```

To verify that 41H has been written to 500H, enter the following command.

```
M>m -db 500

000000500 41
```

When memory was displayed in byte format earlier, the display mode was changed to "byte". The display and access modes from previous commands are saved and they become the defaults.

Specifying the Access and Display Modes

There are a couple different ways to modify the display and access modes. One is to explicitly specify the mode with the command you are entering, as with the command **m -db 500**. The **mo** (display and access mode) command is another way to change the default mode. For example, to display the current modes, define the display mode as "word", and redisplay 500H, enter the following commands.

```
M>mo

mo -ab -db

M>mo -dw
M>m 500

000000500 00000041
```

To continue the rest of program.

```
M>r
U>
```

Display the **Msg_Dest** memory locations (destination of the message, 504H) to verify that the program moved the correct ASCII bytes. At this time you want to see correct byte values, so "-db" option (display with byte) is used.

```
U>m -db 504..523
000000504 43 6f 6d 6d 61 6e 64 20 41 20 65 6e 74 65 72 65
000000514 64 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Running the Sample Program

The emulator allows you to execute a program in memory with the **r** command. The **r** command by itself causes the emulator to begin executing at the current program counter address. The following command will begin running the sample program from 10000H.

```
M> r 10000
```

Note

The defaults number base for address and data values within HP 64700 Terminal Interface is hexadecimal. Other number bases may be specified. Refer to the "Expressions" chapter or the *HP 64700 Terminal Interface Reference* manual for further details.

The **r rst** command specifies that the emulator begin to executing from target system reset (see the "Execution Topics" section in the "In-Circuit Emulation" chapter).

Searching Memory for Data

The **ser** (search memory for data) command is another way to verify that the program did what it was supposed to do.

```
U>ser 500..523="Command A entered "
pattern match at address: 000000504
```

If any part of the data specified in the **ser** command is not found, no match is displayed (No message displayed).

Breaking into the Monitor

You can use the break command (**b**) command to generate a break to the monitor. While the break will occur as soon as possible, the actual stopping point may be many cycles after the break request (depending on the type of instruction being executed and whether the processor is in a special state).

```
U>b
M>
```

Using Software Breakpoints

Software breakpoints are handled by the 70732 BRKRET instruction. When you define or enable a software breakpoint(with the **bp** command), the emulator will replace the opcode at the software breakpoint address with a breakpoint interrupt instruction(BRKRET).

Caution

Software breakpoints should not be set, enabled, disabled, or removed while the emulator is running user code. If any of these commands are entered while the emulator is running user code and the emulator is executing code in the area where the breakpoint is being modified, program execution may be unreliable.

Note

You must only set software breakpoints at memory locations which contain instruction opcodes (not operands or data). If a software breakpoint is set at a memory location which is not an instruction opcode, the software breakpoint instruction will never be executed. Further, your program won't work correctly.

Note

NMI will be ignored, when software breakpoint and NMI occur at the same time.

Note

Because software breakpoints are implemented by replacing opcodes with the BRKRET instructions, you cannot define software breakpoints in target ROM.

You can, however, copy target ROM into emulation memory(see the "Target ROM Debug Topics" section of the "In-Circuit Emulation" chapter). Then you can use software breakpoints.

When software breakpoints are enabled and the emulator detects the BRKRET instruction, it generates a break into the monitor. Since the system controller knows the locations of defined software breakpoints, it can determine whether the BRKRET was generated by an enabled software breakpoint or by a BRKRET instruction in your target program.

If the BRKRET was generated by a software breakpoint, execution breaks to the monitor, and the breakpoint interrupt instruction(BRKRET) is replaced by the original opcode. A subsequent run or step command will execute from this address.

If the BRKRET was generated by a BRKRET instruction in the target program, execution still breaks to the monitor,and an "undefined breakpoint" status message is displayed. To continue program

execution, you must run or step from the target program's breakpoint interrupt vector address.

Displaying and Modifying the Break Conditions

Before you can define software breakpoints, you must enable software breakpoints with the **bc** (break conditions) command. To view the default break conditions and change the software breakpoint condition, enter the **bc** command with no option. This command displays current configuration of break conditions.

```
M>bc
```

```
bc -d bp #disable
bc -e rom #enable
bc -d bnct #disable
bc -d cmbt #disable
bc -d trig1 #disable
bc -d trig2 #disable
```

To enable the software break point feature enter

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

Defining a Software Breakpoint

Now that the software breakpoint feature is enabled, you can define software breakpoints. Enter the following command to break on the address of the **Command_I** (address 10060H) label.

```
M>bp 10060
M>bp
```

```
### BREAKPOINT FEATURE IS ENABLED ###
bp 000010060 #enabled
```

Run the program, and verify that execution broke at the appropriate address.

```
M>r 10000
U>m 500=43
```

```
!ASYNC_STAT 615! Software breakpoint: 000010060
```

```
M>st
```

```
# s ; reg
000010060  -               MOVHI   0x0000,R0,R1
PC = 000010064
reg   pc=00010064  psw=00008000
reg   r0=00000000   r1=00000000   r2=00000000   r3=00000624   r4=00000500
reg   r5=00000504   r6=00000043   r7=00000000   r8=00000000   r9=00000000
reg  r10=00000000  r11=00000000  r12=00000000  r13=00000000  r14=00000000
reg  r15=00000000  r16=00000000  r17=00000000  r18=00000000  r19=00000000
reg  r20=00000000  r21=00000000  r22=00000000  r23=00000000  r24=00000000
reg  r25=00000000  r26=00000000  r27=00000000  r28=00000000  r29=00000000
reg  r30=00000000  r31=00000000
```

When a breakpoint is hit, it becomes disabled. You can use the **-e** option with the **bp** command to re-enable the software breakpoint.

```
                                M>bp
### BREAKPOINT FEATURE IS ENABLED ###
 bp 000010060 #disabled
                                M>bp -e 10060
                                M>bp
### BREAKPOINT FEATURE IS ENABLED ###
 bp 000010060 #enabled
                                M>r
                                U>m 500=43
!ASYNC_STAT 615! Software breakpoint: 000010060
                                M>bp
### BREAKPOINT FEATURE IS ENABLED ###
bp 000010060 #disabled
```

Using the Analyzer

Predefined Trace Labels

Three trace labels are predefined in the 70732 emulator. You can view these labels by entering the **tlb** (trace label) command with no options.

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

Predefined Status Equates

Common values for the 70732 status trace signals have been predefined. You can view these predefined equates by entering the **equ** command with no options.

```
                                M>equ
```

```
### Equates ###
equ bg=0xxxxxxxxxxxxxxy
equ buslock=0xx1xxxx1xxxxxxxy
equ byte=0xx1xxxxxxxxx0xxy
equ data=0xx1xxxxxxx010xxxy
equ exec=0xxx1xxxxxxxxxxxy
equ fault=0xx1xxxxxxx101xxxy
equ fetch=0xx1xxxxxx1011xxxy
equ fetchbr=0xx1xxxxxx1001xxxy
equ fg=1xxxxxxxxxxxxxxy
equ grd=0xx1xx0xxxx0xxxxxy
equ halt=0xx1xxxxxxx111xxxy
equ hold=0xx00xxxxxxxxxxxy
equ hword=0xx1xxxxxxxxxx10xy
equ io=0xx1xxxxxxx110xxxy
equ mem=0xx1xxxxxxx0xxxxxy
equ read=0xx1xxxxxx1xxxxxxy
equ word=0xx1xxxxxxxxxx110y
equ write=0xx1xxxxxx0xxxxxxy
equ wrrom=0xx1x0xxxx00xxxxxy
```

These equates may be used to specify values for the **stat** trace label when qualifying trace conditions.

Specifying a Simple Trigger

The **tg** analyzer command is a simple way to specify a condition on which to trigger the analyzer. Suppose you wish to trace the states of the program after the read of "B"(42H) command from the command input byte. Enter the following commands to set up the trace, run the program, issue the trace, and display the trace status.(Note that the analyzer is to search for a lower byte read of 42H because the command input byte address(500H) is a multiple of four)

```
M>tg addr=500 and data=0xxxxxx42 and
stat=read
M>t

emulation trace started

M>r 10000
U>ts

--- Emulation Trace Status ---
New User trace running
Arm ignored
Trigger not in memory
Arm to trigger ?
States ? (512) ?..?
Sequence term 1
Occurrence left 1
```

The trace status shows that the trigger condition has not been found. You would not expect the trigger to be found because no commands have been entered. Modify the command input byte to "B"(42H) and display the trace status again.

```
U>m 500=42
U>ts
```

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

The trace status shows that the trigger has been found, and that 512 states have been stored in trace memory. Enter the following command to display the first 15 states of the trace.

```
U>tl -t 15
```

```
  Line   addr,H    N70732 Mnemonic                        count,R
 -----   --------  -------------------------------------  ---------
     0   00000500    ......42 data read byte                   ---
     1   00010024    0041a020 fetch                        0.120 uS
     2   00010028    840e0cc1 fetch                        0.120 uS
     3   0001002c    0042a020 fetch                        0.120 uS
     4   00010030    841a0cc1 fetch                        0.120 uS
     5   00010034    002ca800 fetch                        0.120 uS
     6   00010038    0000bc20 fetch                        0.120 uS
     7   0001004c  MOVHI   0x0000,R0,R1                    0.120 uS
     8   00010050  MOVEA   0x0812,R1,R7                    0.120 uS
     9   00010054  MOVHI   0x0000,R0,R1                    0.120 uS
    10   00010058  MOVEA   0x0824,R1,R8                    0.120 uS
    11   0001005c  JR      0x00010070                      0.120 uS
    12   00010060  MOVHI   0x0000,R0,R1                    0.120 uS
    13   00010070  LD.B    0x0000[R7],R9                   0.120 uS
    14   00010074  ST.B    R9,0x0000[R5]                   0.120 uS
```

Line 0 in the trace list above shows the state which triggered the analyzer. The trigger state is always on line 0. When you display trace list, the emulator disassembles "fetch" states, and their disassembled processor mnemonic is displayed at the "fetch" states which are the first byte of the instruction.

To list the next lines of the trace, enter the following command.

```
U>tl
```

```
 Line   addr,H    N70732 Mnemonic                          count,R
-----   --------  -----------------------------------  ---------
   15   00000810    ..43.... data read byte             0.120 uS
   16   00010078  ADD     0x01,R5                       0.120 uS
       =0001007a  ADD     0x01,R7
   17   00000504    ......43 data write byte            0.120 uS
   18   0001007c  CMP     R7,R8                         0.120 uS
       =0001007e  JNZ/JNE 0x00010070
   19   00010080  MOVHI   0x0000,R0,R1                  0.120 uS
   20   00010084  MOVEA   0x0534,R1,R10                 0.120 uS
   21   00010070  LD.B    0x0000[R7],R9                 0.160 uS
   22   00010074  ST.B    R9,0x0000[R5]                 0.120 uS
   23   00000810    6f...... data read byte             0.120 uS
   24   00010078  ADD     0x01,R5                       0.120 uS
       =0001007a  ADD     0x01,R7
   25   00000504    ....6f.. data write byte            0.120 uS
   26   0001007c  CMP     R7,R8                         0.120 uS
       =0001007e  JNZ/JNE 0x00010070
   27   00010080  MOVHI   0x0000,R0,R1                  0.120 uS
   28   00010084  MOVEA   0x0534,R1,R10                 0.120 uS
   29   00010070  LD.B    0x0000[R7],R9                 0.160 uS
```

Specifying a Trace mode

By default, the 70732 instruction cache is enabled. In this case, the emulator can not trace execution status even if you specify that the analyzer trace bus states and execution states("**cf trmode=exe**"). If you want to trace bus status and execution status, you must specify that the 70732 instruction cache is disabled. Enter the following command.

```
M>cf cache=dis
M>cf trmode=exe
```

If bus cycle and execution are occurred simultaneously, bus address can not be traced. Refer "Analyzer Topics" section in the "Using the Emulator" chapter and "CONFIG_ITEM" section in the "70732 Emulation Specific Command Syntax" appendix.

The emulator analyzer has a time or state counter which is affected by clock speed. The analyzer can capture all types of bus cycles correctly up to the maximum clock of 25MHz(clock on the demo board is 25MHz), but it can not count states nor time at those high speeds for certain bus cycle type. Enter the following command:

```
M>tcq none
M>tck -s VF
```

Refer "Analyzer Topics " section in the "Using the Emulator" chapter.

Trigger Position

You can specify where the trigger state will be positioned with in the emulation trace list. The following three basical trigger positions are defined.

- **s** start
- **c** center
- **e** end

When **s**(start) trigger position is selected, the trigger is positioned at the start of the trace list. You can trace the states after the trigger state.

When **c**(center) trigger position is selected, the trigger is positioned at the center of the trace list. You can trace the states around the trigger.

When **e**(end) trigger position is selected, the trigger is positioned at the end of the trace list. You can trace the state before the trigger.

In the above section, you have traced the states of the program after a certain state, because the default trigger position was **s**(start). If you want to trace the states of the program around a certain state, you need to change the trigger position.

For example, if you wish to trace the transition to the command A process, change the trigger position to "center" and specify the trigger condition.

To specify the trigger position, enter the following command.

```
U>tp c
```

Specify the trigger condition by typing

```
U>tg addr=10038 and stat=exec
```

Enter the trace command to start the trace.

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

Modify the command input byte to "A" and display the trace status again.

```
U>m  500=41
U>ts
```

```
--- Emulation Trace Status ---
New User trace complete
Arm ignored
Trigger not in memory
Arm to trigger ?
States 512 (512) -257..254
Sequence term 2
Occurrence left 1
```

The trace status shows that the trigger has been found. Enter the following command to display the states about the execution state of address 10038H.

```
U>tl -5..5
```

```
 Line   addr,H    N70732 Mnemonic                          count,R
-----   --------  -----------------------------------  ---------
   -5   00010022  JZ/JE   0x0001001c              GSS   *********
                    840e0cc1 fetch
   -4   00010024  MOVEA   0x0041,R0,R1                   *********
   -3   00010028  CMP     R1,R6                    GSS   *********
                    0042a020 fetch
   -2   0001002a  JZ/JE   0x00010038              GSS   *********
                    841a0cc1 fetch
   -1   00010038    0000bc20 fetch after branch         *********
    0   00010038  MOVHI   0x0000,R0,R1                   *********
                    0800a0e1 fetch
    1   0001003c  MOVEA   0x0800,R1,R7            GSS   *********
                    0000bc20 fetch
    2   00010040  MOVHI   0x0000,R0,R1            GSS   *********
                    0812a101 fetch
    3   00010044  MOVEA   0x0812,R1,R8            GSS   *********
                    0028a800 fetch
    4   00010048  JR      0x00010070              GSS   *********
                    0000bc20 fetch
    5   00010070    0000c127 fetch after branch         *********
```

The transition states to the process for the command A are displayed.

Note

The character displayed in the right side of trace list specifies the following information.

Character	Information
GSS	Emulator guessed execution address
ADR	Processor masked low bit of address bus by 0
BGM	Background monitor cycles

For a Complete Description

For a complete description of the HP 64700 Series analyzer, refer to the *HP 64700 Emulators Terminal Interface: Analyzer User's Guide.*

Copying Memory

The **cp** (copy memory) command gives you the ability to copy the contents of one range of memory to another. This is a handy feature to test things like the relocatability of programs, etc. To test if the sample program is relocatable within the same segment, enter the following command to copy the program to an unused, but mapped, area of emulation memory. After the program is copied, run it from its new start address to verify that the program is indeed relocatable.

```
U>cp 10500=10000..10095
U>r 10500
U>
```

The prompt shows that the emulator is executing user code, so it looks as if the program is relocatable. You may want to issue a simple trace to verify that the program works while running from its new location.

```
U>tg any
U>t
Emulation trace started

U>tl
```

```
 Line   addr,H    N70732 Mnemonic                          count,R
-----   --------  -----------------------------------  ---------
    0   0001051c  INSTRUCTION--opcode unavailable       *********
                    85fa4cc0 fetch
    1   00000500    ......00 data read byte             *********
    2   00010520  INSTRUCTION--opcode unavailable       *********
    3   00010524    0041a020 fetch                      *********
    4   00010522  JZ/JE   0x0001051c              GSS   *********
                    840e0cc1 fetch
    5   0001051c    0000c0c4 fetch after branch         *********
    6   0001051c  LD.B    0x0000[R4],R6                 *********
                    85fa4cc0 fetch
    7   00000500    ......00 data read byte             *********
    8   00010520  CMP     0x00,R6                 GSS   *********
    9   00010524    0041a020 fetch                      *********
   10   00010522  JZ/JE   0x0001051c              GSS   *********
                    840e0cc1 fetch
```

Resetting the Emulator

To reset the emulator, enter the following command.

```
U>rst
R>
```

The emulator is held in a reset state (suspended) until a **b** (break), **r** (run), or **s** (step) command is entered. A CMB execute signal will also cause the emulator to run if reset.

The **-m** option to the **rst** command specifies that the emulator begin executing in the monitor after reset instead of remaining in the suspended state.

```
R>rst -m
M>
```

3

Using the Emulator

Introduction

Many of the topics described in this chapter involve the commands which are unique to the 70732 emulator such as the **cf** command which allows you to specify emulator configuration.
A reference-type description of the 70732 emulator configuration items can be found in the "CONFIG_ITEMS" section in the "70732 Emulator Specific Command Syntax" appendix.

This chapter will:

- Execution Topics
  - Restricting the Emulator to Real-Time Runs
  - Setting Up to Break on an Analyzer Trigger
  - Making Coordinated Measurements
- Manipulation as 32-bit real numbers
  - Register Manipulation
  - Memory Manipulation
- Memory Mapping
- Analyzer Topics
  - Analyzer Status Qualifiers
  - Specifying Trace Configuration
  - Specifying Data for Trigger or Store Condition
- Instruction Cache
- Real-time Emulation Memory Access
- Monitor Option Topics

Prerequisites

Before performing the tasks described in this chapter, you should be familiar with how the emulator operates in general. Refer to the *Concepts of Emulation and Analysis* manual and the "Getting Started" chapter of this manual.

Execution Topics

The description in this section are of emulation tasks which involve program execution in general.

Restricting the Emulator to Real-Time Runs

By default, the emulator is not restricted to real-time runs. However, you may wish to restrict runs to real-time to prevent accidental breaks that might cause target system problems. Use the **cf** (configuration) command to enable the **rrt** configuration item.

```
R>cf rrt=en
```

When runs are restricted to real-time and the emulator is running user code, the system refuses all commands that cause a break except **rst** (reset), **r** (run), **s**(step), and **b** (break to monitor).

Because the emulator accesses emulation memory by holding microprocessor for 12 clock cycles(not breaking), commands which access emulation memory are allowed while runs are restricted to real-time.

The following commands are not allowed when runs are restricted to real-time:

- **reg** (register display/modification).
- **m** (memory display/modification) commands that access target system memory.
- **io** (I/O display/modification).

The following command will disable the restriction to real-time runs and allow the system to accept commands normally.

```
R>cf rrt=dis
```

Setting Up to Break on an Analyzer Trigger

The analyzer may generate a break request to the emulation processor. To set up to break on an analyzer trigger, follow the steps below.

Specify the Signal Driven when Trigger is Found

Use the **tgout** (trigger output) command to specify which signal is driven when the analyzer triggers. Either the "trig1" or the "trig2" signal can be driven on the trigger.

```
R>tgout trig1
```

Enable the Break Condition

Enable the "trig1" break condition.

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

After you specify the trigger to drive "trig1" and enable the "trig1" break condition, set up the trace, enter the **t** (trace) command, and run the program.

Making Coordinated Measurements

Coordinated measurements are measurements made between multiple HP 64700 Series emulators which communicate via the Coordinated Measurement Bus (CMB). Coordinated measurements can also include other instruments which communicate via the BNC connector. A trigger signal from the CMB or BNC can break emulator execution into the monitor, or it can arm the analyzer. An analyzer can send a signal out on the CMB or BNC when it is triggered. The emulator can send an EXECUTE signal out on the CMB when you enter the **x** (execute) command.

Coordinated measurements can be used to start or stop multiple emulators, start multiple trace measurements, or to arm multiple analyzers.

As with the analyzer generated break, breaks to the monitor on CMB or BNC trigger signals are interpreted as a "request to break". The emulator looks at the state of the CMB READY (active high) line to determine if it should break. It does not interact with the EXECUTE (active low) or TRIGGER (active low) signals.

For information on how to make coordinated measurements, refer to the *HP 64700 Emulators Terminal Interface: Coordinated Measurement Bus User's Guide* manual.

Manipulation as 32-bit Real Numbers

You can display/modify register and memory as 32-bit(single-precision) real numbers.

Register Manipulation

You can display/modify general purpose registers(R0 through R31) as 32-bit(single-precision) real numbers with **reg** command.

To display all general purpose registers as 32-bit real numbers, enter the following command:

```
M>reg -f
```

```
reg  r0=+true zero                  r1=+4.0777785e-0043
reg  r2=+7.6464751e-0039            r3=+3.0183969e-0042
reg  r4=+2.5243344e-0029            r5=+3.7554799e-0042
reg  r6=-3.9134964e-0024            r7=+5.2559930e+0022
reg  r8=+6.8348333e-0041            r9=+2.2043172e-0038
reg r10=+true zero                  r11=+true zero
reg r12=+true zero                  r13=+true zero
reg r14=+true zero                  r15=+true zero
reg r16=+true zero                  r17=+true zero
reg r18=+true zero                  r19=+true zero
reg r20=+true zero                  r21=+true zero
reg r22=+true zero                  r23=+true zero
reg r24=+true zero                  r25=+true zero
reg r26=+true zero                  r27=+true zero
reg r28=+true zero                  r29=+true zero
reg r30=+true zero                  r31=+true zero
```

You can specify register(s) to be displayed. To display two registers R5 and R7 as 32-bit real numbers, enter the following command:

```
M>reg -f r5 r7
```

```
reg  r5=+3.7554799e-0042
reg  r7=+5.2559930e+0022
```

To modify register R5 to the value 12345.678, enter the following command:

```
M>reg -f r5=12345.678
```

Memory Manipulation

You can display/modify memory values as 32-bit(single-precision) real numbers by **m** command with "**-df**" option. This "-df" option does not change current display option. And you can not use **mo** command with "-df" option.

To display memory 100H value as 32-bit real numbers, enter the following command:

```
M>m -df 100

000000100  +4.9ab873e+0003
```

To modify memory 100H to the value 12345.678, enter the follwoing command:

```
M>m -df 100=12345.678
```

Memory Mapping

You can define up to 16 memory ranges(at 4K byte boundaries and at least 4K byte in length). You can characterize memory ranges as emulation RAM, emulation ROM, target RAM, target ROM, or guarded memory. The emulator distinguish left side memory module(bank 0) from right side ones(bank 1) because you can select memory modules whose access speed is different on each bank. When you characterize memory ranges as emulation RAN/ROM, you can specify whether bank number is to be bank 0(**b0**) or bank 1(**b1**) and whether data bus size is to be 16(**d16**) or 32(**d32**) as attributes. When you do not specify bank number, the emulator interprets that bank number is "b0". If you do not specify data bus size, the emulator interprets that data bus size is "d32".

Attributes control specific functionality on a term-by-term basic. Attributes can be the following.

b0_d32	Using emulation memory of bank 0 and data bus width is 32 bits.
b0_d16	Using emulation memory of bank 0 and data bus width is 16 bits.

b1_d32	Using emulation memory of bank 1 and data bus width is 32 bits.
b1_d16	Using emulation memory of bank 1 and data bus width is 16 bits.
b0	Using emulation memory of bank 0 and data bus width is 32 bits.
b1	Using emulation memory of bank 1 and data bus width is 32 bits.

You can specify whether mapping attribute or $\overline{\text{SZRQ}}$ signal from target system is to be valid with "**cf szrq**". If you specify that "**cf bussize=16**", data bus width is 16 regardless of the mapping attribute. Refer "CONFIG_ITEM" section in the "70732 Emulator Specific Command Syntax" appendix.

You can not specify the data bus size for memory mapped as target RAM/ROM. The data bus size for target RAM/ROM is settled by $\overline{\text{SZRQ}}$ signal from target system or "**cf bussize**" configuration.

Analyzer Topics

Analyzer Status Qualifiers

The following are the analyzer status labels which may be used in the "**tg**" and "**tsto**" analyzer commands.

```
backgrnd  0xxxxxxxxxxxxxxy      background
addrerr   0xx1xxxx1xxxxxxxy     bus lock
byte      0xx1xxxxxxxxx0xxy     byte access
data      0xx1xxxxxxx010xxxy    data access
exec      0xxx1xxxxxxxxxxxy     execute instruction
fault     0xx1xxxxxxx101xxxy    machine fault acknowledge
fetch     0xx1xxxxxx1011xxxy    code fetch
fetchbr   0xx1xxxxxx1001xxxy    code fetch after branch
foregrnd  01xxxxxxxxxxxxxxy     foreground
grdacc    0xx1xx0xxxx0xxxxxy    guarded memory access
halt      0xx1xxxxxxx111xxxy    halt acknowledge
hold      0xx00xxxxxxxxxxxy     hold acknowledge
halfwd    0xx1xxxxxxxxx10xy     half word access
io        0xx1xxxxxxx110xxxy    I/O access
mem       0xx1xxxxxxx0xxxxxy    memory access
read      0xx1xxxxxx1xxxxxxy    read cycle
word      0xx1xxxxxxxxx110y     word access
write     0xx1xxxxxx0xxxxxxy    write cycle
wrrom     0xx1x0xxxx00xxxxxy    write to ROM
```

Specifying Trace configuration

You can specify trace configuration with "**cf**" command, when you use analyzer. The differences depend on configuration will be shown using following example.

```
000020000  -    MOVEA   0x1000,R0,R4
000020004  -    ST.W    R0,0x0000[R4]
000020008  -    LD.B    0x0000[R4],R5
00002000c  -    CMP     0x00,R5
00002000e  -    JZ/JE   0x00020008
000020010  -    MOVEA   0x0500,R1,R6
000020014  -    JR      0x00020000
```

Trace Mode

If you wish to make analyzer trace bus status only, enter the following command:

```
M>cf trmode=bus
```

In this case, analyzer can not trace execution state. When you display trace list, the emulator disassembles with "fetch" states, and their disassembled processor mnemonics is displayed at the "fetch" states which are the first byte of the instruction. This is significant when specify the trigger condition at the execution of the instruction which

follows a branch instruction like above. Assume that the process execution instruction of the address range 20000H through 2000eH normally, and the instruction at address 20010H and 20014H are executed at long intervals. If you wish to trigger the analyzer at execution of the address 20010H and you would enter the following command because analyzer can not trace execution state, the trigger will always occur at the fetch of the address 20010H whether or not the branch condition at address 2000eH is satisfied.

```
U>tg addr=20010
```

If you want to trace execution state and bus state, enter the following command:

```
M>cf cache=dis
M>cf trmode=exe
```

Tracing Bus Address

When you specify that the analyzer trace the execute state and bus state, the 70732 emulator transfer execution address to analyzer preferentially. You can specify whether or not forcing the analyzer to trace the bus address as it data when bus cycle and execution are occurred simultaneously. To trace bus address surely, the 70732 emulator transfer bus address as its data to analyzer. So bus data are lost. To trace bus address, enter the following command:

```
U>cf tradr=en
```

When you set this configuration, you will see trace list like follows.

```
 Line   addr,H    N70732 Mnemonic                        count,R
-----   --------  -------------------------------------  ---------
    0   00020008  LD.B    0x0000[R4],R5                  *********
       =0002000c    ******** fetch
    1   00001000    ......** data read byte              *********
    2   0002000c  CMP     0x00,R5                        *********
    3   00020010    ******** fetch                       *********
    4   0002000e  JZ/JE   0x00020008                     *********
       =00020014    ******** fetch
    5   00020008    ******** fetch after branch          *********
    6   00020008  LD.B    0x0000[R4],R5                  *********
       =0002000c    ******** fetch
    7   00001000    ......** data read byte              *********
    8   0002000c  CMP     0x00,R5                        *********
    9   00020010    ******** fetch                       *********
```

To trace bus data when bus cycle and execution are occurred simultaneously, enter the following command:

```
U>cf tradr=dis
```

When you set this configuration, you will see trace list like follows.

```
 Line   addr,H    N70732 Mnemonic                          count,R
-----   --------  -----------------------------------  ---------
    0   00020010    1100a0c0 fetch                     *********
    1   0002000e  INSTRUCTION--opcode unavailable      *********
                    ffecabff fetch
    2   00020008    0000c0a4 fetch after branch        *********
    3   00020008  LD.B    0x0000[R4],R5                *********
                    85fa4ca0 fetch
    4   00001000    ......00 data read byte            *********
    5   0002000c  CMP     0x00,R5                GSS   *********
    6   00020010    1100a0c0 fetch                     *********
    7   0002000e  JZ/JE   0x00020008             GSS   *********
                    ffecabff fetch
    8   00020008    0000c0a4 fetch after branch        *********
    9   00020008  LD.B    0x0000[R4],R5                *********
                    85fa4ca0 fetch
```

Tracing Fetch Cycles

You can specify whether or not analyzer trace fetch cycles. Not to trace fetch cycle, enter the following command:

```
U>cf trfetch=dis
```

When you set this configuration, you will see trace list like follows.

```
 Line   addr,H    N70732 Mnemonic                          count,R
-----   --------  -----------------------------------  ---------
    0   00001000    ......00 data read byte            *********
    1   0002000c  CMP     0x00,R5                    *********
    2   0002000e  JZ/JE   0x00020008                 *********
    3   00020008  LD.B    0x0000[R4],R5              *********
    4   00001000    ......00 data read byte            *********
    5   0002000c  CMP     0x00,R5                    *********
    6   0002000e  JZ/JE   0x00020008                 *********
    7   00020008  LD.B    0x0000[R4],R5              *********
    8   00001000    ......00 data read byte            *********
    9   0002000c  CMP     0x00,R5                    *********
```

To trace fetch cycle, enter following command:

```
U>cf trfetch=en
```

If you specify that analyzer trace only bus state("**cf trmode=bus**"), the analyzer will trace fetch cycle regardless of this configuration.

Disassembling Trace List

You can specify whether the 70732 emulator read data from memory or from trace list when the 70732 emulator disassembles trace list. When the emulator disassembles instructions in stored trace information, the fetch cycles of each instruction are required. When you displayed the trace in mnemonic, some lines which include "INSTRUCTION--opcode unavailable" message were displayed. Each line was instruction execution cycle at the address in the left side of the displayed because the fetch states for the instructions were not stored by the analyzer. To display complete trace list in mnemonic, enter the following command:

```
U>cf dasms=en
```

In this case, the emulator read data from memory to disassemble. When you set this configuration, you will see trace list like follows.

```
 Line   addr,H    N70732 Mnemonic                        count,R
-----   --------  -------------------------------------  ---------
    0   00020008  LD.B    0x0000[R4],R5                  *********
                    85fa4ca0 fetch
    1   00001000    ......00 data read byte              *********
    2   0002000c  CMP     0x00,R5                        *********
    3   00020010    1100a0c0 fetch                       *********
    4   0002000e  JZ/JE   0x00020008                     *********
                    ffecabff fetch
    5   00020008    0000c0a4 fetch after branch          *********
    6   00020008  LD.B    0x0000[R4],R5                  *********
                    85fa4ca0 fetch
    7   00001000    ......00 data read byte              *********
    8   0002000c  CMP     0x00,R5                        *********
    9   00020010    1100a0c0 fetch                       *********
```

To read data from trace list, enter the following command:

```
U>cf dasms=dis
```

If you specify that analyzer trace bus address as data("**cf tradr=en**") or analyzer does not trace fetch cycles("**cf trfetch=dis**"), the emulator read data from memory regardless of this configuration. If you specify that you trace bus state only("**cf trmode=bus**" or "**cf cache=en**"), the emulator read data from trace list regardless of this configuration.

Specifying Trace disassemblely option

If you do not want to see fetch cycles in trace list, specify the **-od** option. To show all bus cycles, specify the **-on** option.

When the analyzer trace actual bus states, you can force disassembly to begin with higher half-word of first trace state by using the **-oh** option. If the disassembled trace list is not what you expected, specify the this option.

Specifying Data for Trigger or Store Condition

The analyzer captures the actual bus states of the 70732 microprocessor, if you specify that "**cf trmode=bus**". When you specify a data in the analyzer trigger condition or store condition, the ways of the analyzer data specification differ according to the data size and address. Suppose that you wish to trigger the analyzer when the processor accesses to the byte data 41H in the address 1000H. You should not specify the trigger condition like this.

```
M>tg addr=1000 and data=41
```

The data condition will be considered as 00000041H. The bit 31 through bit 8 of data bus is unpredictable because of the byte data. You will unable to trigger as you desire. You should have entered as follows.

```
M>tg addr=1000 and data=0xxxxxx41
```

Where **x**' are "don't care" bits.

When the address that you want to trigger is not a multiple of 4, the data bus specification is different from above. If you wish trigger the analyzer at address 1001H instead of the address 1000H, the bit 0 through bit 1 of address are masked by 0 and the data 41H will be output to bit 4 through bit 7 of data bus. You should enter:

```
M>tg addr=1000 and data=0xxxx41xx
```

In case of halfword or word access to data bus, it will be the same.

If user's program access word or halfword data which are not aligned, the 70732 microprocessor mask low bit of address bus(bit 0,1:word data, bit0 :halfword data) by 0. Assume that the processor accesses to the half-word data 1234H in the address 1001H. In this case the following trace list is shown.

```
Line   addr,H    N70732 Mnemonic                          count,R
-----   --------  -----------------------------------  ---------
    0   00020000  MOVEA   0x1001,R0,R4                  *********
    1   00020004  MOVEA   0x1234,R0,R5                  *********
    2   00020008  ST.H    R5,0x0000[R4]                 *********
    3   0002000c  LD.H    0x0000[R4],R6                 *********
    4   00020010  JR      0x00020000                    *********
    5   00001000    ....1234 data write hword     ADR   *********
    6   00001000    ....1234 data read hword      ADR   *********
    7   00020000  MOVEA   0x1001,R0,R4                  *********
    8   00020004  MOVEA   0x1234,R0,R5                  *********
    9   00020008  ST.H    R5,0x0000[R4]                 *********
```

The "ADR"s in the trace list indicate that the 70732 microprocessor masked low bit of address bus by 0.

To trigger the analyzer when the 70732 microprocessor accesses the word data 12345678H at address 1002H in 16 data bus size. The data bus activity of this cycles will be as follows.

```
Sequencer level Address bus Data bus
            1 00001000 xxxx5678
            2 00001002 xxxx1234
```

In this case, you need to use the analyzer sequential trigger capabilities. We do not describe the detail about the sequential trigger feature. Only how to trigger the analyzer at this example is described. To specify the condition of sequencer level 1, enter:

```
M>tif 1 addr=1000 and data=0xxxx5678
```

To specify the condition of sequencer level 2, enter:

```
M>tif 2 addr=1002 and data=0xxxx1234
```

Note

When you trigger/store the analyzer, you should note follows:

1) When you specify "cf tradr=dis", you can not specify address in the analyzer trigger or store condition.
2) When you specify "cf tradr=en", what you specify data in the analyzer trigger or store condition means that you specify address.
3) When execution state and bus state simultaneously, both states are stored in case that both states satisfy store condition.

Analyzer Clock Speed

The emulation analyzer can capture both the execution states and bus states. The analyzer has a counter which allows to count either time or occurrence of bus states.

If you use 64794A/C/D Deep emulation analyzer, the tarce state and time counter quilifiers can be used regardless of clock speed. If you use 64704A emulation analyzer, the trace state and time counter qualifiers are limited by clock speed as the following.

Table 3-1 Analyzer Counter

Clock Speed	Analyzer Speed Setting	Valid count qualifier options
clock =< 16MHz	S(slow)	counting <state> counting time
16MHz < clock =< 20MHz	F(fast)	counting <state>
20MHz < clock =< 25MHz	VF(very fast)	counting off

By default, the analyzer trace only actual bus states, and analyzer counter is turned on. In this case, you can count time and state because the clock speed can be effectively halved even if clock speed is greater than 20MHz.

If you wish to trace both execute states and bus states, you must specify analyzer clock speed.

If your target system clock is equal to 16MHz or less than 16MHz, you can use analyzer time and state counter.

If your target system clock is between 16MHz and 20MHz, you can use the analyzer state counter. In this case, the analyzer state counter counts occurrences of the states which you specify. Assume that you would like to count occurrences of the state which the processor read a data.

```
M>tcq stat=read
M>tck -s F
```

If you use the system clock or your target system clock is greater than 20MHz, you can not use the analyzer counter. Enter the following command:

```
M>tcq none
M>tck -s VF
```

Instruction Cache

You can display/modify/clear instruction cache with "**cache**" command

When the 70732 microprocessor uses instruction cache, entre the following command to view of contents of the instruction cache.

```
M>cache 0..8
```

```
00:00 00000000:ff ff ff ef 00 00 02 41
01:00 00000008:ff a7 df 4e 10 19 00 00
02:00 00000010:ff ff ff ef 00 00 02 41
03:00 00000018:ff a7 df 4e 10 19 00 00
04:00 00000020:ff ff ff ef 00 00 02 41
05:00 00000028:ff a7 df 4e 10 19 00 00
06:00 00000030:ff ff ff ef 00 00 02 41
07:00 00000038:ff a7 df 4e 10 19 00 00
08:00 00000040:ff ff ff ef 00 00 02 41
```

The contents of instruction cache are displayed in the following format.

[Entry No.]:[Valid bit] [Physical Address]:[Data 8 byte]

When you modify instruction cache, you should specify that address is multiple of 8 byte(bit 0-bit 2 is 0) and that bit 3 through bit 9 is equal to entry number. Enter the following command to modify instruction cache.

```
M>cache
07=00,00010038,20,89,00,00,54,80,00,08
```

You can clear all entry of instruction cache with "**cache -cl**" command.

Emulation Memory Access

If you enter display/modify emulation memory command while the user's program is running, HP64700 emulation controller, not the 70732 microprocessor, intends to access the emulation memory. In this case, the emulation controller hold the 70732 bus cycles(not but breaking into the monitor) for 12 clock cycles in order to access to the emulation memory.

Monitor Option Topics

The monitor is a program which is executed by the emulation processor. It allows the emulation system controller to access target system resources. For example, when you enter a command that requires access to target system resources (display target memory, for example), the system controller writes a command code to a communications area and breaks the execution of the emulation processor into the monitor. The monitor program then reads the command from the communications area and executes the processor instructions which access the target system. After the monitor has performed its task, execution returns to the target program.

The background monitor does not take up any processor address space and does not need to be linked to the target program. The monitor resides in dedicated background memory.

Background Monitor

When the emulator is powered up or initialized, the background monitor is selected by default.

Foreground monitor

The default emulator configuration selects the background monitor. The 70732 emulator provides two kinds of foreground monitor. One is included in the 70732 emulator, the other is provided with assembler source file. You can change the emulator configuration to select the foreground monitor. When you select the foreground monitor, processor address space is taken up. The foreground monitor takes up 8K bytes of memory. Use the **cf** command to select the foreground monitor.

```
R>cf mon=fg
R>cf monloc=1000
```

1000 defines an hexadecimal address (on a 8K byte boundary) where the monitor will be located. (Note: this will not load the monitor, it only specifies its location.) The start address of the foreground monitor must be 8k boundary. The foreground monitor must then be loaded into emulation memory. A memory mapper term is automatically created when you execute the **cf mon=fg** command to reserve 8K bytes of memory space for the monitor. The memory map is reset any time **cf mon=bg** is entered. It is only reset when the **cf mon=bg** command is entered if the emulator is not already configured to use the background monitor. Refer to the "Using the Optional Foreground Monitor" appendix.

Note

You must **not** use the foreground monitor if you wish to perform coordinated measurements.

4

In-Circuit Emulation Topics

Introduction

Many of the topics described in this chapter involve the installation, and the commands which relate to using the emulator in-circuit, that is, connected to a target system or demo target board.

This chapter will:

- Show you how to install the emulator probe cable
- Show you how to install the emulation memory module.
- Show you how to install the emulator probe to demo target board.
- Describe the issues concerning the installation of the emulator probe into target systems.
- Describe how to execute program from target reset. This topics is related to program execution in general.
- Describe how to use software breakpoints with ROMed code, and how to test patches to ROMed code. These topics relate to the debugging of target system ROM.

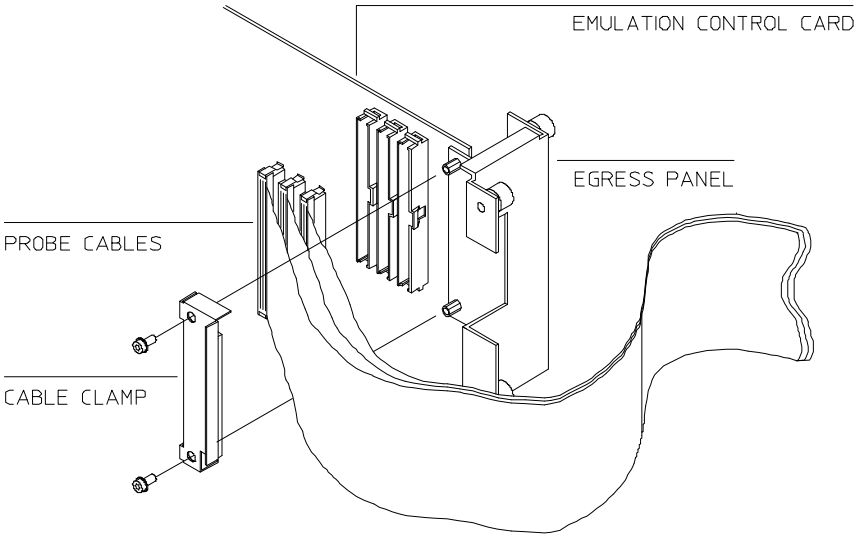
Prerequisites

Before performing the tasks described in this chapter, you should be familiar with how the emulator operates in general. Refer to the *Concepts of Emulation and Analysis* manual and the "Getting Started" chapter of this manual.

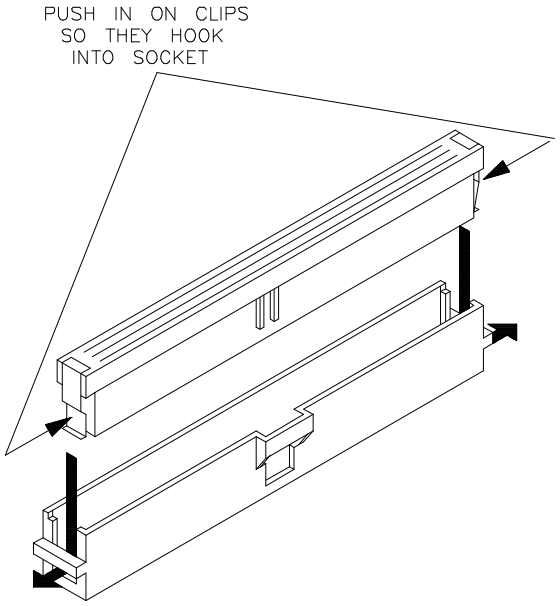
Installing the Emulator Probe Cable

The probe cables consist of three ribbon cables. The longest cable connects to J1 of the emulation control card, and to J1 of the probe. The shortest cable connects to J3 of the emulation control card and J3 of the probe. The ribbon cables are held in place on the emulation control card by a cable clamp attached with two screws. No clamp holds the ribbon cables in the probe.

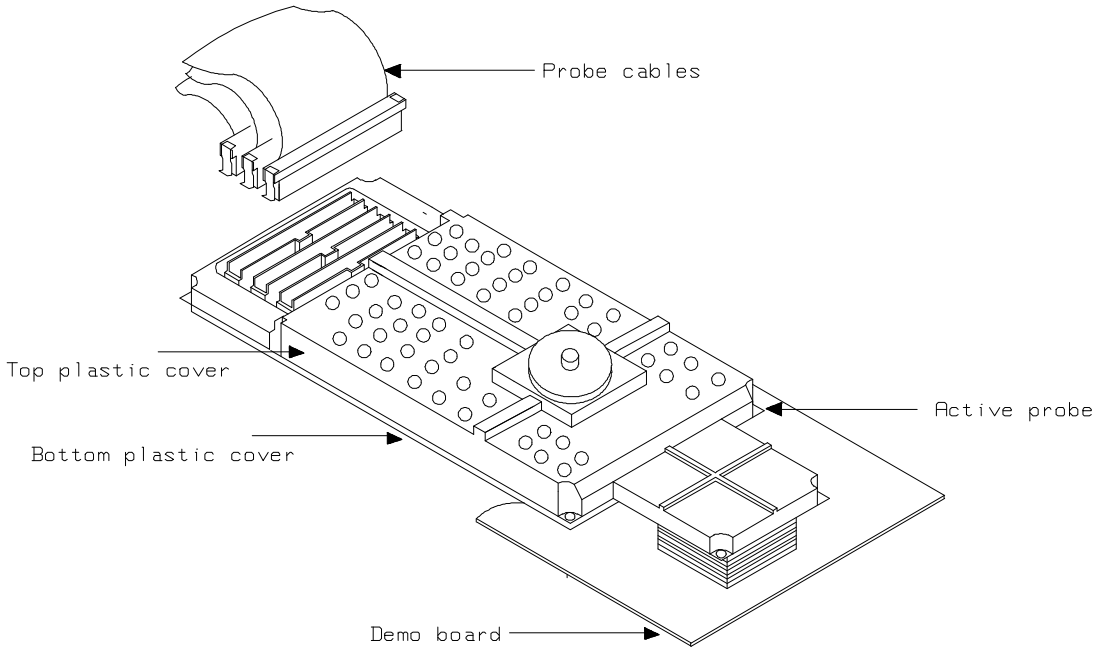
1. Secure the cable on the emulation control card with cable clamp and two screws.



2. When insert the ribbon cables into the appropriate sockets, press inward on the connector clops so that they into the sockets as shown.



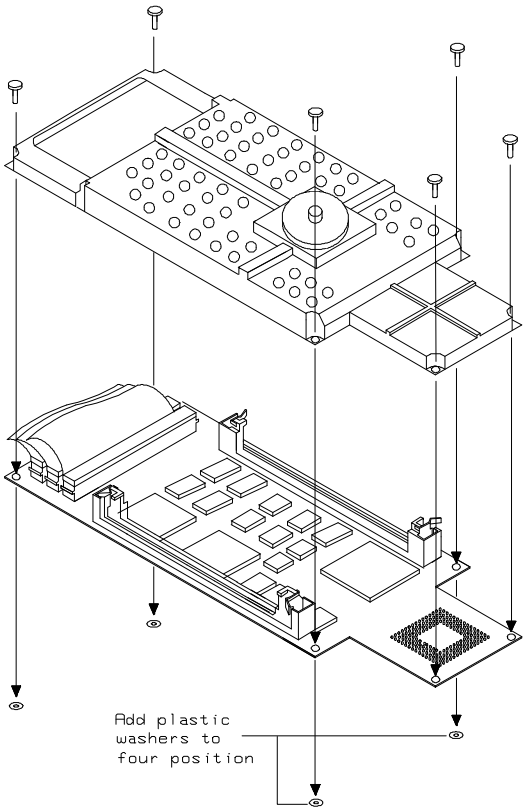
3. Connect the other ends of the cable s to the emulation probe.



Installing the Emulation Memory Module

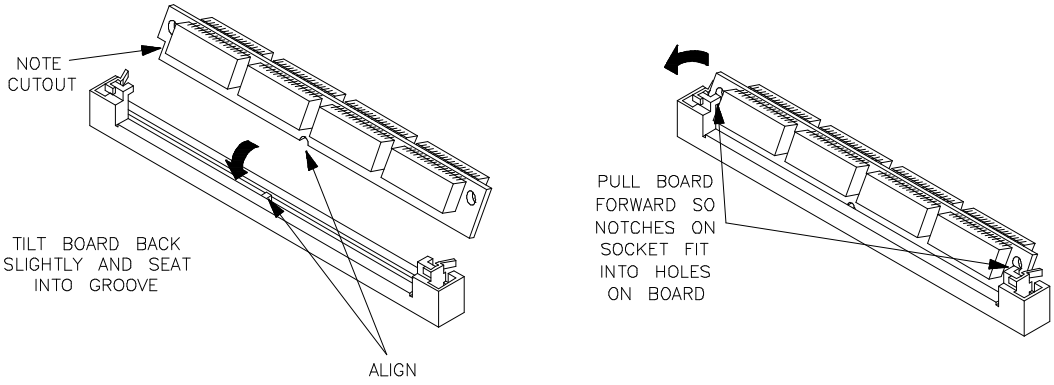
There are four types of emulation memory modules that can be inserted into sockets on the probe.

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. Insert emulation memory module on the emulation probe. There is a cutout on one side of the memory modules so that they can only be installed one way.

   To install memory modules, place the memory module into the socket groove at an angle. Firmly press the memory module into the socket to make sure it is completely seated. Once the memory module is seated in the connector groove, pull the memory module forward so that the notches on the socket fit into the holes on the memory module. There are two latches on the sides of the socket that hold the memory module in place.



3. Replace the plastic cover, and insert new plastic rivets to secure the cover.

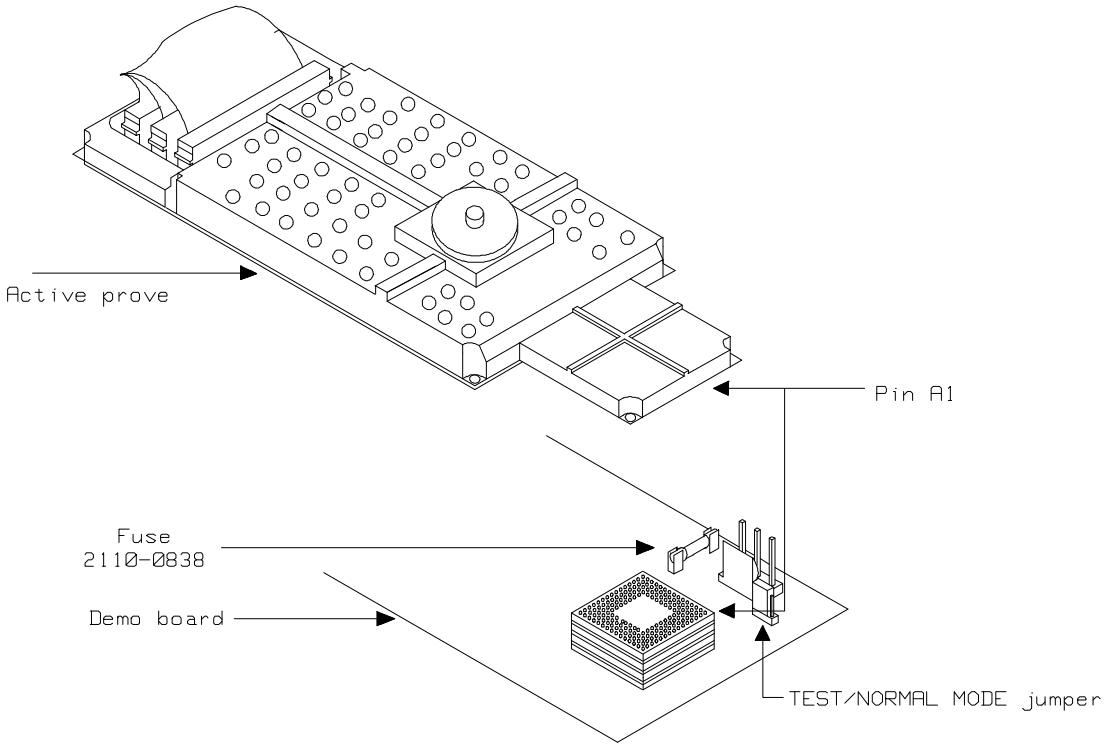
Installing into the Demo Target Board

To connect the microprocessor connector to the demo target board, proceeded with the following instructions.

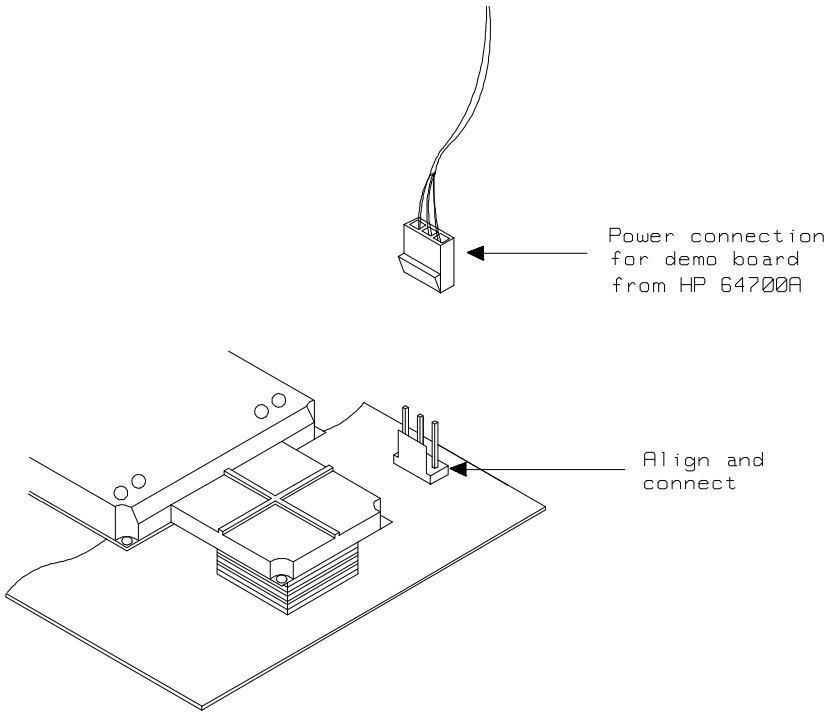
1. Remove front bezel and connect the power cable to connector the HP 64700B front panel. Refer to *HP 64700 Series Installation/Service* manual.

2. With HP 64700B power OFF, connect the emulator probe to the demo target board. When you install the probe into the demo target board, be careful not to bend any of the pins.

   After connecting the probe to the demo target board, set the TEST/NORMAL MODE jumper. Use TEST MODE position when you run performance verification tests, and use NORMAL MODE position when you use emulator normally.



3. Connect the power cable supply wires from the emulator to demo target board. When attaching the wire cable to the demo target board, make sure the connector is aligned properly so that all three pins are connected.



Installing the Emulator Probe into a Target System

The 70732 emulator probe has a 176-pin PGA connector;
The emulator probe is also provided with a conductive pin protector to protect the delicate gold-plated pins of the probe connector from damage due to impact.

Caution

Protect against electrostatic discharge. The emulator probe contains devices that are susceptible to damage by electrostatic discharge. Therefore, precautionary measures should be taken before handling the microprocessor connector attached to the end of the probe cable to avoid damaging the internal components of the probe by electrostatic electricity.

Caution

Make sure target system power is OFF. Do not install the emulator probe into the target system microprocessor socket with power applied to the target system. The emulator may be damaged if target system power is not removed before probe installation.

Caution

Make sure pin 1 of probe connector is aligned with pin 1 of the socket. When installing the emulation probe, be sure that probe is inserted into the processor socket so that pin 1 of the connector aligns with pin 1 of the socket. Damage to the emulator probe will result if the probe is incorrectly installed.

Installing into a PGA Type Socket

To connect the emulator probe to the target system, proceeded with the following instructions.

1. Remove the 70732 microprocessor (PGA type) from the target system socket. Note the location of pin A1 on the microprocessor and on the target system socket.
2. Store the microprocessor in a protected environment (such as antistatic form).
3. Install the emulator probe into the target system microprocessor socket.

Caution

DO NOT use the emulator probe without using a pin protector. The pin protector is provided to prevent damage to the emulator probe when connecting and removing the emulator probe from the target system PGA socket.

Note

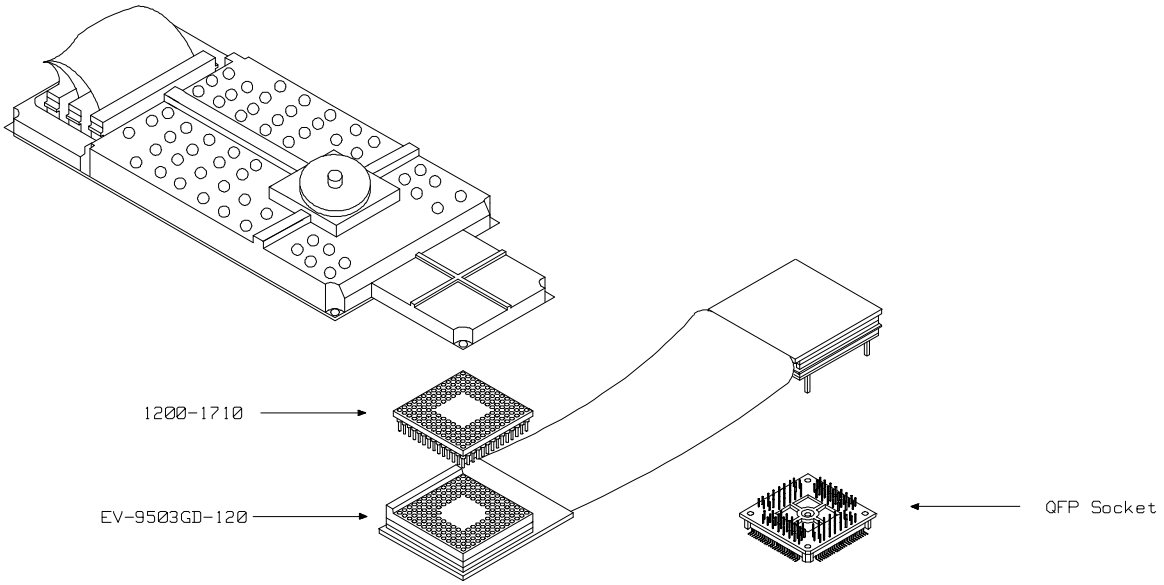
PGA-PGA flexible extender. You can use PGA-PGA flexible extender. When you want to use PGA-PGA flexible extender, you must order E3426A.

Installing into a QFP Type Socket

To connect the 70732 emulator probe to the QFP socket on the target system, use the NEC EV-9503-GD-120.

1. Attach the QPF socket to your target system.
2. Connect the NEC EV-9503-GD-120 to QPF socket on your target system.
3. Connect the IC-Socket(1200-1710) to the ZIP socket on NEC EV-9503-GD-120.
4. Place the 70732 emulator probe to the NEC EV-9503-GD-120 with IC-Socket.

Note Contact NEC Electronics Inc. to purchase QFP socket.



In-Circuit configuration Options

The 70732 emulator provides configuration options for the following in-circuit emulation issues. Refer to the "CONFIG_ITEM" section in the "70732 Emulator Specific Command Syntax" appendix.

Driving Background Cycles to the Target System

You can specify whether emulator bus cycles are driven to your target system bus when the emulator is in the background cycle. If your target system requires bus cycles activities constantly, such as BCYST, will need to drive the emulation bus cycles to your target system bus. By default, bus cycles are driven to the target system in background operation.

The configuration item is "**dbc**"

Allowing $\overline{\text{HLDRQ}}$ Signal from Target System

You can specify whether the emulator accepts or ignores the $\overline{\text{HLDRQ}}$ signal from target system. By default, the emulator accepts the $\overline{\text{HLDRQ}}$signal from the target system.

The configuration item is "**hld**"

Allowing $\overline{\text{NMI}}$ Signal from Target System

You can specify whether the emulator accepts or ignores the $\overline{\text{NMI}}$ signal from target system. By default, the emulator accepts the $\overline{\text{NMI}}$ signal from the target system.

The configuration item is "**nmi**"

Allowing $\overline{\text{READY}}$ SIgnal from Target System

You can specify whether the emulator accepts or ignores the $\overline{\text{READY}}$ signel from target system. By default, the emulator accepts the $\overline{\text{READY}}$ signal from the target system.

The configuration item is "**rdy**"

Allowing $\overline{\text{RESET}}$ SIgnal from Target System

You can specify whether the emulator accepts ot ignores the $\overline{\text{RESET}}$ signel from target system. By default, the emulator accepts the $\overline{\text{RESET}}$ signal from the target system.

The configuration item is "**rst**"

Allowing $\overline{\text{SZRQ}}$ SIgnal from Target System

You can specify whether the emulator accepts ot ignores the $\overline{\text{SZRQ}}$ signel from target system. By default, the emulator accepts the $\overline{\text{SZRQ}}$ signal from the target system.

The configuration item is "**szrq**"

Execution Topics

The descriptions in this section are of emulation tasks which involve program execution in general.

Run from Target System Reset

You can use **"r rst"** command to execute program from target system reset. If you use background monitor, you will see "**T>**" system prompt when you enter "r rst". In this status, the emulator accept target system reset. Then program stars if reset signal from target system is released.

If you use foreground monitor, reset signal from target system is always accepted.

Note

In the "Awaiting target reset" status(T>), you can not break into the monitor. If you enter "r rst" in the configuration that emulator ignores target system reset(cf rst=dis), you must reset the emulator.

The 70732 emulator supports power on reset. If you want program to be executed by power on reset, execute the following process.

1) Enter "r rst"

2) Turn OFF your target system

3) Turn On your target system

Note

When you turn OFF your target system, $\overline{\text{RESET}}$ signal must become low level before voltage become lower than 4V. When you turn ON your target system, $\overline{\text{RESET}}$ signal must be continued in low level for 20 clock cycles after voltage become upper than 4V.

Target ROM Debug Topics

The descriptions in this section are of emulation tasks which involve debugging target ROM. The tasks described below are made possible by the **cim** (copy target system memory image) command.

The **cim** command allows you to read the contents of target memory into the corresponding emulation memory locations. Moving target ROM contents into emulation memory is the key which allows you to perform the tasks described below. For example, if target ROM exists at locations 400H through 0A38H, you can copy target ROM into emulation memory with the following commands.

```
R>map 400..0bff erom
R>cim 400..0a38
```

Using Software Breakpoints with ROMed Code

You cannot define software breakpoints in target ROM memory. However, you can copy target ROM into emulation memory which does allow you to use software breakpoints.

Once target ROM is copied into emulation memory, software breakpoints may be used normally at addresses in these emulation memory locations.

```
R>bc -e bp
R>bp 440
```

Modifying ROMed Code

Suppose that, while debugging your target system, you begin to suspect a bug in some target ROM code. You might want to fix or "patch" this code before programming new ROMs. This can also be done by copying target system ROM into emulation memory with the **cim** (copy target memory image) command. Once the contents of target ROM are copied into emulation memory, you can modify emulation memory to "patch" your suspected code.

Pin State in Background

The probe pins of the emulator are in the following state. While the emulator is running in the background monitor, the pins state are different according to the configuration item "**cf dbc**".

Address Bus	Same as foreground
Data Bus	Always high impedance otherwise you direct the emulator to access target memory. When accessing target memory, I/O by background monitor, same as foreground.
$\overline{DA}$	
(cf dbc=en)	Same as foreground.
(cf dbc=dis)	Always high impedance otherwise you direct the emulator to access target memory. When accessing target memory, I/O by background monitor, same as foreground.
R/$\overline{W}$	Always high level, except accessing target memory, I/O by background monitor.
$\overline{BCYST}$	
(cf dbc=en)	Same as foreground.
(cf dbc=dis)	Always high level, except accessing target memory, I/O by background monitor.
Other	Same as foreground

Electrical Characteristics

The AC characteristics of the HP 64752A emulator are listed in the following table

Table 4-1 AC Electrical Specifications

Characteristic	Symbol	uPD70732 25MHz		HP 64752A			Unit
				Worst Case		Typical (*1)	
		Min	Max	Min	Max		
$\overline{\text{RESET}}$ Width Low	t_{hvr}	*2		*2			ns
$\overline{\text{RESET}}$ Setup Time	t_{srk}	10		17.5			ns
$\overline{\text{RESET}}$ Hold Time	t_{hrk}	10			9		ns
$\overline{\text{RESET}}$ Width Low	t_{wrl}	20		20			t_{cykr}
Cycle Time	t_{cykr}	40	125	40	125		ns
CLOCK Width High	t_{kkhr}	17		18			ns
CLOCK Width Low	t_{kklr}	17		18			ns
Input Setup Time(HLDRQ, ICHEEN, NMI)	t_{sik}	4		11.5			ns
Input Setup Time(INT, INTV0-3)	t_{sik}	4		5			ns
Input Setup Time(SZRQ)	t_{sik}	4		12.4		8.6	ns
Input Hold Time(HLDRQ, ICHEEN, NMI)	t_{hki}	4		3			ns
Input Hold Time(INI, INTV0-3)	t_{hki}	4		3			ns

Table 4-1 AC Electrical Specification(Cont'd)

Input Hold Time(SZRQ)	t_{hki}	4		9.2		0	ns
Input Setup Time(READY)	t_{sryk}	4		12.4		7.8	ns
Input Hold Time(READY)	t_{hkry}	4		9.2		0	ns
Input Setup Time(D0-31)	t_{sdk}	4		11.2		5.4	ns
Input Hold Time(D0-31)	t_{hkd}	4		3		0	ns
Output active delay(D0-31)	t_{dkd}	3	15	4	22.7	17.2	ns
Output inactive delay(D0-31)	t_{hkd}	3	15	4	22.7	17.5	ns
Output active delay(A1-31, $\overline{BE0}$-$\overline{3}$, $\overline{DA}$, R/$\overline{W}$, $\overline{MRQ}$, ST0-1, BLOCK, $\overline{BCYST}$)	t_{dka}	3	15	4	16	11.8	ns
Output inactive delay(A1-31, $\overline{BE0}$-$\overline{3}$, $\overline{DA}$, R/W, $\overline{MRQ}$, ST0-1, BLOCK, $\overline{BCYST}$)	t_{hka}	3	15	4	16	12.6	ns
Floating delay(D0-31)	t_{fkd}	3	20	4	27.2	20.6	ns
Active delay(D0-31)	t_{dkd}	3	20	4	27.2	19.8	ns
Floating delay(A1-31, $\overline{BE0}$-$\overline{3}$, $\overline{DA}$, R/$\overline{W}$, $\overline{MRQ}$, ST0-1, BLOCK, $\overline{BCYST}$)	t_{fka}	3	20	4	22		ns
Active delay(A1-31, $\overline{BE0}$-$\overline{3}$, $\overline{DA}$, R/$\overline{W}$, $\overline{MRQ}$, ST0-1, BLOCK, $\overline{BCYST}$)	t_{dka}	3	20	4	22		ns

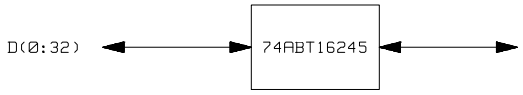
*1 Typical outputs measured with 20pF load
*2 20 t_{cykr} + 1us

4-18 In-Circuit Emulation

Target System Interface

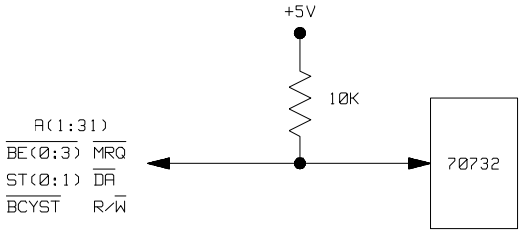
D0-D31

These signals are connected to 74ABT16245.



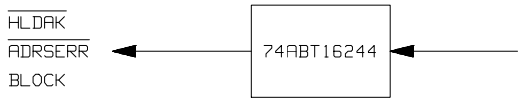
A(1:31)
BE(0:3) **BCYST**
DA **ST(0:1)**
R/W **MRQ**

These signals are connected to 70732 emulation processor through 10k ohm pull-up register.



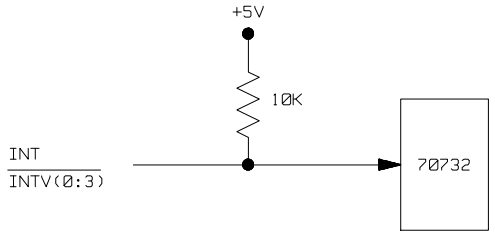
HLDAK **BLOCK**
ADRSERR

These signals are connected to 74ABT16244.



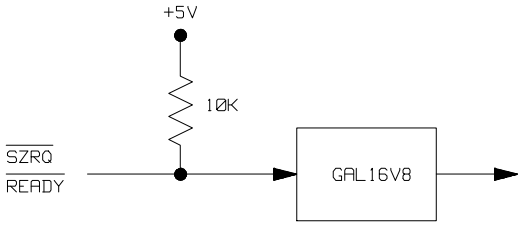
INT
INTV(0:3)

These signals are connected to 70732 emulation processor through 10k ohm pull-up register.



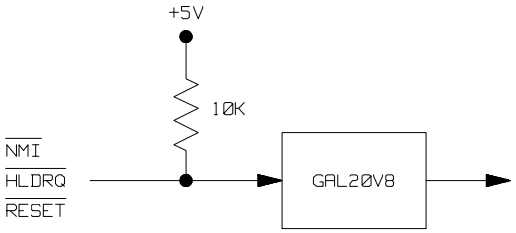
SZRQ
READY

These signals are connected to P16L8 through 10k ohm pull-up register.



HLDRQ NMI
RESET

These signals are connected to P20V8R through 10k ohm pull-up register.



A

70732 Emulator Specific Command Syntax

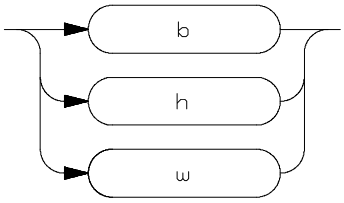
The following pages contain descriptions of command syntax specific to the 70732 emulator. The following syntax items are included (several items are part of other command syntax):

- <ACCESS_MODE>. May be specified in the **mo** (display and access mode), **m** (memory), and **io** (I/O port) commands. The access mode is used when the **m** or **io** commands modify target memory or I/O locations.
- <CONFIG_ITEMS>. May be specified in the **cf** (emulator configuration) and **help cf** commands.
- <DISPLAY_MODE>. May be specified in the **mo** (display and access mode), **m** (memory), **io** (I/O port), and **ser** (search memory for data) commands. The display mode is used when memory locations are displayed or modified.
- <REG_NAME> and <REG_CLASS>. May be specified in the **reg** (register) command.

ACCESS_MODE

Summary Specify cycles used by monitor when accessing target system memory or I/O.

Syntax



Function The **<ACCESS_MODE>** specifies the type of microprocessor cycles that are used by the monitor program to access target memory or I/O locations. When a command requests the monitor to read or write to target system memory or I/O, the monitor program will look at the access mode setting to determine whether byte or word instructions should be used.

Parameters

b **Byte.** Selecting the byte access mode specifies that the emulator will access target memory using byte cycles (one byte at a time).

h **Halfword.** Selecting the halfword access mode specifies that the emulator will access target memory using halfword cycles (one halfword at a time).

w **Word.** Selecting the word access mode specifies that the emulator will access target memory using word cycles (one word at a time).

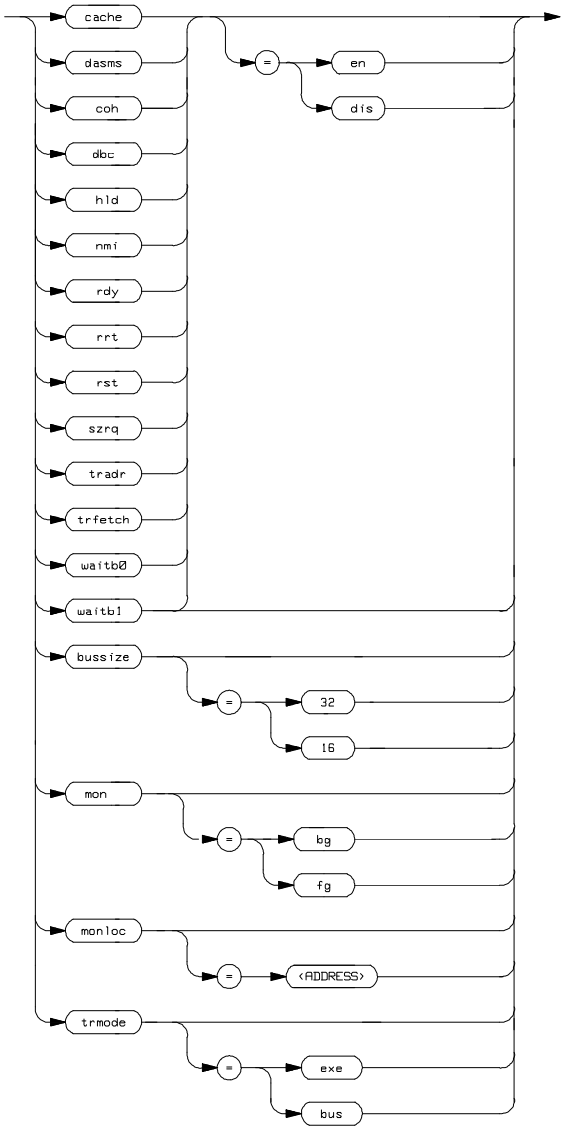
Defaults In the 70732, the <**ACCESS_MODE**> is **b** at power up initialization. Access mode specifications are saved; that is, when a command changes the access mode, the new access mode becomes the current default.

Related Commands **mo** (specify display and access modes)

CONFIG_ITEMS

Summary 70732 emulator configuration items.

Syntax



Function

The **<CONFIG_ITEMS>** are the 70732 specific configuration items which can be displayed/modified using the **cf** (emulator configuration) command. If the "=" portion of the syntax is not used, the current value of the configuration item is displayed.

Parameters

cache Instruction Cache. This configuration item allows you to specify whether enable or disable the instruction cache memory.

The analyzer can not trace transactions that are completed using the processor's instruction cache. Without these transactions, the analyzer may show confusing trace displays, or it may fail to trigger.

Setting **cache** equal to **en** specifies that the instruction cache is enabled.

Setting **cache** equal to **dis** specifies that the instruction cache is disabled.

The 70732 emulator is reset state after specifying this configuration item.

Note

The 70732 emulator operates in accordance with this configuration instead of ICHEEN signal from target system. ICHEEN signal from target system is ignored.

dasms **Disassemble Data Source.** This configuration item allows you to specify whether data are read from memory or from trace list in disassembling trace list.

Setting **dasms** equal to **dis** specifies that data are read from trace list when the emulator disassembles trace list.

Setting **dasms** equal to **en** specifies that data are read from memory when the emulator disassembles trace list.

Note

If you specify that "**cf tradr=en**" or "**cf trfetch=dis**", the data are read from memory regardless of this configuration, If you specify that "**cf trmode=bus**" or "**cf cache=en**", the data are read from trace list regardless of this configuration.

coh **Coherence of Instruction Cache and Memory.** This configuration item allows you to specify whether or not memory is coherent with instruction cache when the emulator modify memory.

Setting **coh** equal to **dis** specifies that cache will not be kept coherent with memory. The emulator does not check the cache contents when the emulator writes to the memory.

Setting **coh** equal to **en** specifies that cache will be kept coherent with memory. The emulator breaks into the monitor to keep cache coherence whenever the emulator writes to the memory.

Note

When you specify that "**cf rrt=en**" and "**cf coh=en**", the emulator can not modify emulation memory while the emulator is running the user program.

dbc **Bus Driven During Background Operation.** This configuration item allows you to specify whether emulator bus cycles are driven to your target system bus when the emulator is in background cycle. If your target system requires bus cycle activities constantly, you will need to drive the emulation bus cycles to your target system bus.

Setting **dbc** equal to **en** specifies that the emulator drives its bus cycles to target system bus whether or not the emulator executed in the background cycles.

Setting **dbd** equal to **dis** specifies that the emulator does not driven any bus cycles to target system bus in background operation.

The 70732 emulator is reset state after specifying this configuration item.

hld **Respond to Target Hold.** This configuration item allows you to specify whether or not the emulator accepts hold signal generated by the target system.

Setting **hld** equal to **en** specifies that the emulator accepts hold signal. When the hold is accepted, the emulator will respond as actual microprocessor.

Setting **hld** equal to **dis** specifies that the emulator ignores hold signal from target system.

nmi **Enable/disable user NMI.** This configuration item allows you to specify whether user NMI is accepted or ignored by the emulator.

Setting **nmi** equal to **en** specifies that the emulator accepts user NMI.

Setting **nmi** equal to **dis** specifies that the emulator ignores user NMI.

The 70732 emulator is reset state after specifying this configuration item.

Note

When target NMI signal is enabled , it is in effect while the emulator is running in the target program. While the emulator is running background monitor, NMI will be suspended until the monitor is finished.

---

rdy — **Allow Target Ready Signals.** This configuration item allows you to specify whether the emulator should honor target system ready signals on accesses to emulation memory.

Setting **rdy** equal to **en** specifies that target ready signals be honored on emulation memory accesses.

Setting **rdy** equal to **dis** specifies that target ready signals be ignored on emulation memory accesses.

rrt — **Restrict to Real-Time Runs.** This configuration item allows you to specify whether program execution should take place in real-time or whether commands should be allowed to cause breaks to the monitor during program execution.

Setting **rrt** equal to **en** specifies that the emulator's execution is restricted to real-time. In this setting, commands which access target system resources (display registers, display/modify target system memory or I/O) are not allowed.

setting **rrt** equal to **dis** specifies that the emulator breaks to the monitor during program execution.

rst **Respond to Target Reset.** This configuration item allows you to specify whether or not the emulator responds target system reset while running in user program or waiting for target system reset.

While running in background monitor, the 70732 emulator ignores target system reset completely independent on this setting.

Setting**rst** equal to **en** specifies that the emulator responds to reset from target system. In this configuration, emulator will accept reset and execute from reset vector in the same manner as actual microprocessor after reset is inactivated.

Setting **rst** equal to **dis** specifies that the emulator ignores reset from target system.

The 70732 emulator is reset state after specifying this configuration item.

Note

When you use the **r rst** (run from reset) command in-circuit to run form processor reset after the target reset input, you must use **"cf rst=en"** configuration setting.

---

szrq **Respond to Bus Size Request.** This configuration item allows you to specify whether or not the emulator responds bus size request signal from target system.

Setting **szrq** equal to **en** specifies that the emulator accepts bus size request signal from target system. In this setting, mapping attribute is ignored.

Setting **szrq** equal to **dis** specifies that the emulator ignores bus size request signal from target system. In this setting, the emulator operates according to mapping attribute.

tradr **Tracing Bus Address.** This configuration item allows you to specify whether or not forcing the analyzer to trace address of bus cycles as its data.

Setting **tradr** equal to **dis** specifies that the analyzer traces execution address, bus data and bus status when execution state and bus state occurs simultaneously. The analyzer trace bus address when bus state only occurs.

Setting **tradr** equal to **en** specifies that the analyzer traces execution address, bus address as its data and bus status when execution state and bus state occurs simultaneously. The analyzer traces bus address and bus status when bus state only occures.

trfsh **Trace Refresh cycles.** This configuration item allows you to specify whether or not the analyzer trace the 70732 emulation processor's refresh cycles.

Setting **trfsh** equal to **en** specifies that the analyzer will trace 70732 refresh cycles.

Setting **trfsh** equal to **dis** specifies that the analyzer will not trace 70732 refresh cycles.

Note

If you specify that "**cf trmode=bus**" or "**cf cache=en**", the analyzer will trace fetch cycles regardless of this configuration

---

waitb0

Wait Cycle for Bank0 This configuration item allow you to specify whether or not the emulator insert wait state when bank0 memory is accessed.

Setting **waitb0** equal to **en** specifies that 1 wait cycle is inserted when emulation memory of bank 0 is accessed. When you use HP64171A/B memory modules and clock speed is above 20MHz, you must specify "**waitb0=en**"

Setting **waitb0** equal to **dis** specifies that no wait cycle is inserted when emulation memory of bank 0 is accessed.

The 70732 emulator is reset state after specifying this configuration item.

waitb1

Wait Cycle for Bank1 This configuration item allow you to specify whether or not the emulator insert wait state when bank1 memory is accessed.

Setting **waitb1** equal to **en** specifies that 1 wait cycle is inserted when emulation memory of bank 1 is accessed. When you use HP64171A/B memory modules and clock speed is above 20MHz, you must specify "**waitb1=en**"

Setting **waitb1** equal to **dis** specifies that no wait cycle is inserted when emulation memory of bank 1 is accessed.

The 70732 emulator is reset state after specifying this configuration item.

Note

Accesses to emulation memory require 0 or 1 wait state depending upon the speed of the target system's clock and the memory module. The following table shows whether you need to insert 1 wait on emulation memory accesses.

frequency of the external clock	Memory Module HP64171A/B (35ns)	HP64172A/B (20ns)
20MHz or less	no-wait	no-wait
above 20MHz	1-wait	no-wait

---

bussize

Data Bus size. This configuration item allows you to specify whether the data bus size is to be 16(data bus size is 16 bits) or 32(data bus size is 32 bits).

Setting **bussize** equal to **32** specifies that data bus width is 32 bit.

Setting **bussize** equal to **16** specifies that data bus width is 16 bit.

The 70732 emulator is reset state after specifying this configuration item.

Note

The 70732 emulator operates in accordance with this configuration instead of SZ16B signal from target system. SZ16B signal from target system is ignored.

---

mon

Monitor Options. This configuration item is used to select the type of monitor to be used by the emulator.

Setting **mon** equal to **bg**(background monitor) specifies that all monitor functions are performed in background.

Setting **mon** equal to **fg**(foreground monitor) specifies that all monitor functions are performed in foreground.

The 70732 emulator is reset after specifying this configuration item.

monloc

Monitor Location This configuration item allows you specify location of monitor program. The monitor must be located on a 8K boundary.

If you use background monitor, setting this configuration specifies that driven address to target system in background cycle. The low 16 bits of address are actual address of the 70732 microprocessor.

If you use foreground monitor, setting this configuration specifies that start address of foreground monitor program.

The 70732 emulator is reset after specifying this configuration item. Refer to the "Using the Optional Foreground Monitor" appendix in this manual.

trmode

Trace Mode. This configuration item allows you to specify whether or not the analyzer trace execution address.

Setting **trmode** equal to **bus** specifies that the analyzer traces bus access only.

Setting **trmode** equal to **exe** specifies that the analyzer trace bus access and execution address. When bus state and execution state occures simultaneously, the analyzer can not trace bus address. Refer to "**cf tradr**" configuration item.

If you specify "**cf cache=en**", the analyzer traces bus access only regardless of this configuration.

Defaults

The default values of 70732 emulator configuration items are listed below.

```
cf bussize=32
cf cache=en
cf coh=dis
cf dasms=dis
cf dbc=en
cf hld=en
cf mon=bg
cf monloc=0
cf nmi=en
cf rdy=en
cf rrt=dis
cf rst=en
cf szrq=en
cf tradr=dis
cf trfetch=en
cf trmode=exe
cf waitb0=en
cf waitb1=en
```

Related Commands

help

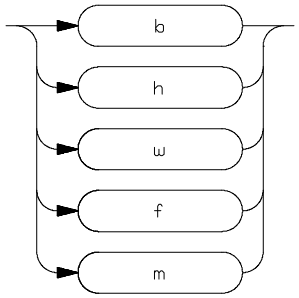
You can get an on line help information for particular configuration items by typing:

```
R>help cf <CONFIG_ITEM>
```

DISPLAY_MODE

Summary Specify the memory display format or the size of memory locations to be modified.

Syntax



Function The **<DISPLAY_MODE>** specifies the format of the memory display or the size of the memory which gets changed when memory is modified.

Parameters

b	**Byte**. Memory is displayed in a byte format, and when memory locations are modified, bytes are changed.
h	**Halfword**. Memory is displayed in a halfword format, and when memory locations are modified, halfwords are changed.
w	**Word**. Memory is displayed in a word format, and when memory locations are modified, words are changed.
f	**Float**. Memory is displayed in a short-float format, and when memory locations are modified, short-floats are changed. When this display mode used, the current default display mode is not changed.

m	**Mnemonic**. Memory is displayed in mnemonic format; that is, the contents of memory locations are inverse-assembled into mnemonics and operands. When memory locations are modified, the last non-mnemonic display mode specification is used. You cannot specify this display mode in the **ser** (search memory for data) command.

Defaults

At powerup or after init,in the 70732 Emulator, the **<ACCESS_MODE>** and **<DISPLAY_MODE>** are **b**.

Display mode specifications are saved; that is, when a command changes the display mode, the new display mode becomes the current default.

Related Commands

mo (specify access and display modes)

m (memory display/modify)

io (I/O display/modify)

ser (search memory for data)

REGISTER CLASS and NAME

Summary 70732 register designator. All available register class names and register names are listed below.

<REG_CLASS>

<REG_NAME>	Description	
***(All basic registers)**		
pc psw **r0 r1 r2 r3 r4** **r5 r6 r7 r8 r9** **r10 r11 r12 r13** **r14 r15 r16 r17** **r18 r19 r20 r21** **r22 r23 r24 r25** **r26 r27 r28 r29** **r30 r31**	BASIC registers.	
sys(System Control registers)		
eipc	Exception/Interrupt PC	
eipsw	Exception/Interrupt PSW	
fepc	Fatal error PC	
fepsw	Fatal error PSW	
ecr	Exception cause	(Read Only)
pir	Processor ID	(Read Only)
tkcw	Task control word	(Read Only)
chcw	Cache control word	
adtre	Address trap	

Function

The **<REG_CLASS>** names may be used in the **reg**(register) command to display a class of 70732 registers.

The **<REG_NAME>** names may be used with the **reg** command to either display or modify the contents of 70732 registers.

Refer to your 70732 use's manual for complete details on the use of the 70732 registers.

Related Commands

reg (register display/modify)

B

Using the Optional Foreground Monitor

By using and modifying the optional Foreground Monitor, you can provide an emulation environment which is customized to the needs of a particular target system.

Comparison of Foreground and Background Monitors

An emulation monitor is required to service certain requests for information about the target system and the emulation processor. For example, when you request a register display, the emulation processor is forced into the monitor. The monitor code has the processor dump its registers into certain emulation memory locations, which can then be read by the emulator system controller without further interference.

Background Monitors

A *background* monitor is an emulation monitor which overlays the processor's memory space with a separate memory region.

Usually, a background monitor is easier to work. The monitor is immediately available upon powerup, and you don't have to worry about linking in the monitor code or allocating space for the monitor. No assumptions are made about the target system environment; therefore, you can test and debug hardware before any target system code has been written. All of the processor's address space is available for target system use, since the monitor memory is overlaid on processor memory, rather than subtracted from processor memory. Processor resources such as interrupts are not fully taken by the background monitor.

However, all background monitors sacrifice some level of support for the target system. For example, while the emulation processor enters the monitor code to display registers, it will not respond to target system interrupt requests. This may pose serious problems for

applications that rely on the microprocessor for real-time, non-intrusive support. Also, the background monitor code resides in emulator firmware and can't be modified to handle special conditions.

Foreground Monitors

A *foreground* monitor may be required for more interrupt intensive applications. A foreground monitor is a block of code that runs in the same memory space as your program. You link this monitor into your code so that when control is passed to monitor program, the emulator can still service real-time events, such as interrupts or watchdog timers. For most multitasking, you will need to use a foreground monitor.

You can tailor the foreground monitor to meet your needs, such as servicing target system interrupts. However, the foreground monitor does use part of the processor's address space, which may cause problems in some applications. You must also properly configure the emulator to use a foreground monitor (see the "Emulation topics" chapter and the examples in this appendix).

Using Built-in Foreground Monitor

The 70732 emulator includes foreground monitor. The built-in foreground monitor saves your tasks for assembling, linking and loading the monitor program. To use the built-in foreground monitor, all you have to do is to specify the location of the monitor. The location is specified by the configuration item "**cf monloc**". Specify the monitor location as follows.

```
R>cf monloc=<address>
```

After you issued the configuration command, the built-in foreground monitor is set up automatically.

Using Custom Foreground Monitor

The custom foreground monitor allows you customize the monitor for your target system. To use the monitor, you need to assemble, link and load the monitor program into emulator.

If you use NEC K&R-C compiler, you must modify the link directive file 'fm70732.d' to use the monitor. In this example, the monitor will be located at 20000 hex, so the modified link detective file like this:

```
TEXT :  !LOAD   ?RX   V0x00020000 {
        .text =  $PROGBITS ?AX;
};
```

If you use Green Hills Software C-Compiler, you must modify the following statement of the monitor program 'gfm70732.s' to use the monitor. In this example , the monitor will be located 20000 hex, so the modified statement looks like this:

```
MON_ADDR        =        0x000020000
```

You can load the monitor at any base address on a 8k byte boundary.

Assemble and Link the monitor

If you use NEC K&R-C Compiler, you can assemble and link the foreground monitor program with the following commands:

```
$ as732 fm70732.s <RETURN>
$ ld732 -D fm70732.d -o fm70732.abs
fm70732.o <RETURN>
$ v810cnv -x fm70732 <RETURN>
```

If you use Green Hills Software C-Compiler, you can assemble and link the foreground monitor program with the following commands:

```
$ as810 gfm70732.s -o gfm70732.o <RETURN>
$ lx -sec @gfm70732.d -o gfm70732.x
gfm70732.o <RETURN>
$ v810cnv -x gfm70732.x <RETURN>
```

You need to tell the emulator that you will be using a foreground monitor and allocate the memory space for the monitor. This is all done with one configuration command. To locate the monitor on a 8k boundary starting at 20000 hex, type:

```
R> cf monloc=0x20000
```

Load the Foreground Monitor

Now it's time to load the sample program and monitor. In the example shown, we're loading the program from a host with the emulator in Transparent Configuration. If you're using the standalone configuration with a data terminal, you will need to enter the data using the **m** command. (You can get the data from your assembly listings.) Load the program by typing:

```
R> load -hbs "transfer -tb fm70732.X"
```

##############

An Example Using the Foreground Monitor

In the following example, we will show how the emulator switches from state to state using a foreground monitor.

Mapping Memory for the Example

When you specify a foreground monitor and enter the monitor address, all existing memory mapper terms are deleted and a term for the monitor block will be added. Add the additional term to map memory for the demo program.

Load the Sample Program

Assuming the sample program has been assembled and linked as shown in "Getting Start" chapter, you can load the sample program by typing:

```
R> load -hbs "transfer -tb cmd_rds.X"
```

#####

Set Analyzer Master Clock Qualifiers

We want to view the transitions made between the different emulator states; reset to break, break to run, run to break. Since the foreground monitor is actually entered via a few cycles in the emulator's built-in background monitor, we need to be able to view the background states.

We can do this by modifying the emulation analyzer's master clock qualifier to include tracing of background code. To see the initial clock qualifier, type:

```
M> tck
tck -r L -u -s S
```

Modify this as follows:

```
M> tck -r L -ub -s S
```

Now, reset the processor so we can make the first measurement from a known state:

```
M> rst
```

Reset to Break

We want to see the monitor's transition from the reset state to running in the foreground monitor. Since the foreground monitor occupies the address range from 20000 through 21fff hex, we can simply trigger on any access to that range:

```
R> tg addr=20000..21fff
```

We also want see the states leading up to the transition between reset and foreground monitor execution. We can position the trigger so that there are 20 states **before** the trigger as follows:

```
R> tp -b 15
```

Start the measurement:

```
R> t
Emulation trace started
```

Now, break the emulator into the monitor:

```
R> b
```

Display 20 disassembled states of the trace from the top the trace:

```
M> tl -td 15
```

```
 Line   addr,H    N70732 Mnemonic                          count,R
-----   --------  -----------------------------------  ---------
  -12   fffffff8  OUT.W   R31,-0x0001[R31]        BGM         ---
  -11   00000000    00008000 data write word      BGM    0.360 uS
  -10   00000004    fffffff0 data write word      BGM    0.080 uS
   -9   ffffffe0  ST.W    R31,0x00ac[R0]          BGM    0.440 uS
   -8   ffffffe4  ST.W    R29,0x00a4[R0]          BGM    0.080 uS
   -7   ffffffe8  LD.W    0x0200[R0],R29          BGM    0.080 uS
   -6   ffffffec  JMP     [R29]                   BGM    0.080 uS
       =fffffffe  MOV     R0,R0                   BGM
   -5   000000ac    00000001 data write word      BGM    0.080 uS
   -4   000000a4    00020000 data write word      BGM    0.080 uS
   -3   00000200    00020300 data read word       BGM    0.080 uS
   -2   fffffff0  MOV     R0,R0                   BGM    0.080 uS
       =fffffff2  MOV     R0,R0                   BGM
   -1   fffffff4  MOV     R0,R0                   BGM    0.080 uS
       =fffffff6  MOV     R0,R0                   BGM
    0   00020300  MOVEA   0x0300,R0,R31           BGM    0.080 uS
    1   00020304  SUB     R31,R29                 BGM    0.080 uS
       =00020306  LD.W    0x0004[R29],R31         BGM
    2   00020308    dffd0004 fetch                BGM    0.080 uS
```

At line -11, the processor began executing code; it executed in the background monitor. To see the transition from background execution to foreground monitor program execution, type:

```
M> tl 15..25
```

```
 Line   addr,H    N70732 Mnemonic                          count,R
-----   --------  -----------------------------------  ---------
   15   00021a10  NOP                              BGM    0.120 uS
       =00021a12  BRKRET  0x01                     BGM
   16   00021a14  MOV     R0,R0                   BGM    0.080 uS
       =00021a16  MOV     R0,R0                   BGM
   17   00021a18  MOV     R0,R0                   BGM    0.080 uS
       =00021a1a  MOV     R0,R0                   BGM
   18   00000000    00008000 data read word       BGM    0.360 uS
   19   00000004    00020380 data read word       BGM    0.080 uS
   20   00020380  LD.B    0x00f8[R29],R31                0.280 uS
   21   00020384  CMP     0x00,R31                       0.080 uS
       =00020386  JNZ/JNE 0x0002041e
   22   00020388  ST.W    R0,0x0030[R29]                 0.080 uS
   23   000200f8    ......01 data read byte              0.080 uS
   24   0002038c  ST.W    R1,0x0034[R29]                 0.080 uS
   25   0002041c    dc1d0010 unused fetch                0.200 uS
       =0002041e  ST.W    R0,0x00a4[R29]
```

The foreground monitor start at states 20.

Monitor to User Program

We can look at the transition from the foreground monitor to running the user program by triggering the trace on a user program address. Type:

```
M> tg addr=10000
```

We will leave the trigger position where it was for the last measurement(20 states are retained before the trigger position). Start the measurement:

```
M> t
Emulation trace started
```

Now, run the sample program:

```
M> r 10000
```

Display trace states from -15 to +5 in inverse-assembled form as follows:

```
U> tl -d -10..5
```

```
  Line   addr,H    N70732 Mnemonic                         count,R
 -----   --------  ------------------------------------  ---------
   -10   00021a08  LD.W    0x0038[R29],R2                  0.080 uS
    -9   00021a0c  LD.W    0x00a4[R29],R29                 0.080 uS
    -8   00021a10  NOP                                     0.080 uS
        =00021a12  BRKRET  0x01
    -7   00020020    ......04 data write byte              0.080 uS
    -6   00020038    00000000 data read word               0.080 uS
    -5   000200a4    00000000 data read word               0.080 uS
    -4   00021a14  MOV     R0,R0                           0.080 uS
        =00021a16  MOV     R0,R0
    -3   00021a18  MOV     R0,R0                           0.080 uS
        =00021a1a  MOV     R0,R0
    -2   00000000    00008000 data read word               0.360 uS
    -1   00000004    00010000 data read word               0.080 uS
     0   00010000  MOVHI   0x0000,R0,R1                    0.320 uS
     1   00010004  MOVEA   0x0634,R1,R3                    0.120 uS
     2   00010008  MOVHI   0x0000,R0,R1                    0.120 uS
     3   0001000c  MOVEA   0x0500,R1,R4                    0.120 uS
     4   00010010  ST.W    R0,0x0000[R4]                   0.120 uS
     5   00010014  MOVHI   0x0000,R0,R1                    0.120 uS
```

At state -8 in the trace listing, the processor executed the **BRKRET** instruction to transfer execution to the user program at state 0.

User Program Run to Break

You can trace the execution from the user program run to the foreground monitor due to a break condition by setting as follows:

```
U> tg stat=bg
```

Start the measurement:

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

Satisfy the trigger condition by break the emulator into the monitor:

```
U> b
```

Now, display trace states from -10 to +10 in disassembled form as follows:

```
M> tl -5..5
```

At state 0 of the trace list, the processor entered the background monitor to make the transition. And actual foreground monitor program start at after several background monitor execution.

```
 Line   addr,H    N70732 Mnemonic                                  count,R
-----   --------  -----------------------------------  ---------
   -5   0001001c  LD.B    0x0000[R4],R6                 0.120 uS
   -4   00010020  CMP     0x00,R6                       0.120 uS
       =00010022  JZ/JE   0x0001001c
   -3   00000500    ......00 data read byte             0.120 uS
   -2   00010024  MOVEA   0x0041,R0,R1                  0.120 uS
   -1   00010028  CMP     R1,R6                         0.120 uS
       =0001002a  JZ/JE   0x00010038
    0   00000000    00008001 data write word      BGM    0.320 uS
    1   00000004    00010022 data write word      BGM    0.080 uS
    2   fffffe0  ST.W    R31,0x00ac[R0]          BGM    0.440 uS
    3   fffffe4  ST.W    R29,0x00a4[R0]          BGM    0.080 uS
    4   fffffe8  LD.W    0x0200[R0],R29          BGM    0.080 uS
    5   ffffffec  JMP     [R29]                  BGM    0.080 uS
       =ffffffee  MOV     R0,R0                  BGM
```

To see the starting point of foreground monitor, type:

```
M> tl 25..35
```

```
 Line   addr,H    N70732 Mnemonic                          count,R
-----   --------  -----------------------------------  ---------
   25   00020324  MOV     R0,R0                    BGM    0.080 uS
       =00020326  MOV     R0,R0                    BGM
   26   00021a10  NOP                              BGM    0.120 uS
       =00021a12  BRKRET  0x01                     BGM
   27   00021a14  MOV     R0,R0                    BGM    0.080 uS
       =00021a16  MOV     R0,R0                    BGM
   28   00021a18  MOV     R0,R0                    BGM    0.080 uS
       =00021a1a  MOV     R0,R0                    BGM
   29   00000000    00008001 data read word        BGM    0.360 uS
   30   00000004    00020380 data read word        BGM    0.080 uS
   31   00020380  LD.B    0x00f8[R29],R31                  0.280 uS
   32   00020384  CMP     0x00,R31                         0.080 uS
       =00020386  JNZ/JNE 0x0002041e
   33   00020388  ST.W    R0,0x0030[R29]                   0.080 uS
   34   000200f8    ......00 data read byte               0.080 uS
   35   0002038c  ST.W    R1,0x0034[R29]                   0.080 uS
```

At state 31, the foreground monitor program starts.

Limitations of Foreground Monitors

Synchronized measurements

You cannot perform synchronized measurements over the CMB when using a foreground monitor. If you need to make such measurements, use background monitor.

Notes

Index