HP 64774

29000/29050 Emulator

PC Interface User's Guide

HEWLETT PACKARD

HP Part No. 64774-97010
Printed in U.S.A.
June 1992

Edition 4

Notice

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

Printing History

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

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

Edition 1	64774-97002, June 1990
Edition 2	64774-97004, August 1990
Edition 3	64774-97007, May 1991
Edition 4	**64774-97010, June 1992**

Using this Manual

This manual shows you how to use the HP 64774 29000/29050 emulator with the PC Interface.

This manual:

- Lists the features of the 29000/29050 emulator.
- Shows you how to use emulation commands.
- Describes the target system design considerations that you must make when using the 29000/29050 emulator.
- Shows you how to connect the emulator to the target system.
- Shows you how to configure the emulator for your development needs.

This manual does not show you how to use every PC Interface command and option. This is done in the *HP 64700-Series Emulators PC Interface Reference.*

Organization

Chapter 1 **Introduction**. This chapter describes the emulator functions and lists its basic features.

Chapter 2 **Getting Started**. This chapter shows you how to use emulation commands to view the execution of a sample program.

Chapter 3 **Using the Emulator — The Basics**. This chapter describes target system design considerations that you must make when using the 29000/29050 emulator. It also shows you how to connect the emulator to the target system.

Chapter 4 **Using the Emulator — In Depth**. This chapter describes some of the emulation features in more detail. Topics include the memory mapper and emulation memory, and coordinated measurements.

Chapter 5 **Configuring the Emulator**. This chapter describes the options available for configuring the 29000/29050 emulator. It also describes storing and loading the emulator configuration.

Appendix A **Using the HP 64000 Reader**. This appendix describes the HP 64000 reader.

Appendix B **Using the IEEE-695 Reader**. This appendix describes the IEEE-695 reader.

Contents

Illustrations

Tables

1

Introduction to the 29000/29050 Emulator

Purpose of the Emulator

The HP 64774 29000/29050 emulator is designed to replace the Am29000/Am29050 microprocessor in your target system to help you integrate target system software and hardware. The emulator acts like the processor that it replaces and gives you information about the operation of the processor. You can control target system execution and view or modify the contents of processor registers, target system memory, and I/O resources.

Description

The HP 64774 emulator supports the Am29000 and Am29050 microprocessors. The HP 64774 is a self-contained emulation and analysis system that can contain up to 4 Mbytes of emulation memory.

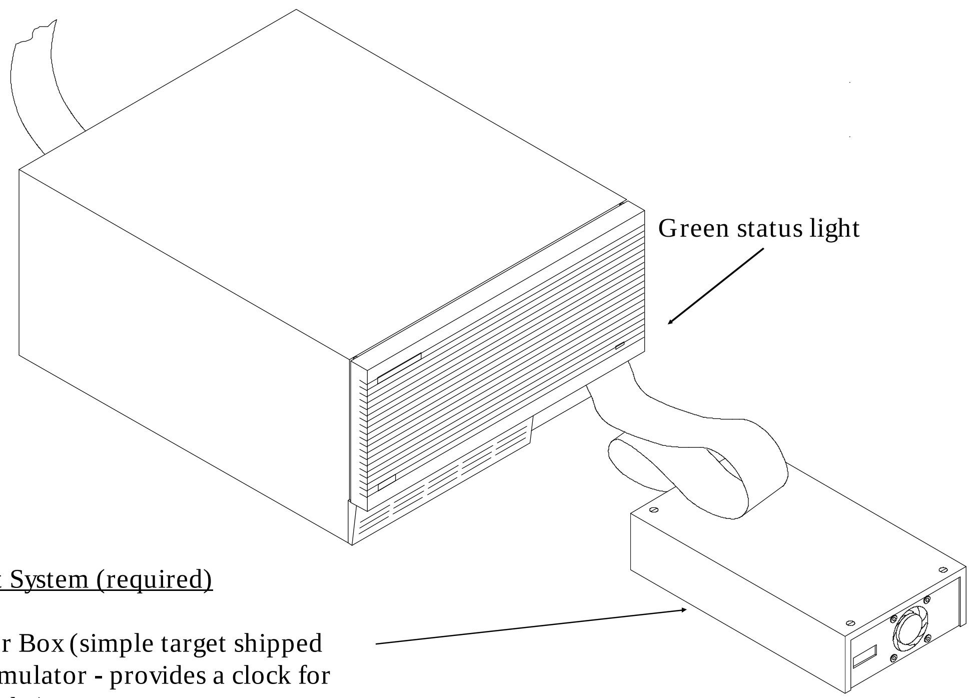

Figure 1-1. The HP 64774 Emulator for the 29000/29050

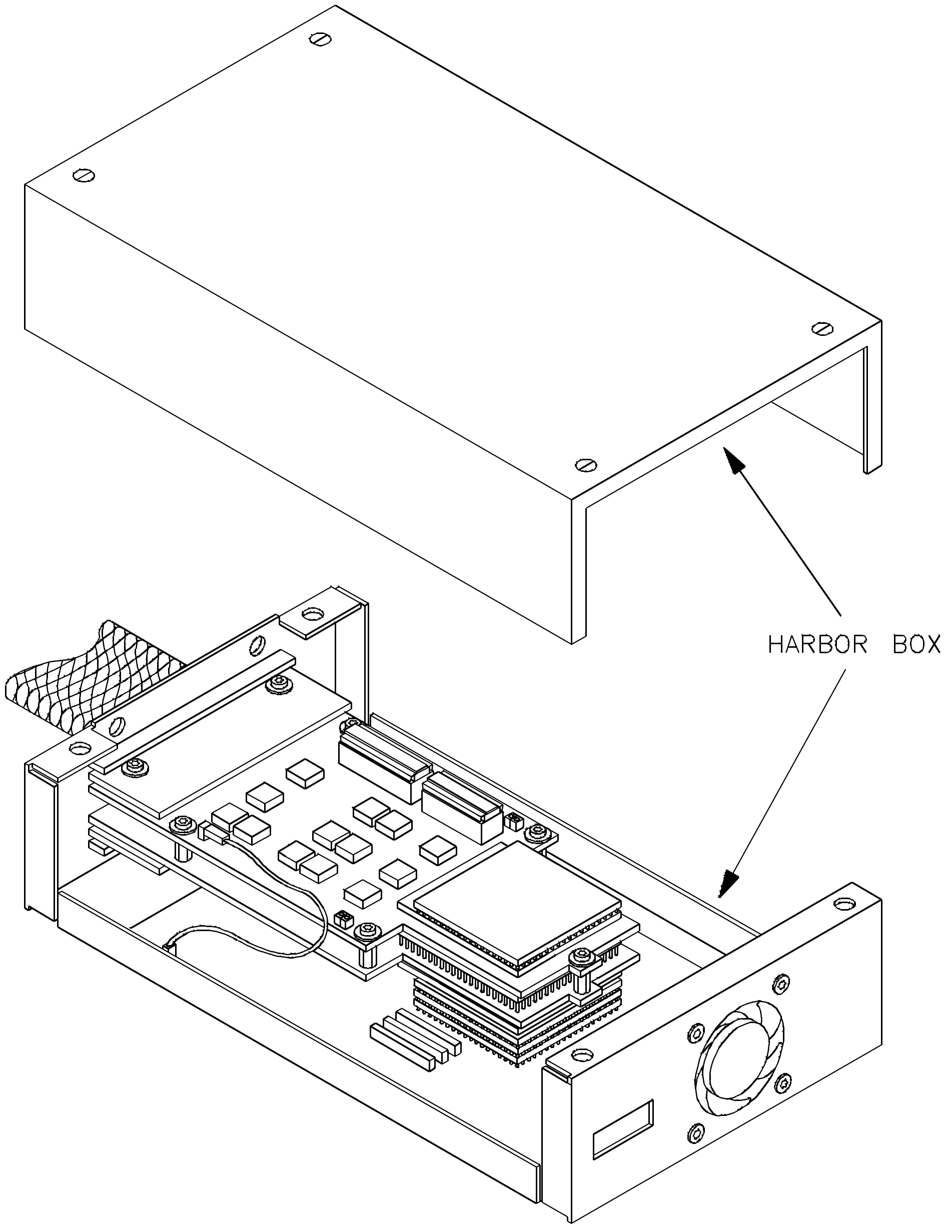

Figure 1-2. 29000/29050 Probe Plugged into Harbor Box

Features of the 29000/29050 Emulator

This section introduces the features of the HP 64774 29000/29050 emulator.

Full-Featured Operation at 25 MHz

Note

The 29000/29050 emulator can execute in a target system at full clock speed (33 MHz); however, the analyzer may provide incorrect data above 25 MHz. As a result, Hewlett-Packard only supports operation of the HP 64774 emulator with analysis at clock speeds up to 25 MHz.

Operation of the emulator at high clock speeds is achieved with:

- Active-probe technology.
- Unbuffered instruction and data busses.
- A mode that bypasses the memory mapper.

Active-Probe Technology

Active-probe technology allows the emulator to closely imitate the electrical characteristics of the microprocessor, avoiding the timing problems that can occur with passive probes.

You need a target system clock to use the emulator. The emulator is shipped with a “harbor box” assembly around the active probe connector, which provides the target system clock when you are operating the emulator out-of-circuit. It also suppresses radio frequency interference (RFI), and cools the processor in the probe.

Unbuffered Instruction and Data Busses

The instruction and data busses at the target system are unbuffered to achieve high-speed emulator operation in the target system. The emulator provides control signals to the target system to tell the target system data bus buffers when one or both of the busses must be tristated.

Memory Mapper Bypass Mode

Memory mapper circuitry affects signal timing. To allow the processor to execute in target memory above 25 MHz without wait states, the emulator has a mode that bypasses the mapper circuitry. (No emulation memory can be used in the bypass mode.) In this mode, the emulator must execute out of target memory.

Because large amounts of emulation memory cannot be put in the probe, and because the probe cabling affects timing, wait-states are generated when executing out of emulation memory.

Single-Step and Disassembly of Instruction Bus

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

While the 29000/29050 processor will fetch instructions from memory connected only to its I-bus, it cannot explicitly read it. However, the HP 64774 emulator has circuitry to read this memory so that it can both single-step and disassemble instructions without a D-bus connection.

Product Upgrades

Because the HP 64774 contains programmable parts, you can reprogram the emulator firmware without disassembling the emulator. This means that you can update product firmware without having to call an HP field representative to your site.

Emulation Memory

Emulation memory is multi-ported memory. This allows the emulator to run in real-time while you enter emulation commands that access emulation memory.

Emulation Memory Size

Emulation memory is available in 0.5 Mbyte or 2 Mbyte block with the HP 64774Y 0.5 Mbyte 29000/29050 Emulation Memory Module or HP 64774Z 2 Mbyte 29000/29050 Emulation Memory Module. Zero, one, or two of either module may be plugged into the 29000/29050 emulator card. The emulator supports the following configurations:

- 0.0 M bytes
- 0.5 M bytes
- 1.0 M bytes
- 2.0 M bytes
- 2.5 M bytes
- 4.0 M bytes

Memory in the 2 Mbyte memory module has a slower access time (35 ns) than memory in the 0.5 Mbyte memory module (25 ns). When you mix these modules, the emulator treats both modules as the slower memory.

You can see how much memory is installed in your HP 64774 by executing the Terminal Interface **map** command.

Independent Banks of Emulation Memory

If the emulator contains two banks of emulation memory, code space may be mapped in one bank and data space in the other. This allows simultaneous fetching of both instruction and data bus information - eliminating the need for instruction/data bus arbitration that might affect emulator performance.

Memory Mapping

You can map up to 15 memory ranges. Mapped ranges must be at least 64 Kbytes long. You can characterize memory ranges as emulation RAM or ROM, target system RAM or ROM, or as guarded memory. Additionally, you must select the bank and block of emulation memory into which the address range is mapped.

The emulator issues an error message and breaks execution into the monitor when guarded memory is accessed.

Also, you can enable a break condition that causes emulator execution to break out of the user program (into the emulation monitor program) when writes to memory mapped as ROM occur.

Fast Upload/Download with RS-422 Card

The RS-422 capability of the emulator's A communication port and an RS-422 interface card on the host computer (for example, the HP 64037 for the PC) provide upload/download rates of up to 230.4K baud.

Coverage Measurements

Coverage memory is provided along with emulation memory. This memory allows you to make code coverage measurements, as well as measurements that determine maximum stack sizes.

Analysis

The analyzer supplied with the emulator, called the *emulation analyzer*, captures emulator bus cycle information and bus cycle states synchronously with the processor clock.

Note

No external analysis (capability to probe signals external to the emulator) is available with the HP 64774 emulator.

See the *HP 64700 Emulators PC Interface: Analyzer User's Guide* for a complete list of analyzer features.

Floating Point Format Displays

The 29000/29050 emulator has commands that allow you to display memory and registers in floating point formats.

Register Display and Modification

You can display or modify the contents of one or more registers in the 29000/29050. The register display command allows the display and modification of multiple registers. Also, the bit fields of various registers are labeled in the default display. An option allows you to display only the full hex value of the registers.

Breakpoints

You can set up the emulator/analyzer interaction so that when the analyzer finds a specific state, emulator execution will break out of the user program into the emulation monitor. These are called hardware breakpoints. Note that the analyzer monitors bus activity, which may not correspond to execution, because the 29000/29050 is a pipelined processor.

You can also define software breakpoints in your program. When you define a software breakpoint and the opcode at the breakpoint address is executed, the emulator breaks into background.

Reset Support

You can reset the emulator from the emulation system, or your target system can reset the emulation processor.

Real-Time Execution

Real-time operation means continuous execution of your program without interference from the emulator. Such interference occurs when the emulator temporarily breaks into the monitor so that it can access register contents or target system memory.

You can restrict the emulator to real-time execution. When the emulator is executing your program in real-time, commands that display or modify registers or target system memory are not allowed.

2

Getting Started

Introduction

This chapter is a tutorial that shows how to use the HP 64774 29000/29050 emulator with the PC Interface.

This chapter will:

- Tell you what to do before you use the emulator in the tutorial examples.
- Describe the sample program used for the examples.
- Briefly describe how to enter PC Interface commands and how emulator status is displayed.

This chapter will show you how to:

- Start the PC Interface from the MS-DOS prompt.
- Define (map) emulation and target system memory.
- Load programs into emulation and target system memory.
- 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 keystroke macros, searching memory, running, breaking, using breakpoints, tracing program execution, changing the trace format, copying memory, and testing coverage.

Before You Begin

The examples in this chapter were performed with the 29000 the emulator plugged into the harbor box. Also, the emulator contained 0.5 Mbyte of emulation memory. Your emulator must have an emulation memory module to run the sample program.

Prerequisites

Before beginning the tutorial, you must do the following:

1. Connect the emulator to your computer. The *HP 64700 Series Emulators Installation Guide* shows you how to do this.

2. Install the PC Interface software on your computer. Software installation instructions are shipped with the media containing the PC Interface software. The *HP 64700 Emulators PC Interface Reference* manual contains more information on the installation and setup of the PC Interface.

3. You should read and understand the concepts of emulation presented in the *Concepts of Emulation and Analysis* manual. Understanding these concepts may help you avoid problems later.

4. You should read the *HP 64700 Emulators PC Interface Reference* manual to learn general PC Interface operation. It contains information specific to the 29000/29050 emulator.

A Look at the Sample Program

Figure 2-1 shows the sample program used in this chapter. The program is a primitive command interpreter.

Data Declarations

The first two lines in the “cmd_rdr.src” source file define the **Msgs**, **Init**, **Cmd_Input**, and **Msg_Dest** labels as global symbols.

```
Advanced Micro Devices, Inc. Am29000 Assembler - Version 2.0-5
Copyright (c) 1987, 1988 Microtec Research Inc.
ALL RIGHTS RESERVED.   Serial Number AS2002563

Advanced Micro Devices, Inc. ASM29K Assembler Rel. 2.0-5   Sat Mar 03 19:20:26 1990
Page 1


Cmdline - \USR\C29K\BIN\AS29.EXE -l -fgosx cmd_rdr.src
Line   Address
                                             .global     Init,Msgs,Cmd_Input
                                             .global     Msg_Dest

                                             .data
                                Msgs:
       00000000 43 6F 6D 6D    Msg_A:        .ascii      "Command A entered "
                61 6E 64 20
                41 20 65 6E
                74 65 72 65
                64 20
                                             .align      4
       00000014 45 6E 74 65    Msg_B:        .ascii      "Entered B command "
                72 65 64 20
                42 20 63 6F
                6D 6D 61 6E
                64 20
                                             .align      4
       00000028 49 6E 76 61    Msg_I:        .ascii      "Invalid Command "
                6C 69 64 20
                43 6F 6D 6D
                61 6E 64 20
                                             .align      4
                                End_Msgs:


                                             .text
                                ;***********************************************
                                ;* Clear previous command.
                                ;***********************************************
       00000000 03 00 41 00    Init:         const       gr65,0
       00000004 03 00 40 00  R               const       gr64,Cmd_Input
       00000008 02 00 40 00  R               consth      gr64,Cmd_Input
       0000000C 1E 00 41 40                  store       0,0,gr65,gr64
                                ;***********************************************
                                ;* Read command input byte.  If no command has
                                ;* been entered, continue to scan for input.
                                ;***********************************************
       00000010 16 00 41 40    Scan:         load        0,0,gr65,gr64
       00000014 61 42 41 00                  cpeq        gr66,gr65,0
       00000018 AC FF 42 FE                  jmpt        gr66,Scan
                                ;***********************************************
                                ;* A command has been entered.  Check if it is
                                ;* command A, command B, or invalid.
                                ;***********************************************
       0000001C 61 42 41 41    Exe_Cmd:      cpeq        gr66,gr65,0x41
       00000020 AC 00 42 05                  jmpt        gr66,Cmd_A
```

Figure 2-1. Sample Program Listing

```
00000024 61 42 41 42                    cpeq        gr66,gr65,0x42
00000028 AC 00 42 08                    jmpt        gr66,Cmd_B
0000002C 70 40 01 01                    aseq        0x40,gr1,gr1 ; NOP
00000030 A0 00 00 0B                    jmp         Cmd_I
                       ;**********************************************
                       ;* Command A is entered.  gr65 = the number of
                       ;* bytes in message A divided by 4.
                       ;* gr66 = location of the message.  Jump to the
                       ;* routine which writes the messages.
                       ;**********************************************
00000034 03 00 41 04   Cmd_A:           const       gr65,((Msg_B-Msg_A)/4)-1
00000038 03 00 42 00  R                 const       gr66,Msg_A
0000003C 02 00 42 00  R                 consth      gr66,Msg_A
00000040 A0 00 00 0A                    jmp         Write_Msg
00000044 70 40 01 01                    aseq        0x40,gr1,gr1 ; NOP
                       ;**********************************************
                       ;* Command B is entered.
                       ;**********************************************
00000048 03 00 41 04   Cmd_B:           const       gr65,((Msg_I-Msg_B)/4)-1
0000004C 03 00 42 14  R                 const       gr66,Msg_B
00000050 02 00 42 00  R                 consth      gr66,Msg_B
00000054 A0 00 00 05                    jmp         Write_Msg
00000058 70 40 01 01                    aseq        0x40,gr1,gr1 ; NOP
                       ;**********************************************
                       ;* An invalid command is entered.
                       ;**********************************************
0000005C 03 00 41 03   Cmd_I:           const       gr65,((End_Msgs-Msg_I)/4)-1
00000060 03 00 42 28  R                 const       gr66,Msg_I
00000064 02 00 42 00  R                 consth      gr66,Msg_I
                       ;**********************************************
                       ;* Message is written to the destination.
                       ;**********************************************
00000068 03 00 43 04  R Write_Msg:      const       gr67,Msg_Dest
0000006C 02 00 43 00  R                 consth      gr67,Msg_Dest
00000070 CE 00 87 41                    mtsr        cr,gr65
00000074 36 00 48 42                    loadm       0,0,gr72,gr66
00000078 CE 00 87 41                    mtsr        cr,gr65
0000007C 3E 00 48 43                    storem      0,0,gr72,gr67
                       ;**********************************************
                       ;* The rest of the destination area is filled
                       ;* with zeros.
                       ;**********************************************
00000080 03 00 40 00                    const       gr64,0
00000084 03 00 42 24  R                 const       gr66,Msg_Dest+0x20
00000088 02 00 42 00  R                 consth      gr66,Msg_Dest+0x20
0000008C 15 41 41 01                    add         gr65,gr65,1
00000090 81 41 41 02                    sll         gr65,gr65,2
00000094 14 43 43 41                    add         gr67,gr67,gr65
00000098 1E 00 40 43   Fill_Dest:       store       0,0,gr64,gr67
0000009C 15 43 43 04                    add         gr67,gr67,4
000000A0 60 41 42 43                    cpeq        gr65,gr66,gr67
000000A4 A4 FF 41 FD                    jmpf        gr65,Fill_Dest
000000A8 70 40 01 01                    aseq        0x40,gr1,gr1 ; NOP
                       ;**********************************************
                       ;* Go back and scan for next command.
                       ;**********************************************
```

Figure 2-1. Sample Program Listing (Cont'd)

```
92     000000AC A0 FF 00 D5                    jmp         Init
93     000000B0 70 40 01 01                    aseq        0x40,gr1,gr1 ; NOP
94
95                                             .bss
96                              ;**********************************************
97                              ;* Command input byte.
98                              ;**********************************************
99     00000000                 Cmd_Input:      .block      4
100                             ;**********************************************
101                             ;* Destination of the command messages.
102                             ;**********************************************
103    00000004                 Msg_Dest:       .block      0xfb
104                                             .end

Advanced Micro Devices, Inc. ASM29K Assembler Rel. 2.0-5   Sat Mar 03 19:20:26 1990
Page 4

                   Cross  Reference

Label              Value       References

Cmd_A      .text:00000034    35  -46
Cmd_B      .text:00000048    37  -54
Cmd_I      .text:0000005C    39  -62
Cmd_Input  .bss :00000000    1  20  21  -98
End_Msgs   .data:00000038    -12  62
Exe_Cmd    .text:0000001C    -34
Fill_Dest  .text:00000098    -84  87
Init       .text:00000000    1  -19  92
Msg_A      .data:00000000    -6  46  47  48
Msg_B      .data:00000014    -8  46  54  55  56
Msg_Dest   .bss :00000004    2  68  69  79  80  -102
Msg_I      .data:00000028    -10  54  62  63  64
Msgs       .data:00000000    1  -5
Scan       .text:00000010    -27  29
Write_Msg  .text:00000068    49  57  -68
```

Figure 2-1. Sample Program Listing (Cont'd)

The ".data" area defines the messages used by the program to respond to various command inputs. These messages are labeled **Msg_A**, **Msg_B**, and **Msg_I**.

Reading Input

The instructions that follow the **Read_Cmd** label clear any random data or previous commands from the **Cmd_Input** word. The **Scan** loop continually reads the **Cmd_Input** word to see if a command is entered (a value other than 0H).

Processing Commands

When a command is entered, the instructions after **Exe_Cmd** determine whether the command was “A”, “B”, or an invalid command.

If the command input word is “A” (ASCII 41H), execution jumps to the **Cmd_A** label where the length of the “Command A entered” message is loaded into register gr65, the location of **Msg_A** is loaded into register gr66, and execution is transferred to the instructions at **Write_Msg**. Similar operations are performed when the command input word is “B” or when there is an invalid command.

The instructions at **Write_Msg** load a word from the appropriate message location, store the word into the destination area, increment the source and destination addresses, decrement the message length counter, and perform these instructions again until all words are transferred.

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

The Destination Area

The “.bss” area declares storage for the command input byte and the destination area.

Assembling the Sample Program

You can use several PC hosted software development tools to generate absolute files. However, your compiler/assembler/linker must be able to generate files in one of the following formats:

- IEEE-695 MUFOM (Microprocessor Universal Format for Object Modules).
- HP absolute (either with associated symbol files or just raw absolute file).
- Intel hexadecimal.
- Motorola S-records.
- Tektronix hexadecimal.

The sample program was written for and assembled with AMD's PC hosted assembler for the 29000/29050. The following command was used to assemble the sample program.

```
C> as29 -l -fgosx cmd_rdr.src > cmd_rdr.lis
<RETURN>
```

In addition to the assembler listing (cmd_rdr.lis), the "cmd_rdr.obj" relocatable file is created. Relocatable files are linked together to form the absolute file, which is loaded into the emulator.

Linking the Sample Program

The linker command file (cmd_rdr.lnk) shown in figure 2-2 and the following linker command were used to generate the absolute file.

```
C> ld29 -c cmd_rdr.lnk -o cmd_rdr.abs -m -e
Init > cmd_rdr.map <RETURN>
```

In addition to the linker load map listing (cmd_rdr.map), the "cmd_rdr.abs" absolute file is created. This file contains the COFF format absolute code.

```
ORDER .data=0x10000,.text=0x2000,.bss=0x20000
LOAD cmd_rdr.obj
```

Figure 2-2. The "cmd_rdr.lnk" Linker Command File

Converting Absolute File Format

Because the COFF absolute file format created by the “ld29” linker is not accepted by the HP 64774 emulator, the absolute file must be converted to a format that is acceptable. The AMD software development tools contain a utility that converts COFF format files. The following command uses this utility to create a Motorola S-record format absolute file.

```
C> coff2hex -m -o cmd_rdr.hex cmd_rdr.abs
<RETURN>
```

Starting the 29000/29050 PC Interface

If you have set up the emulator device table and the **HPTABLES** shell environment variable as shown in the *HP 64700 Emulators PC Interface Reference*, you can start up the 29000/29050 PC Interface by entering the following command from the MS-DOS prompt:

```
C> pcam29k <emulname>
```

In the command above, **pcam29k** is the command to start the PC Interface; “< emulname> ” is the logical emulator name given in the emulator device table.

The processor type in the emulator device table should be “am29000”.

You should see the display shown in figure 2-3. If this command is not successful, you will see an error message and will return to the MS-DOS prompt.

Selecting PC Interface Commands

You can select command options by either using the left and right arrow keys to highlight the option and press the **< Enter>** key, or you can type the first letter of that option. If you select the wrong option, you can press the **< ESC>** key to retrace up the command tree.

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

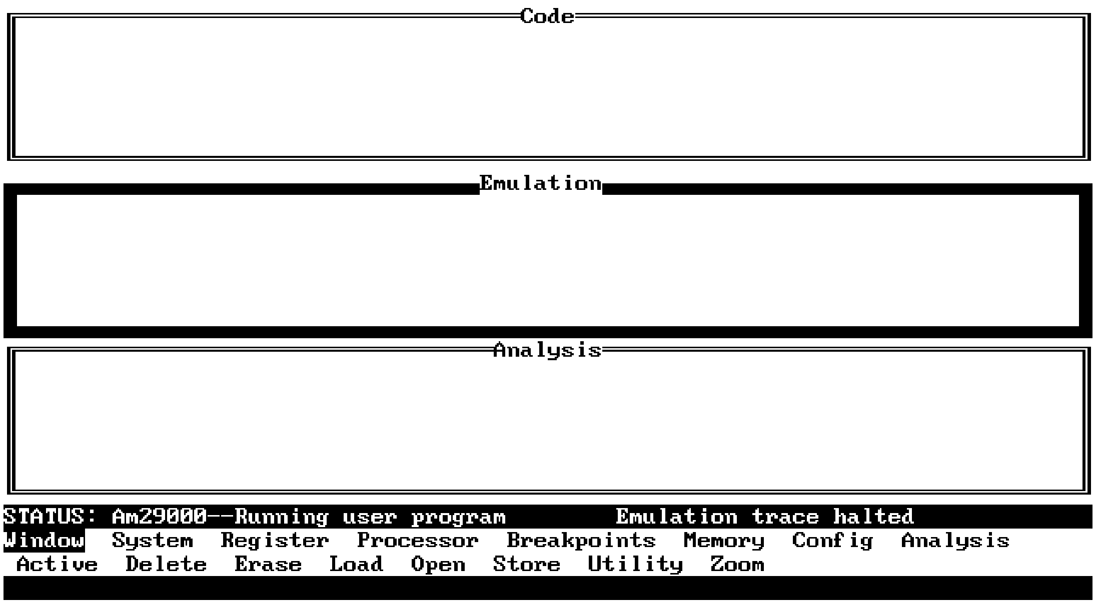

Figure 2-3. PC Interface Display

Emulator Status

The status of the emulator is shown on the line above the command options. The PC Interface periodically checks the status of the emulator and updates the status line. Error messages are saved in the “Error Log” window. Status messages include the following:

Status Messages	Description of Message
Emulator in reset state	The emulation processor is reset.
Running user program	The emulator is running a program.
No target system clock	The emulator is expecting, but not receiving, a clock from the target system.
Monitor is accepting commands	The monitor is allowing commands to be executed.
No bus cycles	Activity on the bus cannot be processed.
Waiting for CMB to become ready	The emulator is waiting for a READY signal on the CMB.
Processor halted	The emulation processor was stopped.
Waiting for target reset	The emulator is waiting for a reset from the target system.
Target system reset active	A reset has been executed in the target system.
Unknown state	The state of the emulation processor is unknown.

Mapping Memory

The memory mapper tells the emulator how to access memory locations in a particular range.

- The emulator needs to know whether memory is located in the emulator or in the target system.
- The emulator also needs to know whether the memory is RAM or ROM, which locations of physical emulation memory are used for a particular address range, which ranges are overlaid, and whether word, half-word, or byte accesses should be used for particular ranges in target memory.

Which Memory Locations Should Be Mapped?

Typically, assemblers generate relocatable files and linkers combine relocatable files to form the absolute file. The linker load map listing will show what memory locations your program will occupy in memory. Figure 2-4 shows an Advanced Micro Devices linker load map listing for the sample program.

```
Advanced Micro Devices, Inc. Am29000 Loader - Version 2.0-9
Copyright (c) 1987 - 1989 Microtec Research Inc.
ALL RIGHTS RESERVED. Serial Number AS2002563
Advanced Micro Devices, Inc. ASM29K Linker Rel. 2.0-9       Sat Mar 03 19:20:29 1990
Page   1

Command line: \USR\C29K\BIN\LD29.EXE  -c cmd_rdr.lnk -o cmd_rdr.abs -m -e Init

ORDER .data=0x10000,.text=0x2000,.bss=0x20000
LOAD cmd_rdr.obj


OUTPUT MODULE NAME:    cmd_rdr.abs
OUTPUT MODULE FORMAT:  ABSOLUTE
---------------------

SECTION SUMMARY
---------------

SECTION     TYPE     START     END         SIZE        ALIGN      MODULE

.text       TEXT     00002000  000020B3    000000B4    4 BYTES    cmd_rdr.obj
.data       DATA     00010000  00010037    00000038    8 BYTES    cmd_rdr.obj
.bss        BSS      00020000  000200FF    00000100    8 BYTES    cmd_rdr.obj
```

Figure 2-4. Linker Load Map for the Sample Program

```
LOCAL SYMBOL TABLE       * (DEBUG) - Special symbolic debugging symbol
------------------       * (ABSOLUTE) - Absolute symbol

SYMBOL          SECTION         VALUE           MODULE


.file           (DEBUG)         00000013        cmd_rdr.obj
.text           .text           00002000        cmd_rdr.obj
.data           .data           00010000        cmd_rdr.obj
.bss            .bss            00020000        cmd_rdr.obj
Write_Msg       .text           00002068        cmd_rdr.obj
End_Msgs        .data           00010038        cmd_rdr.obj
Scan            .text           00002010        cmd_rdr.obj
Exe_Cmd         .text           0000201C        cmd_rdr.obj
Cmd_A           .text           00002034        cmd_rdr.obj
Cmd_B           .text           00002048        cmd_rdr.obj
Msg_I           .data           00010028        cmd_rdr.obj
Fill_Dest       .text           00002098        cmd_rdr.obj
Cmd_I           .text           0000205C        cmd_rdr.obj
Msg_A           .data           00010000        cmd_rdr.obj
Msg_B           .data           00010014        cmd_rdr.obj


GLOBAL SYMBOL TABLE
-------------------

SYMBOL          SECTION         VALUE

Init            .text           00002000
Msgs            .data           00010000
Cmd_Input       .bss            00020000
Msg_Dest        .bss            00020004
edata           (ABSOLUTE)      00010038
end             (ABSOLUTE)      00020100
etext           (ABSOLUTE)      000020B0


CROSS REFERENCE TABLE    * - - Defined Module
---------------------

SYMBOL          SECTION         REFERENCED

Init            .text           - cmd_rdr.obj
Msgs            .data           - cmd_rdr.obj
Cmd_Input       .bss            - cmd_rdr.obj
Msg_Dest        .bss            - cmd_rdr.obj
edata           (ABSOLUTE)      LINKER-DEFINED
end             (ABSOLUTE)      LINKER-DEFINED
etext           (ABSOLUTE)      LINKER-DEFINED


START ADDRESS:   00002000
```

Figure 2-4. Linker Load Map for Sample Program (Cont'd)

From the load map listing, you can see that the sample program occupies code segment locations in three address ranges. The ".text" area, which contains the opcodes and operands of the sample program, occupies locations 2000H through 20B3H. The ".data" area, which contains the ASCII values of the messages the program displays, occupies locations 10000H through 10037H. The ".bss" segment in the sample program, which contains the command input byte and the destination area, occupies locations 20000H through 200FFH.

The minimum size of a block of emulation memory is 64 Kbytes. When mapping memory, you can specify a range smaller than 64 Kbytes, but the mapper will automatically map the entire 64 Kbyte block in which that range resides.

Two mapper terms are specified for the example program. Since the program writes to the destination locations, the mapper block containing the destination locations should not be characterized as ROM memory.

To map memory for the sample program, select:

```
Config Map Modify
```

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

```
0..0ffff@r
```

The "@r" appended to the address range above is an *address space designator*; it specifies that the range be mapped to instruction ROM address space. Address space designators are described in more detail in the chapter "Using the Emulator — In Depth".

Move the cursor to the "memory type" field of term 1, and press the **< Tab>** key to select the "erom" (emulation ROM) type. Move the cursor to the "attribute" field of term 1, and press the **< Tab>** key to select "bnka1". Move the cursor to the "address range" field of term 2, and enter:

```
10000..2ffff@d
```

The "@d" address space designator maps the range to data memory address space.

Move the cursor to the “memory type” field of term 2, and press the **< Tab>** key to select the “eram” (emulation RAM) type. Move the cursor to the “attribute” field of term 2, and press the **< Tab>** key to select “bnka2”.

To save the memory map, use the **< Enter>** key to exit the field in the lower right corner. (The **< End>** key on Vectra keyboards moves the cursor directly to the last field.) Figure 2-5 shows the memory configuration display.

```
                              Memory Map Configuration
                              Unmapped Memory Type  tram
Term                         Address Range                           Type  Attribute
  1 0..0ffff@r                                                       erom   bnka1
  2 10000..2ffff@d                                                   eram   bnka2
  3 Empty                                                            grd
  4 Empty                                                            grd
  5 Empty                                                            grd
  6 Empty                                                            grd
  7 Empty                                                            grd
  8 Empty                                                            grd
  9 Empty                                                            grd
 10 Empty                                                            grd
 11 Empty                                                            grd
 12 Empty                                                            grd
 13 Empty                                                            grd
 14 Empty                                                            grd
 15 Empty                                                            grd

  ←↑↓→ :Interfield movement   CTRL ←→ :Field editing     TAB :Scroll choices
STATUS: Am29000--Running user program        Emulation trace halted

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

Figure 2-5. Memory Map Configuration

When mapping memory for your target system programs, you may want to map emulation memory locations containing programs and constants (locations which should not be written to) as ROM. This will prevent programs and constants from being written over accidentally, and will cause breaks when instructions attempt to do so.

For more information on the memory mapper and emulation memory, refer to the “Mapping Memory” section in the chapter “Using the Emulator — In Depth”.

Loading Programs into Memory

Because two address spaces were mapped (instruction ROM with @r, and data memory with @d), you must load the absolute file twice. First, load the instruction ROM address space by selecting:

```
Memory Load
```

Enter the format of your absolute file. The emulator accepts absolute files in the following formats:

- IEEE-695 MUFOM (Microprocessor Universal Format for Object Modules).
- HP absolute (either with associated symbol files or just raw absolute file).
- Intel hexadecimal.
- Motorola S-records.
- Tektronix hexadecimal.

The “cmd_rdr.hex” absolute file is in Motorola S-record format, so use the < **Tab**> key to select “Motorola_Hex”.

The next field allows you to selectively load the portions of the absolute file which reside the following:

- emulation memory
- target system memory
- both emulation and target system memory
- background monitor

Because emulation memory is mapped for sample program locations, you can enter either “emulation” or “both”.

Next, you select the address space designator for the load operation. You will load the sample program into instruction ROM address space with the “r” designator.

Finally, enter the name of your absolute file (“cmd_rdr.hex” in this example) in the last field, and press < **Enter**> to start the memory download.

Now, to load the absolute file into data memory, select:

```
Memory Load
```

Use all the same options except for the address space designator. Move the cursor to the “address space designator” field and use the < **Tab**> key to select “d” for data memory designator.

Press the < **End**> key to move the cursor to the last field, and press < **Enter**> to start the memory download.

Using Symbols

IEEE-695 or HP 64000 Format Symbols

Symbols are part of the IEEE-695 and HP 64000 file format definitions. That is, symbols can be contained in these file formats. When you load IEEE-695 or HP 64000 format files, the PC Interface uses a reader program to create files with the extensions “.HPA” and “.HPS” (whose base names are the same as the absolute file). The “.HPA” file is in a binary format that is compatible with the HP 64700-Series firmware. The “.HPS” file is an ASCII source file that contains the symbol to address mappings used by the PC Interface. See the appendices “Using the HP 64000 Reader” and “Using the IEEE-695 Reader” for more information.

Other File Formats

When your absolute file is not an IEEE-695 or HP 64000 format file, the PC Interface does not create “.HPA” or “.HPS” files. However, you can use an editor create a “.HPS” file using symbol information from the linker load map output listing. Figure 2-6 shows the “.HPS” file for the sample program.

Note

The format of a “.HPS” file requires

- module names to be preceded by a single space,
- symbols and addresses to be separated by a single < **Tab**> character, and
- the lines in the file that you will sort (you can use the MS-DOS **sort** command to do this).

```
 cmd_rdr
Cmd_Input       00020000@d
Init    00002000@r
Msg_Dest        00020004@d
Msgs    00010000@d
|cmd_rdr|Cmd_A  00002034@r
|cmd_rdr|Cmd_B  00002048@r
|cmd_rdr|Cmd_I  0000205C@r
|cmd_rdr|End_Msgs       00010038@d
|cmd_rdr|Exe_Cmd        0000201C@r
|cmd_rdr|Fill_Dest      00002098@r
|cmd_rdr|Msg_A  00010000@d
|cmd_rdr|Msg_B  00010014@d
|cmd_rdr|Msg_I  00010028@d
|cmd_rdr|Scan   00002010@r
|cmd_rdr|Write_Msg      00002068@r
```

Figure 2-6. The "cmd_rdr.hps" Symbol File

Loading Global Symbols

When you load memory with IEEE-695 format absolute files, global symbols are automatically loaded.

When you load memory with non-IEEE-695 format absolute files and create the ".HPS" file, you must load the symbols in the ".HPS" file by selecting the following command:

System **S**ymbols **G**lobal **L**oad

Enter the name of the ".HPS" file, and press **< Enter>** to load the symbols.

Displaying Global Symbols

After global symbols are loaded, both global and local symbols can be used when entering expressions. Global symbols are entered as they appear in the source file or in the global symbols display.

To display global symbols, select:

System **S**ymbols **G**lobal **D**isplay

The symbols window automatically becomes active. You can press **< CTRL> Z** to zoom the window. The resulting display follows.

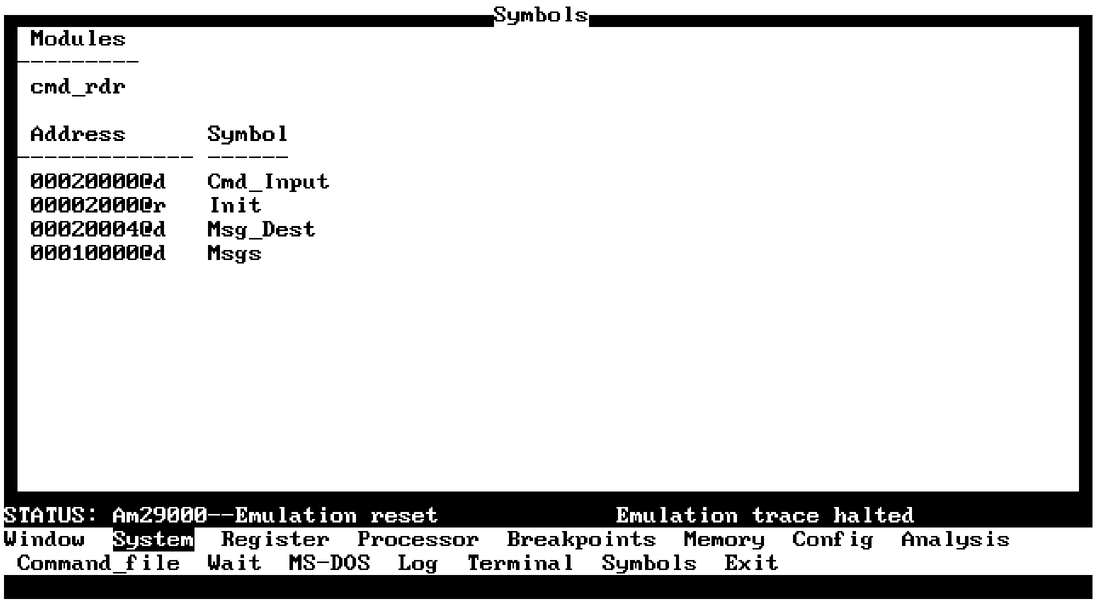

Figure 2-7. Global Symbols Display

Displaying Local Symbols

To load and display local symbols, select:

System **S**ymbols **L**ocal,**D**isplay

Enter the name of the module in which the local symbols appear (for example, “cmd_rdr”). Press **< Enter>** . The resulting display follows.

After you load and display local symbols with the **S**ystem **S**ymbols **L**ocal command, you can use local symbols as they appear in the source file or local symbol display.

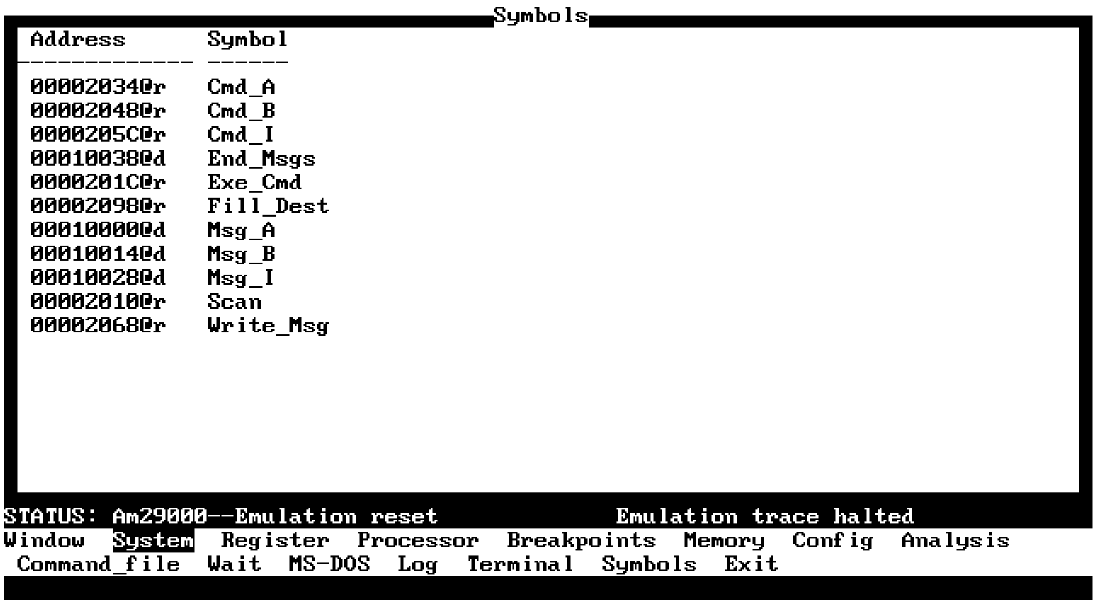

Figure 2-8. Local Symbols Display

Transferring Symbols to the Emulator

Before you can view symbols in mnemonic memory and trace displays, you must transfer them to the emulator.

After global symbols are loaded, you can transfer them to the emulator by selecting:

`**S**ystem **S**ymbols **G**lobal **T**ransfer`

To transfer local symbols for the module “cmd_rdr” to the emulator, select:

`**S**ystem **S**ymbols **L**ocal **T**ransfer **G**roup “cmd_rdr”`

Removing Symbols from the Emulator

If you transferred many symbols to the emulator, you can fill the memory used to store them. You can remove global and local symbols from the emulator by using the **R**emove option in place of the **T**ransfer option in the symbols commands.

Displaying Memory in Mnemonic Format

Once you have loaded a program into the emulator, you can verify that the program was loaded by displaying memory in mnemonic format. To do this, select:

`Memory Display Mnemonic`

Enter the address range **Init..**. The two periods indicate you are specifying an address range. The range size is 128 bytes when no second address in the range is specified. The emulation window automatically becomes active. You can press **< CTRL> Z** to zoom the window and **< PgUp>** to see the beginning of the range.

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

```
                                    Emulation
  Address     Symbol            Mnemonic
------------ ---------------- ------------------------------------------------
00000200@r   Init              const    gr65,0x0000
000002004@r  -                 const    gr64,0x0000
000002008@r  -                 consth   gr64,0x00020000
00000200c@r  -                 store    0,0x00,gr65,gr64
000002010@r  cmd_rdr:Scan      load     0,0x00,gr65,gr64
000002014@r  -                 cpeq     gr66,gr65,0x00
000002018@r  -                 jmpt     gr66,cmd_rdr:Scan
00000201c@r  cmd_rdr:Exe_Cmd   cpeq     gr66,gr65,0x41
000002020@r  -                 jmpt     gr66,cmd_rdr:Cmd_A
000002024@r  -                 cpeq     gr66,gr65,0x42
000002028@r  -                 jmpt     gr66,cmd_rdr:Cmd_B
00000202c@r  -                 nop
000002030@r  -                 jmp      cmd_rdr:Cmd_I
000002034@r  cmd_rdr:Cmd_A     const    gr65,0x0004
000002038@r  -                 const    gr66,0x0000
00000203c@r  -                 consth   gr66,0x00010000
000002040@r  -                 jmp      cmd_rdr:Write_Msg

STATUS: Am29000--Emulation reset           Emulation trace halted
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Display  Modify  Load  Store  Copy  Find  Report
```

Figure 2-9. Memory Mnemonic Display

Stepping Through the Program

The emulator allows you to execute one instruction or a number of instructions with the step command. To begin stepping through the sample program, select:

Processor **S**tep **A**ddress

Enter a step count of 1, enter the symbol "Init" (defined as a global symbol in the source file), and press < **Enter**> to step from the program's first address, 2000H in instruction ROM. The executed instruction, the program counter address, and the resulting register contents are displayed. Figure 2-10 shows the resulting display.

```
                                   Emulation
00000205c@r  cmd_rdr:Cmd_I     const    gr65,0x0003
000002060@r  -                 const    gr66,0x0028
000002064@r  -                 consth   gr66,0x00010000
000002068@r  d_rdr:Write_Msg   const    gr67,0x0004
00000206c@r  -                 consth   gr67,0x00020000
000002070@r  -                 mtsr     cr,gr65
000002074@r  -                 loadm    0,0x00,gr72,gr66
000002078@r  -                 mtsr     cr,gr65
00000207c@r  -                 storem   0,0x00,gr72,gr67

000002000@r   Init             const    gr65,0x0000
PC = 000002004@r
      CA IP TE TP TU FZ LK RE WM PD PI SM IM DI DA
  cps  0  0  0  0  0  1  0  1  0  1  1  1 00  1  1
      PRL DW VF RV BO CP CD
  cfg  03  0  1  0  0  0  1
  pc0 fffffffc  pc1 fffffffc

STATUS: Am29000--Running in monitor          Emulation trace halted
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Go  Break  Reset  CMB  Step
```

Figure 2-10. Register Contents

Note

You cannot display registers if the processor is reset. Use the **P**rocessor **B**reak command to cause the emulator to start executing in the monitor.

You can display registers while the emulator is executing a user program (if execution is not restricted to real-time). Emulator execution will temporarily break to the monitor.

To continue stepping through the program, you can select:

```
Processor Step Pc
```

After selecting the command above, you can change the previous step count. If you want to step the same number of times, press **< Enter>** to start the step.

To repeat the previous command, press **< CTRL> R**.

You can also step 1 time from the current program counter by pressing the **< F1>** key. The sequence of keystrokes for a single step command is assigned to the **< F1>** key by default. You can modify this keystroke macro or define other keystroke macros with the **C**onfig **K**ey_macro command. For more information on function key macros, see the *PC Interface Reference.*

Specifying a Step Count

If you want to continue to step from the current program counter, select:

```
Processor Step Pc
```

The previous step count is displayed in the “number of instructions” field. You can enter a number from 1 through 99 to specify the number of times to step. Type 5 into the field, and press **< Enter>** . Figure 2-11 shows the resulting display.

```
                                    Emulation
  cps  0  0  0  0  0  1  0  1  0  1  1  1 00  1  1
      PRL DW VF RV BO CP CD
  cfg  03  0  1  0  0  0  1
  pc0 fffffffc  pc1 fffffffc


0000020100r  cmd_rdr:Scan     load     0,0x00,gr65,gr64
0000020140r  -                cpeq     gr66,gr65,0x00
0000020180r  -                jmpt     gr66,cmd_rdr:Scan
00000201c0r  cmd_rdr:Exe_Cmd  cpeq     gr66,gr65,0x41
0000020100r  cmd_rdr:Scan     load     0,0x00,gr65,gr64
PC = 0000020140r
      CA IP TE TP TU FZ LK RE WM PD PI SM IM DI DA
  cps  0  0  0  0  0  1  0  1  0  1  1  1 00  1  1
      PRL DW VF RV BO CP CD
  cfg  03  0  1  0  0  0  1
  pc0 fffffffc  pc1 fffffffc

STATUS: Am29000--Running in monitor          Emulation trace halted
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Go  Break  Reset  CMB  Step
```

Figure 2-11. Stepping the Processor

When you specify step counts greater than one, all the instructions that are executed are displayed, but the register contents are only displayed after the last instruction.

Modifying Memory

The preceding step commands show the sample program executing in the **Scan** loop, where it continually reads the command input word to see whether if a command was entered. To simulate entry of a sample program command, you can modify the command input word by selecting:

```
Memory Modify Word
```

Now enter the address of the memory location to be modified, an equal sign, and the new value of that location, for example, “Cmd_Input= ’A’”. (The **Cmd_Input** label was defined as a global symbol in the source file.)

To verify that 41H was written to **Cmd_Input** (20000H), select:

```
Memory Display Word
```

Type the symbol “Cmd_Input”, and press **< Enter>** .

You can continue to step through the program as shown earlier in this chapter to view the instructions that are executed when an “A” (41H) command is entered.

Running the Program

To start the emulator executing the sample program, select:

```
Processor Go Pc
```

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

Searching Memory for Data

You can search the message destination locations to verify that the sample program writes the appropriate messages. The command "A" (41H) was entered above, so the "Command A entered " message should have been written to the **Msg_Dest** locations. To search the destination memory location for this sequence of characters select:

```
Memory Find
```

Enter the range of the memory locations to be searched, "Msg_Dest..", and enter the data "'Command A entered '". The resulting information shows that the message was written:

```
pattern match at address: 000020004@d
```

To verify that the sample program works for the other allowed commands, you can modify the command input byte to "B" and search for "'Entered B command '", or you can modify the command input byte to "C" and search for "'Invalid Command '".

Breaking into the Monitor

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

```
Processor Break
```

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

While the break will occur as soon as possible, the actual stopping point may be many cycles after the break request (due to the speed of the processor).

A break is achieved by transferring the processor from the normal operating mode to the halt mode. See the topic "Effects of the Background Monitor" in the "Using the Emulator — the Basics" chapter. Also see the Am29000/Am29050 microprocessor data book for more information about how a break is implemented.

Using Breakpoints

You can define breakpoints at opcode locations in the user program. When the breakpoint is hit, execution of the emulator is diverted from the user program to the monitor.

Note

Breakpoints can only be set at opcode locations; that is, the address of the breakpoint must be on a 4 byte boundary.

Commands that define, set, or clear breakpoints cause emulator execution to break into the monitor.

Defining a Breakpoint

Defining a breakpoint enables the “breakpoints” feature. To define a breakpoint at the address of the **Cmd_I** label of the sample program (205c@r), select:

```
Breakpoints Add
```

Enter the local symbol “cmd_rdr:Cmd_I”. After the breakpoint is added, it appears in the emulation window and is shown as set.

Run the program by selecting:

```
Processor Go Pc
```

The status line shows that the emulator is running the user program. Modify the command input word to an invalid command:

```
Memory Modify Word
```

Enter an invalid command, such as “Cmd_Input= 75”. The following messages result:

```
ALERT:  Software Breakpoint: 205c@r
STATUS: Am29000--Running in monitor
```

Displaying Breakpoints

To view the status of the breakpoint, select:

```
Breakpoints Display
```

The information displayed shows that the breakpoint has been cleared.

Setting a Breakpoint

When a breakpoint is hit, it becomes disabled. To reenable the software breakpoint, you can select:

```
Breakpoints Set Single
```

As with the **B**reakpoints **A**dd command, the breakpoint is shown as set in the emulation window.

Clearing a Breakpoint

If you want to clear a software breakpoint (that does not get hit during program execution, for example) you can select:

```
Breakpoints Clear Single
```

You are given a field in which to specify the breakpoint to be cleared. Type in the symbol or address of the breakpoint, and press **< Enter>** to clear the breakpoint.

Using the Analyzer

The emulation analyzer has 80 trace signals which monitor internal emulation lines (instruction bus, data bus, and status lines). Because 80 trace signals are not enough to monitor all the signals associated with the 29000/29050, the analyzer has three modes:

- Instruction/Data bus mode. All instruction accesses are recorded, and data accesses are recorded if they don't occur on the same clock cycle as instruction accesses. States in which instruction and data accesses occur on the same cycle are marked in the trace.

- Data/Instruction bus mode. All data accesses are recorded, and instruction accesses are recorded if they don't occur on the same clock cycle data accesses. States in which instruction and data accesses occur on the same cycle are marked in the trace.

- Status mode. Status information for all transactions is stored.

The "Getting Started" chapter describes how to modify the analysis mode in the general emulator configuration. The "Configuring the Emulator" chapter describes the analyzer modes.

The analyzer collects data at each pulse of a clock signal, and saves the data (a trace state) if it meets a "storage qualification" condition.

Note

External analysis (capability to probe signals external to the emulator) is not available with the HP 64774 emulator.

Predefined Trace Labels

Four trace labels are predefined in the 29000/29050 analyzer: addr, data, stat, and extra. Table 2-1 shows the trace label names and emulation processor lines associated with the trace signals.

Table 2-1. Trace Signal Assignments

Trace Labels and associated Trace Signals	Assignment (depends on analyzer mode)		
	Instruction Bus Mode	**Data Bus Mode**	**Status Mode**
addr 0..31	A_0-A_{31}	A_0-A_{31}	A_0-A_{31}
data 32..63	I_0-I_{31}	D_0-D_{31}	BGRT
33			BREQ
34			BINV
35			CDA
36			DBACK
37			DBREQ
38			DERR
39			DRDY
40			DREQ
41			DREQT0
42			DREQT1
43			IBACK
44			IBREQ
45			IERR
46			INTR0
47			INTR1
48			INTR2
49			INTR3
50			IRDY
51			IREQ
52			IREQT
53			LOCK
54			MPGM0
55			MPGM1
56..63			I_{24}-I_{31}

Table 2-1. Trace Signal Assignments (Cont'd)

Trace Labels and associated Trace Signals	Assignment (depends on analyzer mode)		
	Instruction Bus Mode	**Data Bus Mode**	**Status Mode**
stat 64..78	SUP/US	SUP/US	STAT0
65	MPGM0	MPGM0	STAT1
66	MPGM1	MPGM1	STAT2
67	STAT0	STAT0	SUP/US
68	STAT1	STAT1	TRAP0
69	STAT2	STAT2	TRAP1
70	LOCK	DREQT0	WARN
71	IREQT	DREQT1	OPT0
72	IERR	R/W	OPT1
73		OPT0	OPT2
74		OPT1	PDA
75		OPT2	PEN
76		DERR	PIA
77	(collision)	(collision)	
78	= 0	= 1	R/W
extra 79	= 1	= 1	= 0

A collision indicates that either the instruction or data was taken; the other is lost.

Predefined Status Equates

Common values for the status trace signals have been predefined. These equates may be used to specify values for the "stat" trace labels when qualifying trace conditions.

Predefined status equates can't be used wehn specifying values under the "data" trace label while in "status" analyzer mode. Instead, you can specify the actual binary value associated with a particular status.

Table 2-2 shows how the equates are used in the different analyzer modes.

Table 2-2. Predefined Equates for Analyzer Labels

Instruction or Data Bus Mode		
Trace Label	**Equate**	**Description**
stat	byte	Byte access.
	cpx	Coprocessor transfer.
	dbus	D-bus access.
	derr	Data bus error.
	dind	Data bus instruction/data access.
	drom	Data bus instruction ROM access.
	exec	Normal execution.
	hfwd	Half-word access.
	ibus	I-bus access.
	ierr	Instruction bus error.
	iind	Instruction bus instruction/data access.
	inout	Data bus I/O read access.
	inttrp	Interrupt or trap.
	iret	Interrupt return.
	irom	Instruction bus instruction ROM access.
	nonseq	Non-sequential instruction fetch.
	rd	Data bus read.
	run	Processor is running.
	sup	Supervisor mode.
	usr	User mode.
	word	Word access.
	wr	Data bus write.

Table 2-2. Predefined Equates for Analyzer Labels (Cont'd)

Status Mode		
Trace Label	**Equate (or binary value)**	**Description**
data	`0xxxxxxxx xxxxxxxx xxxxxxxx xxxx0xxxY`	Coprocessor data accept.
	`0xxxxxxxx xxxxxxxx xxxx1xxx xxxxxxxxY`	Coprocessor transfer.
	`0xxxxxxxx xxxxxxxx xxxxxxxx x0xxxxxxY`	Data bus error.
	`0xxxxxxxx xxxxxxxx xx0xxxxx xxxxxxxxY`	Instruction bus error.
	`0xxxxxxxx xxxxxxxx xxxxx00x xxxxxxxxY`	Data bus instruction/data access.
	`0xxxxxxxx xxx0xxxx xxxxxxxx xxxxxxxxY`	Instruction bus instruction/data access.
	`0xxxxxxxx xxxxxxxx xxxxx01x xxxxxxxxY`	Data bus I/O access.
	`0xxxxxxxx xxx1xxxx xxxxxxxx xxxxxxxxY`	Instruction bus instruction ROM access.

Resetting the Analysis Specification

To be sure that the analyzer is in its default or power-up state, select:

Analysis **T**race **R**eset

Specifying a Simple Trigger

Suppose you want to trace the states of the sample program that follow the read of a “B” (42H) command from the command input word. You must modify the trace specification by selecting:

Analysis **T**race **M**odify

Move the cursor to the “Trigger on” field, use the < **Tab**> key to select pattern “a”, and press < **Enter**> .

You are now given a screen in which you assign values to patterns. Move the cursor to the “addr” column associated with pattern “a”. You will notice an expanded field in the lower portion of the screen in which you may specify the address value to be associated with pattern “a”. Type in “Cmd_Input”, and press < **Enter**> .

The cursor is now in the data field associated with pattern “a”. Type in “42” in the entry field at the bottom of the display, and press < **Enter**> . The cursor is now in the “stat” column associated with pattern “a”.

Enter “rd” and press < **Enter**> . Figure 2-12 shows the final patterns and expressions screen.

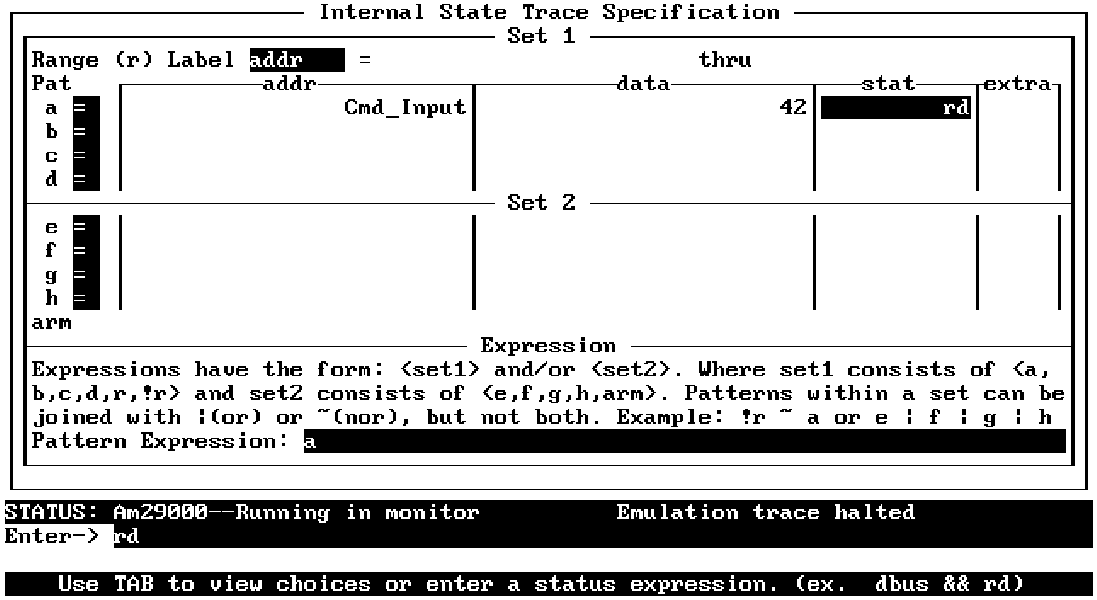

Figure 2-12. Analyzer Patterns and Expressions

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

2 Store         any state





   Branches off          Count  off      Prestore off    Trigger position
                                                         start  of 1024
STATUS: Am29000--Running in monitor           Emulation trace halted

 Use the TAB and Shift-TAB keys to select a trigger position or enter a number.
```

Figure 2-13. Trace Specification

To save the pattern assignment, press < **End**> to move the cursor to the field in the lower right corner, and press < **Enter**> .

To save the trace specification (see figure 2-13), press < **End**> to move the cursor to the field in the lower right corner, and press < **Enter**> .

Starting the Trace

First, make sure that the program is running by selecting:

`Processor Go Address`

Enter the starting address, “Init”, and press < **Enter**> .

To start the trace, select:

`Analysis Begin`

A message on the status line shows that the trace is running.

You do not expect the trigger to be found because no commands have been entered. Modify the command input word to “B” by selecting:

Memory **M**odify **W**ord

Enter “Cmd_Input= ’B’”. The status line now shows that the emulation trace is complete.

Displaying the Trace

To display the trace, select:

Analysis **D**isplay

There are two fields in which to specify the states to display. Use the right arrow key to move the cursor to the “Ending state to display” field. Type the number of the starting state plus 15 into the ending state field, press **< Enter>** , and use **< CTRL> Z** to zoom the trace window. A display similar to figure 2-14 will be shown.

```
                                    Analysis
  Line    addr,H     Am29000 mnemonic
 -----    --------   --------------------------------------------------
     0    md_Input   data rd wd:00000042             [su i/d mp:0]
     1    00002020   jmpt     gr66,cmd_rdr:Cmd_A     [su rom mp:0]
     2    00002024   cpeq     gr66,gr65,0x42         [su rom mp:0]
     3    00002028   jmpt     gr66,cmd_rdr:Cmd_B     [su rom mp:0]
     4    0000202c   nop                             [su rom mp:0]
     5    00002030   jmp      cmd_rdr:Cmd_I          [su rom mp:0]
     6    dr:Cmd_B   const    gr65,0x0004            [su rom mp:0]
     7    0000204c   const    gr66,0x0014            [su rom mp:0]
     8    00002050   consth   gr66,0x00010000        [su rom mp:0]
     9    00002054   jmp      cmd_rdr:Write_Msg      [su rom mp:0]
    10    00002058   nop                             [su rom mp:0]
    11    dr:Cmd_I   const    gr65,0x0003            [su rom mp:0]
    12    rite_Msg   const    gr67,0x0004            [su rom mp:0]
    13    0000206c   consth   gr67,0x00020000        [su rom mp:0]
    14    00002070   mtsr     cr,gr65                [su rom mp:0]
    15    00002074   loadm    0,0x00,gr72,gr66       [su rom mp:0]

STATUS: Am29000--Running user program        Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Begin  Halt  CMB  Format  Trace  Display
```

Figure 2-14. Displayed States

Table 2-3. Trace Mnemonics

Mnemonic	Processor Lines	Description	An. Modes
ad	OPT2..OPT1 = 110	ADAPT29K accesses.	D, S
bq	BREQ = 0	Bus Request active.	S
bus grant	BGRT = 0	Bus Grant active.	S
bus invalid	BINV = 0	Bus Invalid active.	S
by	OPT2..OPT1 = 001	Byte access.	D, S
cc	OPT2..OPT1 = 101	Cache control.	D, S
cda	CDA = 0	Coprocessor Data Accept active.	S
cpx	DREQT1..DREQT0 = 1x	Coprocessor transfer.	D, S
dba	DBACK = 0	Data Burst Acknowledge active.	S
dbq	DBREQ = 0	Data Burst Request active.	S
de	DERR = 0	Data Error active.	D, S
dq	DREQ = 0	Data Request active.	S
dr	DRDY = 0	Data Ready active.	S
ex	STAT2..STAT0 = 111	Executing Mode.	S
hlt	STAT2..STAT0 = 000	Halt or Step Modes.	S
hw	OPT2..OPT1 = 010	Half-word access.	D, S
i/d	IREQT = 0	Instruction/data memory access.	I, S
i/d	DREQT1..DREQT0 = 00	Instruction/data memory access.	D, S
i/o	DREQT1..DREQT0 = 01	Input/output access.	D, S
i/t	STAT2..STAT0 = 101	Taking Interrupt or Trap.	S
i0	INTR0 = 0	Interrupt Request 0 active.	S
i1	INTR1 = 0	Interrupt Request 1 active.	S
i2	INTR2 = 0	Interrupt Request 2 active.	S
i3	INTR3 = 0	Interrupt Request 3 active.	S
iba	IBACK = 0	Instruction Burst Acknowledge active.	S
ibq	IBREQ = 0	Instruction Burst Request active.	S
ie	IERR = 0	Instruction Error active.	I, S
iq	IREQ = 0	Instruction Request active.	S
ir	IRDY = 0	Instruction Ready active.	S
irt	STAT2..STAT0 = 100	Interrupt Return.	S
lck	LOCK = 0	Lock active.	I, S
lti	STAT2..STAT0 = 010	Load Test Instruction Mode.	S
mp:< val.>	MPGM1..MPGM0	MMU Programmable.	I, D, S
ns	STAT2..STAT0 = 110	Non-sequential Instruction Fetch.	S
pd	PDA = 0	Pipelined Data Access active.	S

Table 2-3. Trace Mnemonics (Cont'd)

Mnemonic	Processor Lines	Description	An. Modes
pe	PEN = 0	Pipeline Enable active.	S
ph	STAT2..STAT0 = 001	Pipeline Hold Mode.	S
pi	PIA = 0	Pipelined Instruction Access active.	S
rd	R/W = 1	Read access.	D, S
rm	OPT2..OPT1 = 100	Instruction ROM access (as data).	D, S
rom	IREQT = 1	Instruction Read-only Memory access.	I, S
rs	OPT2..OPT1 = 011, 111	Reserved.	D, S
su	SUP/US = 1	Supervisor Mode.	I, D, S
tr0	TRAP0 = 0	Trap Request 0 active.	S
tr1	TRAP1 = 0	Trap Request 1 active.	S
us	SUP/US = 0	User Mode.	I, D, S
wd	OPT2..OPT1 = 000	Word-length access.	D, S
wr	R/W = 0	Write access.	D, S
wrn	WARN = 0	Warn active.	S
wt	STAT2..STAT0 = 011	Wait Mode.	S
*	DREQ = 0 and IREQ = 0	Collision, I-bus and D-bus request on same cycle.	I, D

Line 0 in the previous trace list shows the state that triggered the analyzer. The trigger state is always on line 0.

For each captured state, there is additional mnemonic informtaion (enclosed in brackets) next to the data/instruction information in the mnemonic column. Table 2-3 shows the definitions of the bracketed mnemonics.

For example, the trigger state in the previous trace shows that the processor was in the supervisor mode, that it was an instruction/data memory word read access, and that the MMU programmable outputs were zeros. The bracketed mnemonics are from the **stat** trace signals in the Instruction/Data and Data/Instruction analyzer modes and from the **data** and **stat** trace signals in the Status analyzer mode.

Switching the Analysis Mode at the Trigger Point

The "Analysis mode" and "Analysis switching signal" configuration items let you switch the analysis mode at the trigger point. For example, if you want to trigger on the same state as in the previous example, but switch to the status analysis mode at the trigger point, you would do the following:

1. Modify the general emulator configuration.
2. Modify the trigger configuration.
3. Restart the trace.

Note

When using analysis mode, the trigger condition switching must be specified in terms of the analysis mode before the trigger.

Modifying the General Emulator Configuration

To modify the "Analysis mode" and "Analysis switching signal" configuration items, select:

```
Config General
```

Move the cursor to the "Analysis mode" field and use the **<Tab>** key to select "ds", which means data/instruction mode before trigger and status mode after trigger. The "Configuring the Emulator" chapter describes the analyzer modes.

Move the cursor to the "Analysis switching signal" field and if the value "TRIG1" is not shown, use the **<Tab>** key to select it. This specifies that the internal TRIG1 signal is used to do the mode switching.

To save the general configuration, press **<End>** to move the cursor to the field in the lower right corner, and press **<Enter>**. Notice that changing values in the general configuration causes emulator execution to break into the monitor.

Modifying the Trigger Configuration

To modify the trigger configuration so that the analyzer drives the internal TRIG1 signal, select:

Config **T**rigger

Move the cursor to the “Analyzer” field in the TRIG1 portion of the display and use the < **Tab**> key to select the arrow pointing from the analyzer to the TRIG1 line. This tells the analyzer to drive TRIG1 when the trigger is found.

To save the trigger configuration, press < **End**> to move the cursor to the field in the lower right corner, and press < **Enter**> .

Restarting the Trace

Again, make sure that the program is running by selecting:

Processor **G**o **A**ddress

Enter the starting address, “Init”, and press < **Enter**> . To restart the trace, select:

Analysis **B**egin

Modify the command input word to “B” by selecting:

Memory **M**odify **W**ord

Enter “Cmd_Input= ’B’”. The status line now shows that the emulation trace is complete. Enter the following command to display the resulting trace.

Analysis **D**isplay

Press < **Enter**> three times to select the defaults. A display similar to figure 2-15 will be shown.

```
                                 Analysis
  Line    addr,H     Am29000 mnemonic
 -----   --------   -------------------------------------------------
    -1   :Exe_Cmd   cpeq      gr66,gr65,0x41      [su rom mp:0]
     0   md_Input   data rd wd:00000042           [su i/d mp:0]
     1   00002020   jmpt      gr66,cmd_rdr:Cmd_A  [su rom mp:0]
     2   00002024   cpeq      gr66,gr65,0x42      [su rom mp:0]
     3   00002028   jmpt      gr66,cmd_rdr:Cmd_B  [su rom mp:0]
     4   0000202c   nop                           [su rom mp:0]
     5   00002030   jmp       cmd_rdr:Cmd_I       [su rom mp:0]
     6   00002048             [su ph  cda rom ibq iba i/d dba rd wd mp:
     7   dr:Cmd_B   const     [su ph  cda rom ir ibq iba i/d dba rd wd
     8   00002048             [su ph  cda rom ibq iba i/d dba rd wd mp:
     9   dr:Cmd_B   const     [su ph  cda rom ir ibq iba i/d dba rd wd
    10   00002048             [su ex  cda rom ibq iba i/d dba rd wd mp:
    11   dr:Cmd_B   consth    [su ph  cda rom ir ibq iba i/d dba rd wd
    12   00002048             [su ex  cda rom ibq iba i/d dba rd wd mp:
    13   dr:Cmd_B   jmp       [su ph  cda rom ir ibq iba i/d dba rd wd
    14   00002048             [su ex  cda rom ibq iba i/d dba rd wd mp:

STATUS: Am29000--Running user program        Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Begin  Halt  CMB  Format  Trace  Display
```

Figure 2-15. Resulting Analysis Trace

Notice that it takes several states after the trigger start for the status analyzer mode to become active.

Notice also that there is no right bracket for states captured in the status analysis mode. This shows that all of the appropriate status mnemonics could not be shown. You cannot increase the width of the mnemonic column. However, you can display the signals associated with the “data” and “stat” trace labels in binary form by changing the trace display format, which allows more information to fit on the display. The next section explains how to do this.

Changing the Trace Display Format

To change the trace display format so that it shows the binary values of the signals associated with the “data” and “stat” trace labels, select:

```
Analysis Format
```

Move the cursor to the “mne” label, and use the < **Tab**> key to select the “data” label instead. Press < **Enter**> . The cursor is now in a field that specifies the number base to be used for the “data” signals in the trace display. Use the < **Tab**> key to select “bin”, and press < **Enter**> .

The cursor is now in the field for the next trace label. Use the < **Tab**> key to select the “stat” label. Press < **Enter**> . Again, use the < **Tab**> key to select the “bin” number base, and press < **Enter**> . The resulting analysis format is shown in figure 2-16.

To save the analysis format changes, press < **End**> to move the cursor to the field in the lower right corner, and press < **Enter**> .

```
                  Internal State Format Specification

    Qualify States   Clock Speed
      user             very fast


 Label Pol Base Width                     Label Pol Base Width
addr       hex                           --OFF--
data       bin                           --OFF--
stat       bin                           --OFF--
--OFF--                                  --OFF--
--OFF--                                  --OFF--
--OFF--                                  --OFF--
--OFF--                                  --OFF--
--OFF--                                  --OFF--
--OFF--                                  --OFF--
--OFF--                                  --OFF--
--OFF--                                  --OFF--
   ←↑↓→ :Interfield movement   Ctrl ←→ :Field editing     TAB :Scroll choices
STATUS: Am29000--Running in monitor          Emulation trace complete


  Use the TAB and Shift-TAB keys to select the displayed base for this label.
```

Figure 2-16. Analysis Format Specification

To display the trace using the new format, select:

Analysis **D**isplay

Move the cursor to the “address disassembly mode” field and select “address”. This field specifies whether address information, symbol information, or both types of information should appear in the “addr” column of the trace. When symbol information is shown, it also appears in the “mne” column of the trace. Because you want to display the “data” and “stat” labels as binary numbers (in other words, they contain no symbol information), the “address” selection is appropriate.

Type the number of the first available state into the starting state field and the number of the starting state plus 15 into the ending state field, and press **< Enter>** . A display similar to figure 2-17 will be shown.

```
                                   Analysis
  Line   addr,H     data,Y                              stat,Y
 -----   --------   --------------------------------   ----------------
    -1   0000201c   01100001010000100100000101000001   000000111001001
     0   00020000   00000000000000000000000001000010   101000100001001
     1   00002020   10101100000000000100001000000101   000000111001001
     2   00002024   01100001010000100100000101000010   000000111111001
     3   00002028   10101100000000000100001000001000   000000111001001
     4   0000202c   01110000010000000000000100000001   000000111001001
     5   00002030   10100000000000000000000000001011   000000111001001
     6   00002048   10100000001111111100001111100111   101110001111001
     7   00002048   00000011001110111100001111100111   101110001111001
     8   00002048   00000011001111111100001111100111   101110001111001
     9   00002048   00000011001110111100001111100111   101110001111001
    10   00002048   00000011001111111100001111100111   101110001111111
    11   00002048   00000010001110111100001111100111   101110001111001
    12   00002048   00000010001111111100001111100111   101110001111111
    13   00002048   10100000001110111100001111100111   101110001111001
    14   00002048   10100000001111111100001111100111   101110001111111

STATUS: Am29000--Running user program        Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Begin  Halt  CMB  Format  Trace  Display
```

Figure 2-17. Resulting Analysis Display

For a Complete Description

For a complete description of the HP 64700-Series analyzer, see the *HP 64700 Emulators PC Interface: Analyzer User’s Guide.*

Testing for Coverage

For each byte of emulation memory, there is an additional bit of RAM used by the emulator to provide coverage testing. When the emulator is executing the target program and accesses a byte in emulation memory, the corresponding bit of coverage memory is set. With the **M**emory **R**eport command, you can see which bytes in a range of emulation memory have (or have not) been accessed. *Beware that the results of the coverage test may be inaccurate because the Am29000/Am29050 is a pipelined processor.*

For example, suppose you want to determine how extensively some test input exercises a program (in other words, how much of the program is covered by using the test input). You can run the program with the test input and then use the **M**emory **R**eport command to see which locations in the program range were accessed.

The following commands break the processor, reset all coverage bits to “non-accessed”, and perform coverage testing on the sample program.

```
Processor Break
Memory Report Reset
Processor Go Address
```

Enter the starting address of the program, “Init”, and press **< Enter>** . To display how much of the sample program is accessed by initialization and scanning for input, select:

```
Memory Report Accessed
```

Enter the address range of the sample program, **2000..20b3@r**, press **< Enter>** , and press **< CTRL> Z** to zoom the emulation window.

Now, enter the sample program command **A** by selecting:

```
Memory Modify Word
```

Enter **Cmd_Input= ’A’**, press **< Enter>** , and run the memory report command again by selecting:

```
Memory Report Accessed
```

Enter the sample program command **B** by selecting:

`Memory Modify Word`

Enter **Cmd_Input= 'B'**, press < **Enter**> , and run the memory report command again by selecting:

`Memory Report Accessed`

Finally, enter an invalid command by selecting:

`Memory Modify Word`

Enter **Cmd_Input= 'C'**, press < **Enter**> , and run the memory report command again by selecting:

`Memory Report Accessed`

Notice, in figure 2-18, that more of the sample program address range is covered after each command is entered.

```
                                    Emulation
Status   Address
------   ----------
 Clear   00000205c@r

000002000..00000202b@r
percentage of memory accessed:  % 24.4

000002000..00000202b@r
000002034..00000204b@r
000002068..0000020b3@r
percentage of memory accessed:  % 80.0

000002000..00000205f@r
000002068..0000020b3@r
percentage of memory accessed:  % 95.5

000002000..0000020b3@r
percentage of memory accessed:  % 100.0

STATUS: Am29000--Running user program        Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Display  Modify  Load  Store  Copy  Find  Report
```

Figure 2-18. Results of Memory Coverage Report

Copying Memory

You can copy the contents of one range of memory to another. This is a useful feature to test things like the relocatability of programs. To test whether the sample program is relocatable within the same segment, copy the program to an unused, but mapped, area of emulation memory. For example, select:

```
Memory Copy
```

Enter **2000..20b3@r** as the source memory range to be copied, and enter **3000@r** as the destination address.

To verify that the program is relocatable, run it from its new address by selecting:

```
Processor Go Address
```

Enter **3000@r**. The status line shows that the emulator is "Running user program". You may want to trace program execution or enter valid and invalid commands and search the message destination area (as shown earlier in this chapter) to verify that the program is working correctly from its new address.

Resetting the Emulator

To reset the emulator, select:

```
Processor Reset Hold
```

The emulator is held in a reset state (suspended) until a **P**rocessor **B**reak, **P**rocessor **G**o, or **P**rocessor **S**tep command is entered. If there was a previous **P**rocessor **C**MB **G**o command, a CMB execute signal will also cause the emulator to run.

You can also specify that the emulator begin executing in the monitor after reset instead of remaining in the suspended state. To do this, select:

```
Processor Reset Monitor
```

Exiting the PC Interface

There are several different ways to exit the PC Interface. You can exit the PC Interface using the “locked” option which specifies that the current configuration will be present next time you start up the PC Interface. You can select this option as follows.

```
System Exit Locked
```

Another way to exit the PC Interface is with the “unlocked” option which specifies that the default configuration will be present the next time you start up the PC Interface. You can select this option with the following command.

```
System Exit Unlocked
```

The last way you can exit the PC Interface is the with the “no save” option. This option is similar to the “locked” option except that it specifies that the configuration present when you entered the PC Interface will be present the next time you start the PC Interface. You can select this option with the following command.

```
System Exit No_save
```

For more information on exiting the PC Interface see the *PC Interface Reference* manual.

Notes

3

Using the Emulator — The Basics

Target System Design Considerations

The HP 64774 29000/29050 emulator *requires* a target system to operate. In other words, the emulator must always be “in-circuit”. The target system may be the harbor box that is shipped with the emulator or your own target system. Specifically, you must provide + 5V and a clock signal *must* be provided. If these are not present, the emulator cannot access emulation memory.

When the emulator is connected to your target system, you can power up the emulator when the target system is powered down. This is not possible when using the harbor box as the target system.

Access for Emulator Probe

The target system must be able to accept a 169 pin PGA (Pin Grid Array) processor package.

There must be enough clearance in the target system to allow the HP 64774 emulation probe to be plugged in and the cable routed from the target system to the emulator control box. See figure 3-1 for probe and cable dimensions and pin orientation.

Probe Power Requirements

An additional 750 mA of + 5V must be available at the processor socket to power the HP 64774 probe. This guards against latch-up problems in the HP 64774 probe circuitry.

Probe Cooling

We recommend that you provide 100 lfm of forced air cooling for the HP 64774 probe, especially at higher system clock rates where power dissipation is greater.

Disable Target Data Bus Buffers

To provide maximum performance, there are no buffers between the target system’s I and D busses and the emulation processor.

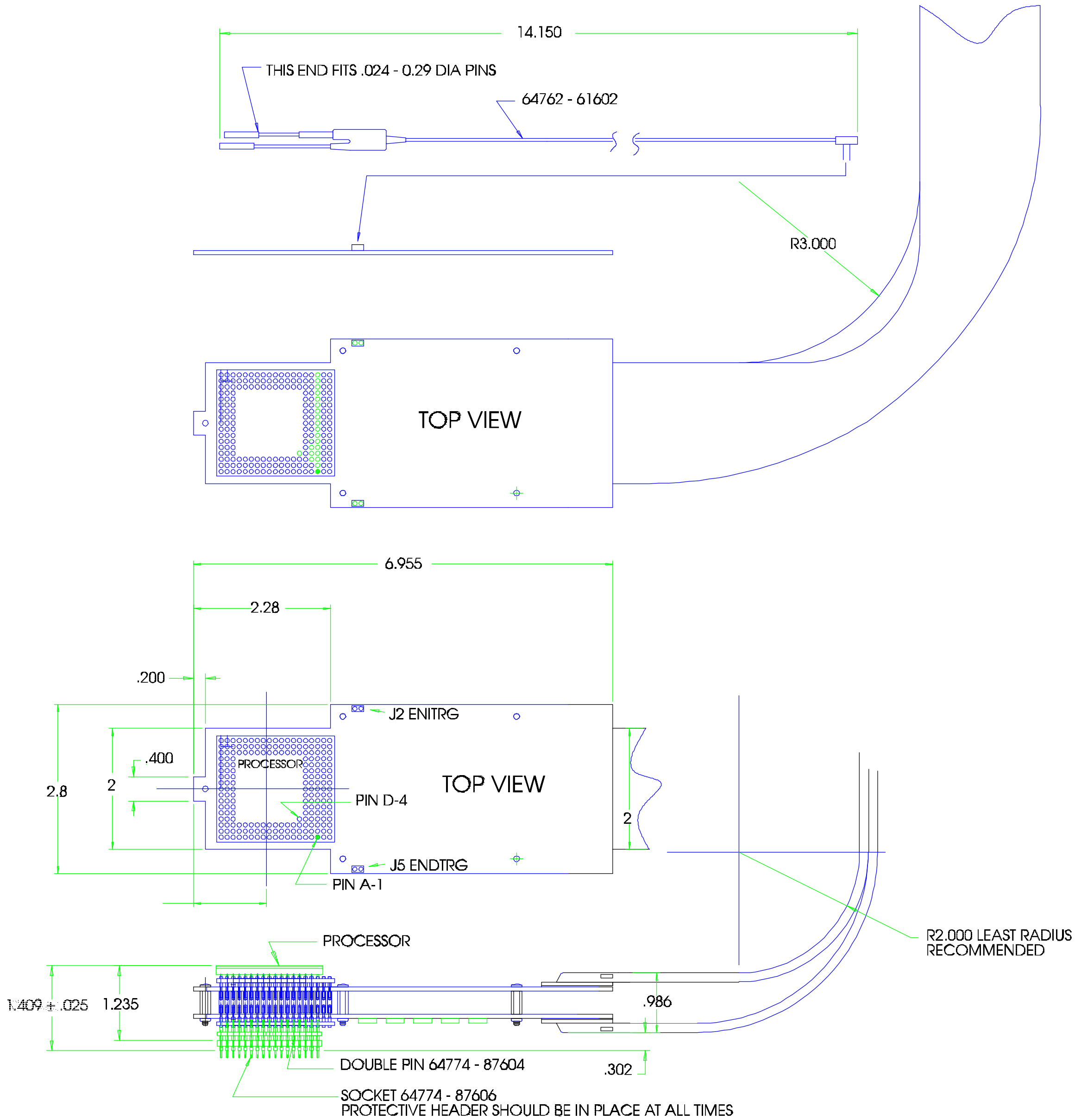

Figure 3-1. HP 64774 Emulator Probe Dimensions

Provisions *must* be made in the target system to disable the target system data buffers on both the I (instruction) and D (data) busses.

When Using Emulation Memory

If you need to use the emulation memory (sometimes referred to as overlay memory) of the HP 64774, then you must connect two control signals from the HP 64774 probe to the target system to selectively disable the I and D data bus buffers. These two signals, ENITRG (Enable Instruction Target data buffers) and ENDTRG (Enable Data Target data buffers) are TTL level signals that are driven low to tell the target system that it must disable the buffers on that bus. These signals remain low until the emulator no longer needs the bus. ENDTRG is asserted for all emulation memory data accesses, of whatever type [(opt 110, 000, 001, 010)], in addition to ADAPT cycles.

The leads provided for the ENITRG and ENDTRG signals will plug on to a standard IC clip with the following characteristics:

Pin diameters of 0.024 to 0.029 inches.

Minimum pin spacing of 0.100 inches.

Pin length of 0.200 to 0.300 inches.

These outputs can sink 20 mA of current in the low state, and the leads have an electrical impedance of about 90 Ω. For best performance you should terminate these leads in the target system by about 150 Ω to 2.7V (this can be done with a resistive divider of 270 Ω to + 5V and 330 Ω to ground). See figure 3-1 for lead dimensions and connector specifications.

Figure 3-2 shows the timing requirements of the ENITRG and ENDTRG signals. These signals are synchronous with the positive edge of SYSCLK and are valid 10 ns after that edge. Notice that the emulator drives only the last half of each cycle that ENITRG is held false. Also note that while the emulator may drive data during a cycle, it is only guaranteed to be valid when $\overline{\text{xRDY}}$ is true.

The ENITRG and ENDTRG signals may also be used to terminate burst transfers to/from the target system for the bus in question.

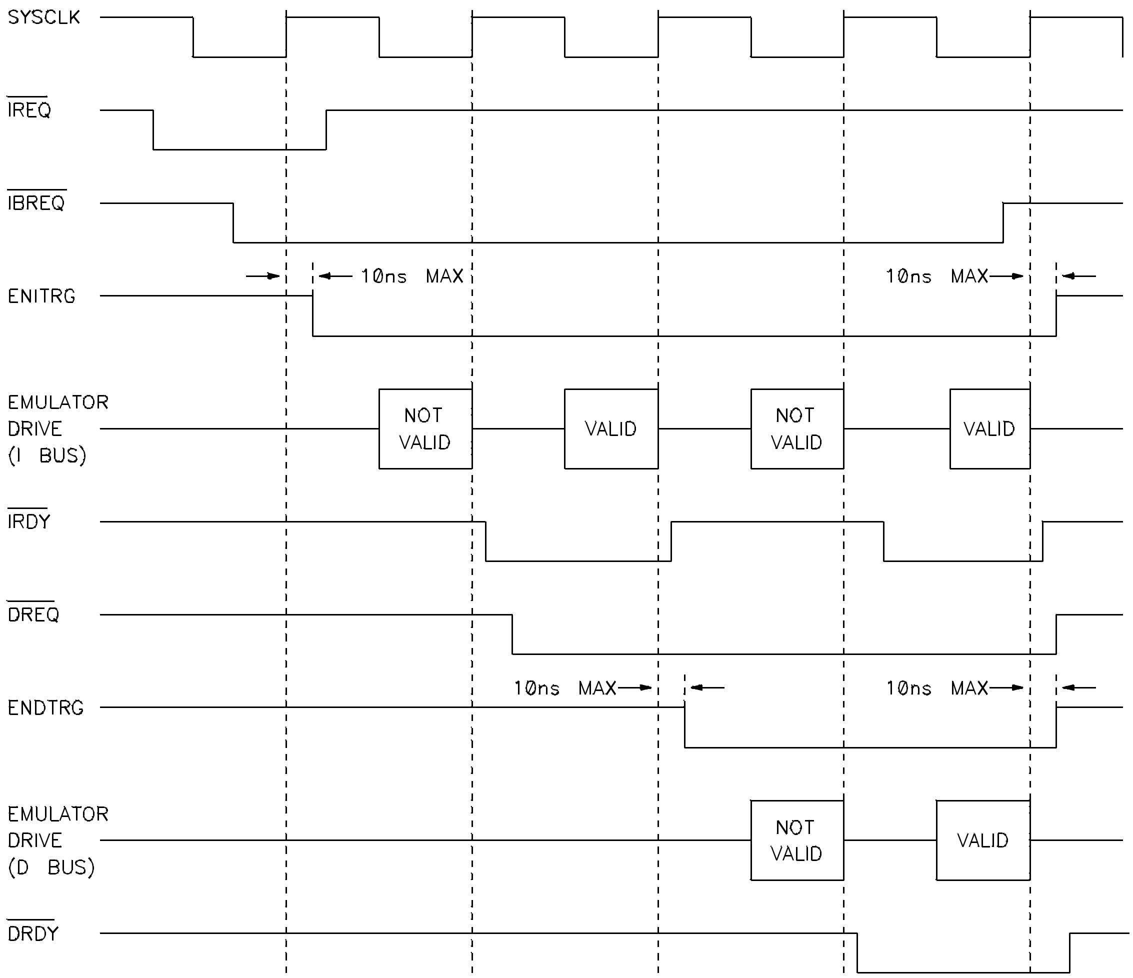

Figure 3-2. ENITRG and ENDTRG Timing

When Not Using Emulation Memory

If you don't need emulation memory, then the target system buffers may be disabled by monitoring the alignment pin (pin D4 or 169) on the Am290xx socket and disabling the buffers on both the I and D busses when this signal goes LOW. The target system must respond within 100 ns to the assertion/de-assertion of the alignment pin. If the buffers will be disabled, as outlined in the previous "When Using Emulation Memory" section (using the ENITRG and ENDTRG signal wires from the probe) then monitoring the alignment pin is not needed.

Processor Signal Considerations

The HP 64774 emulator uses some of the processor signals to provide emulation features. So, these signals are affected by the emulator.

$\overline{\text{WARN}}$ Line

The $\overline{\text{WARN}}$ line is intercepted by the HP 64774 and *not* driven to the emulation processor. The HP 64774 can be configured to break processor execution when the $\overline{\text{WARN}}$ signal is driven true.

Control Lines Intercepted

To control the processor, the emulator intercepts the following signals:

> $\overline{\text{IRDY}}$, $\overline{\text{DRDY}}$, $\overline{\text{BREQ}}$, $\overline{\text{IBACK}}$, $\overline{\text{DBACK}}$, $\overline{\text{PEN}}$, $\overline{\text{IERR}}$, $\overline{\text{DERR}}$, $\overline{\text{RESET}}$, INCLK

AND/OR arrays of very fast logic are used to minimize the delays. These delays are typically 1 ns, and about 1.5 ns in the worst case.

SYSCLK

SYSCLK is more heavily loaded than other signals — about 3 TTL loads and 50 pF.

Other Signals

All signals except the intercepted control lines and SYSCLK are loaded with 1 FCT load, and 30 pF, and are pulled up to + 3.5V through a 100K Ω resistor. Capacitance is about 30 pF.

Effects of Using Emulation Memory

Using emulation memory has the following effects:

- Effects of emulation memory has when disabling target data bus buffers (see the previous section “Disable Target Data Bus Buffers”).
- Pipelined access mode is disabled ($\overline{\text{PEN}}$ line is blocked).
- Single clock cycle accesses on the initial access ($\overline{\text{xREQ}}$ true) are not supported at clock rates above 25 MHz.

Effects of the Background Monitor

Emulator execution temporarily breaks into the background monitor when you use emulation commands to display processor registers or target system memory. The background monitor affects the emulation processor in these ways:

- Interrupts are not serviced while the emulator is in the background monitor.
- Bus requests may be held off for about 100 us while in the background monitor.

The HP 64774 29000/29050 emulator does not have a monitor program, as is common with other HP 64700 Series Emulators. The HP 64774 utilizes an 80186 microprocessor to accomplish the duties associated with a background monitor. Therefore, no emulation or target memory resources are required. You do not have to be concerned about overlaying target memory or emulation memory on top of a monitor program.

The on-board 80186 accomplishes the “monitor” functions by controlling the CNTL0 and CNTL1 signals. By asserting different combinations of the two signals, the 29000/29050 can be placed in one of four states: RUN, HALT, STEP, and LOAD TEST INSTRUCTION. It is by means of the LOAD TEST INSTRUCTION that the emulator can examine and modify the internal state of the processor without altering the processor’s instruction stream.

Advanced Micro Devices, Inc., provides a tool called the MON29K Target Resident Monitor. The 64774 emulator does not use the MON29K monitor.

Memory Accesses

Depending on how the emulator is configured (the clock speed and whether emulation memory is being used) the emulator may insert wait states on emulation memory and target memory accesses. See the “Emulator Speed Configuration” section of the “Configuring the Emulator” chapter for complete details of how memory accesses are affected by the emulator.

Plugging the Emulator into a Target System

The emulator probe has a 169-pin Pin Grid Array (PGA) connector.

Caution

Possible Damage to the Emulator Probe. The emulator probe comes with a pin extender. *Do not use the probe without a pin extender installed.* Replacing a broken pin extender is much less expensive than replacing the emulator probe.

Don’t use more than one pin extender, unless it is needed for mechanical clearance, because pin extenders degrade signal quality.

The emulator probe is also provided with a foam pin protector to: (1) protect the probe from damage due to electrostatic discharge (ESD), and (2) protect the delicate gold-plated pins of the probe connector from impact damage.

Caution

Possible Damage to the Emulator Probe. The emulation probe contains devices that can be damaged by static discharge. You should take precautions before handling the microprocessor connector attached to the end of the probe cable to avoid damaging the internal components of the probe.

Caution

Possible Damage to the Emulator. Make sure target system power is OFF before installing the emulator probe into the target system. Do not install the emulator probe into the processor socket with power applied to the target system.

Caution

Damage to the Emulator Probe will Result if the Probe is Incorrectly Installed. Make sure pin 1 of the probe connector is aligned with pin 1 of the socket. When installing the emulation probe, be sure that the probe is inserted into the processor socket so that the alignment pin, D4, of the connector aligns with that pin of the socket (as shown in the figure below).

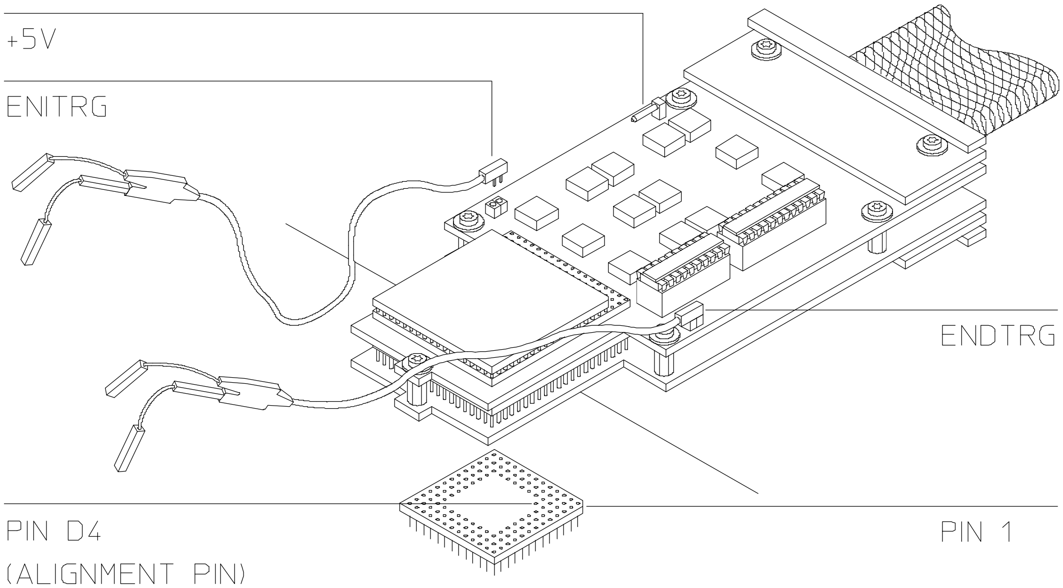

Figure 3-3. Plugging into a Target System

There are two extra rows of holes on top of the emulator probe microprocessor socket (see figure 3-3), and two extra rows of pins on the bottom of the probe. When attaching the emulator probe to a target system microprocessor socket, make sure you connect the forwardmost rows of pins, toward the tip of the probe, to the microprocessor socket.

Notes

4

Using the Emulator — In Depth

Introduction

The “Getting Started” chapter shows you how to use the basic features of the 29000/29050 emulator. This chapter describes some of those features in more detail. Also, this chapter describes features of the emulator that were not covered in the “Getting Started” chapter.

This chapter contains information on the following topics:

- Mapping memory.
- Modifying and displaying memory in mnemonic format.
- Storing the contents of memory into absolute files.
- Modifying and displaying registers.
- Making coordinated measurements.

Prerequisites

Before performing the tasks described in this chapter, you should be familiar with general emulator operation. See the *Concepts of Emulation and Analysis* manual and the “Getting Started” chapter of this manual.

Mapping Memory

The memory mapper tells the emulator how to access memory locations in a particular range.

- The emulator needs to know whether memory is located in the emulator or in the target system.
- The emulator also needs to know whether the memory is RAM or ROM, which locations of physical emulation memory are used for a particular address range, which ranges are overlaid, and whether word, half-word, or byte accesses should be used for particular ranges in target memory.

A total of 15 ranges can be mapped. Each range that you map is associated with a mapper term. To enter the memory mapper, select:

Config Map Modify

Figure 4-1 shows the memory map configuration display. Notice that there are three fields associated with each mapper term: address range, memory type, and attribute.

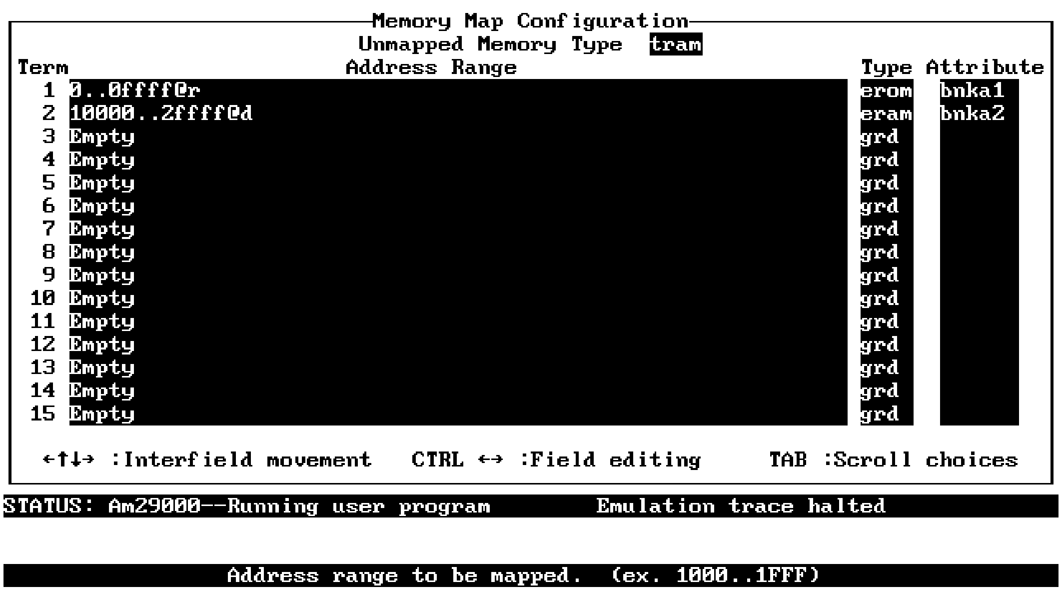

Figure 4-1. Memory Map Configuration

Address Ranges

The range specified when defining a mapper term may be any valid subrange of the processor address space. The starting address will always be masked to be the beginning of a 64K byte block. The ending address will always be modified to be the end of a 64K byte block.

Address Space Designators

Because separate blocks of memory can be connected to the I-bus and D-bus of the 29000/29050 and because memory can be accessed differently depending on whether it's ROM or RAM, an address space designator must be supplied with address ranges. If an address space designator is not supplied with an address range, the default address space designator (see the "Default address space" configuration item) will be used. The address space designators are:

@i	Instruction bus address space (IREQT = 0).
@d	Data bus address space (DREQT = 00, OPT = 000,001,010).
@id	Instruction and data bus address space (combination of "@i" and "@d").
@r	Instruction ROM on the instruction bus (IREQT = 1).
@a	Instruction ROM on the data bus (DREQT = 00, OPT = 100).
@ra	Instruction ROM on instruction and data bus (combination of "@r" and "@a").

(The IREQT signal is a reflection of the ROM Enable bit (8) in the CPS register. DREQT = 00 means an instruction/data access, as opposed to I/O or coprocessor accesses. OPT is taken from bits 18..16 of the load or store instruction opcode.)

A maximum of 8 I-bus connections and 7 D-bus connections are allowed. I-bus connections are made when the "@i" or "@r" designators are used. D-bus connections are made when the "@d" or "@a" designators are used.

When the “@id” or “@ra” designators are used, two connections are made, one to the I-bus and one to the D-bus (see figure 4-2).

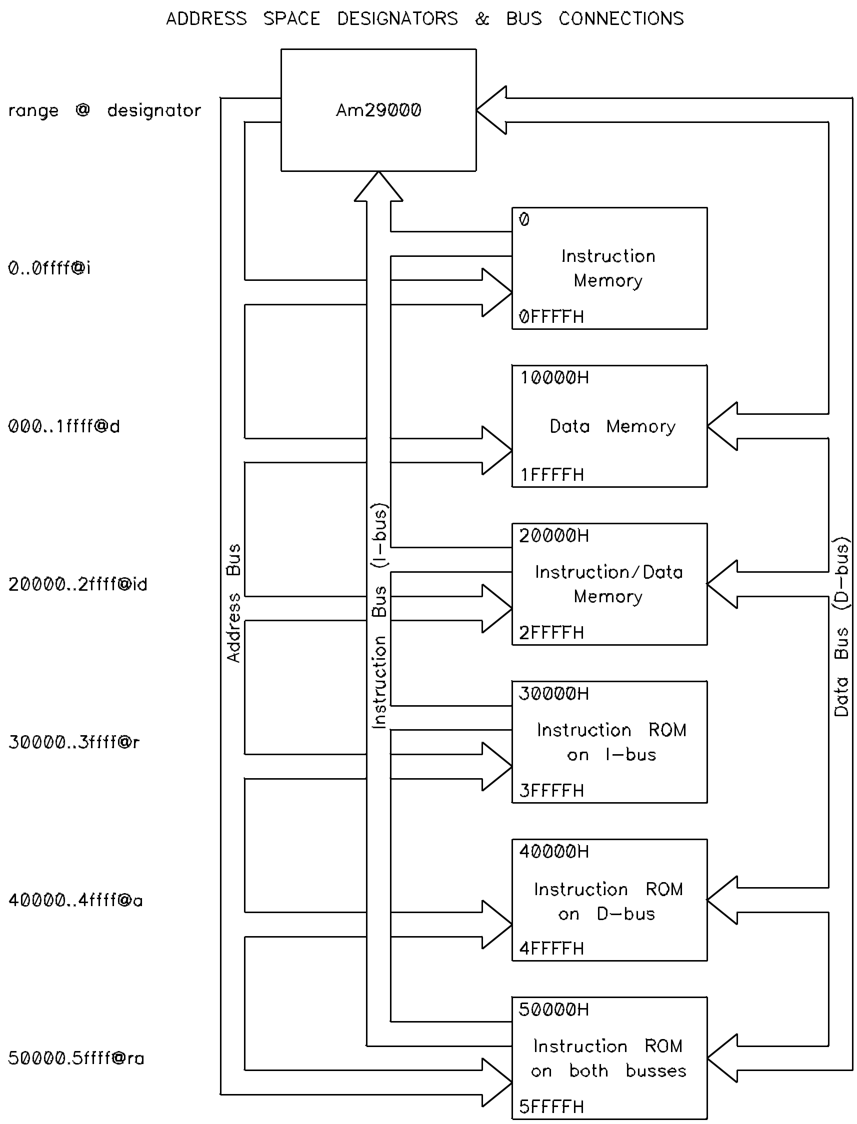

Figure 4-2. 29000/29050 Addresses & Bus Connections

If, while operating the emulator, you see the status message "Slow I-bus cycles", it simply means that the emulator does not see any activity on the instruction bus.

Types of Memory

When mapping an address range, you must classify the type of memory as either emulation RAM (eram), emulation ROM (erom), target RAM (tram), target ROM (trom), or guarded memory (grd).

Accesses to guarded memory locations will cause emulator execution to break to the monitor. Writes to ROM will cause emulator execution to break to the monitor if the "Brk on write to ROM" general configuration item is "on".

Attributes

When mapping emulation memory ranges, you must include an attribute that names the bank and block of memory into which that range should be mapped.

When mapping ranges of target memory, attributes can be included to specify locations that have different I-bus and D-bus addresses. Also, attributes can be used to specify the access mode to be used with a range of target memory.

Emulation Memory Available

The HP 64774 emulator can have 0, 1, or 2 banks of emulation memory. There are two blocks of memory in each bank. Emulation memory is mapped in ranges of at least 64 Kbytes, beginning on 64 Kbyte boundaries (see figure 4-3).

If the banks have 64Kx4 static RAMs, each bank contains 512 Kbytes, and each block contains 256 Kbytes (40000H).

If the banks have 256Kx4 static RAMs, each bank contains 2 Mbytes, and each block contains 1 Mbytes (100000H). Though there is four times as much memory when 256Kx4 RAMs are used, ranges can still be mapped at a resolution of 64 Kbytes.

Because each bank has its own memory arbiter, the I-bus and D-bus can be configured to operate independently. You can optimize each bank for I-bus or D-bus accesses (see the "bnka" and "bnkb" configuration items).

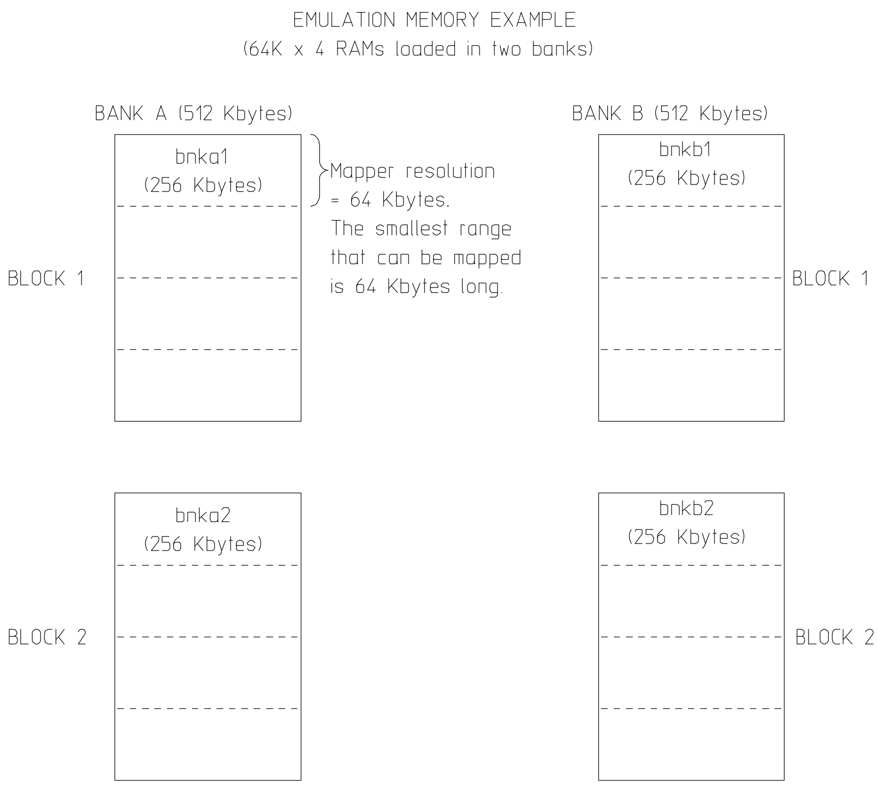

Figure 4-3. Emulation Memory Example

Emulation Memory Attributes

The “bnka1”, “bnka2”, “bnkb1”, and “bnkb2” attributes assign ranges to a particular bank and block of emulation memory. These attributes allow you to map ranges into banks optimized for either I-bus or D-bus accesses, and they allow you to overlay memory ranges.

Overlaying Ranges in Emulation Memory. Suppose your emulator contains one bank of memory loaded with 64Kx4 RAMs (0.5 Mbyte, 256 Kbytes per block). This means that a block of emulation memory contains 40000H bytes (0 through 3FFFFH) and there are 18 significant address lines to that memory. Therefore, all ranges (mapped to that block) whose 18 least significant bits are the same will be overlaid. For example, consider the following memory map configuration:

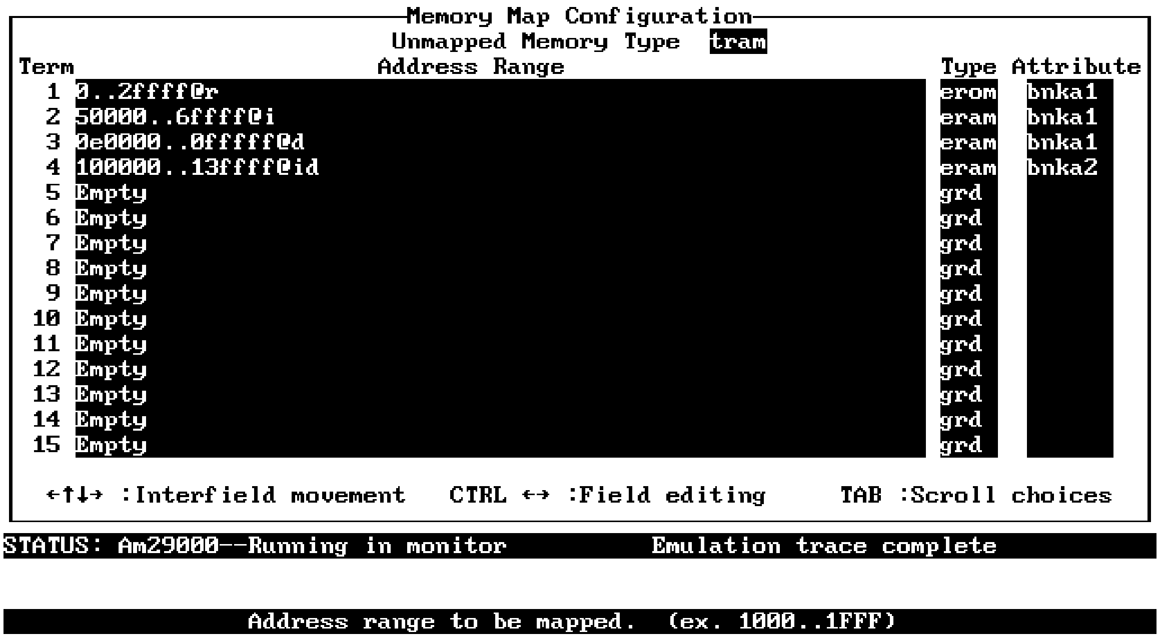

Figure 4-4. Example Memory Map Configuration

Parts of the first three ranges above are overlaid because their 18 least significant bits are the same. While the 18 least significant bits of the last range are the same as those of the first three ranges, the last range is not overlaid because it is mapped to a different block of emulation memory.

Displaying Overlaid Ranges. You can display the ranges that are overlaid in the 64 Kbyte regions of emulation memory, by selecting:

Config, **M**ap, **D**isplay

The display corresponding to the previous memory map configuration is shown below.

```
                                    Emulation
mapped memory.  This permits easy determination of which addresses are
overlaid.  The numbers on the left are the 64K regions for the particular
block of emulation memory.  If an access size has been specified, it
appears next.  The memory ranges mapped to that region are then displayed.
Only regions with an applicable map term will be shown.
bnka1:  256 Kbytes
0          00000000..0000ffff@r
1          00010000..0001ffff@r
00050000..0005ffff@i
2          00020000..0002ffff@r
00060000..0006ffff@i
000e0000..000effff@d
3          000f0000..000fffff@d
bnka2:  256 Kbytes
0          00100000..0010ffff@id
1          00110000..0011ffff@id
2          00120000..0012ffff@id
3          00130000..0013ffff@id

STATUS: Am29000--Running in monitor          Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Load  Store  General  Map  Trigger  Key_macro
```

Figure 4-5. Memory Map Display

Target Memory Attributes

While the emulator can assume that all ranges not mapped to emulation memory are in the target system (by mapping all other memory as target system emulation ram), there is still information that the emulator must know about accessing particular ranges in the target system.

Ranges with Different I-bus and D-bus Addresses. The "blk1" through "blk8" attributes are for use with target RAM ranges where the same physical memory is assigned to different I-bus and D-bus addresses. Up to eight of these ranges can be mapped. The following memory map configuration shows how these attributes can be used.

All ranges mapped using the same bus target memory attribute must be the same size.

Access Mode Attributes

The memory mapper allows you to specify an access mode for individual ranges of memory. The access mode tells the emulator whether to use word, half-word, or byte accesses when reading or writing memory. For example, the access mode is used, when you display or modify target memory locations or when you load absolute files into target memory.

Access mode attributes for emulation memory are needed only if little endian byte ordering is used. The access mode is used to ensure proper loading and dumping of the memory.

The letter “w”, “h”, or “b” may be used as an attribute or appended to a “blk” or “bnk” attribute to indicate that the memory should always be accessed as words, half-words, or bytes. This will override the access mode set in the “Access width” field of the general emulator configuration. Access mode attributes are useful if loading a file containing several data areas whose memory should be accessed differently.

Memory mapped as instruction ROM, or instruction memory connected only to the I-bus cannot have an access size attribute.

Also, memory mapped as “other” will always use the access size set in the “Access width” field of the general emulator configuration.

Modifying and Displaying Memory

You modify or display memory by selecting the following commands:

```
Memory Modify ...
Memory Display ...
```

When you select these commands, you can display or modify the following size memory locations:

`Byte`	8 bits
`Half`	16 bits
`Word`	32 bits

Also, 29000/29050 emulator allows you to display or modify processor memory space using floating-point values. Floating-point values must be displayed or modified at addresses that are multiples of four bytes.

`Float`	32-bit float format
`Double`	64-bit double format
`Extended`	80-bit extended format
`Quad`	128-bit quad format

Note

When modifying memory with floating point values of type “extended” or “quad”, expressions should not contain or evaluate to values outside the range of type “double”. Single values (not in an expression) can have any legal value for that type.

When displaying memory, you have the following options:

`Mnemonic`	Assembly language mnemonics.
`Repetitively`	Performs last memory command repetitively.

After you have chosen from the options above, you are given a field in which to specify the addresses to be displayed or the addresses and the new values of the locations to be modified.

Storing Memory Contents to Absolute Files

The “Getting Started” chapter shows you how to load absolute files into emulation or target system memory. You can also store emulation or target system memory to an absolute file with the following command.

```
Memory Store
```

Note

You can name the absolute file with a total of eight alphanumeric characters, and include an extension of up to three alphanumeric characters.

Caution

File may be overwritten! The **M**emory **S**tore command writes over an existing file if it has the same name that is specified with the command. You may want to verify beforehand that the specified filename does not already exist.

Modifying and Displaying Registers

You modify or display registers by selecting the following commands:

```
Registers Display ...
Registers Modify ...
```

When you select those commands, you have the following options:

Verbose	Using this option in register commands causes bit fields of the special-purpose registers to be separated and labeled.
Terse	Using this option in register commands causes only hexadecimal contents of the registers to be shown.

The 29000/29050 emulator allows you to display or modify processor registers using floating point values.

Float	32-bit float format
Double	64-bit double format
Extended	80-bit extended format
Quad	128-bit quad format

After you specify the type of register access, you can specify which registers to display or modify.

Basic	Selects the basic registers. These registers include the current processor status register, the configuration register, and program counter registers 0 and 1.
Class	Selects a class of registers (see table 4-1).

Range	Allows you to display or modify a range of the general purpose registers.

Register Names and Classes

Table 4-1 lists the register names and classes that may be used with the display/modify register commands. Registers that apply only to the 29050 are marked with “*”. Commands that display the execute and decode addresses are shown at the end of this table.

Table 4-1. Register Names and Classes

< REGCLASS>	**< REGNAME>**	**Description**
glob	**gr1, gr2*, gr3*, gr64..gr127**	Global Registers
loc	**lr0..lr127**	Local Registers
gen	**r1, r2*, r3*, r64..r255** (combination of **glob** and **loc** register classes, used when accessing absolute register numbers)	General-Purpose Registers
prot (Protected Special-Purpose Registers)	**vab** **ops** **cps** **cfg** **cha** **chd** **chc** **rbp** **tmc** **tmr** **pc0, pc1, pc2** **mmu** **lru**	Vector Area Base Address Old Processor Status Current Processor Status Configuration Channel Address Channel Data Channel Control Register Bank Protect Timer Counter Timer Reload Program Counter 0, 1, 2 MMU Configuration LRU Recommendation

Table 4-1. Register Names and Classes (Cont'd)

< REGCLASS>	< REGNAME>	Description
* **prot** (Protected Special-Purpose Registers)	**rsn** **rma0** **rmc0** **rma1** **rmc1** **spc0** **spc1** **spc2** **iba0** **ibc0** **iba1** **ibc1**	Reason Vector Region Mapping Address 0 Region Mapping Control 0 Region Mapping Address 1 Region Mapping Control 1 Shadow Program Counter 0 Shadow Program Counter 1 Shadow Program Counter 2 Instruction Breakpoint Address 0 Instruction Breakpoint Control 0 Instruction Breakpoint Address 1 Instruction Breakpoint Control 1
unprot (Unprotected Special-Purpose Registers)	**ipc, ipa, ipb** **q** **alu** **bp** **fc** **cr** **fpe *** **inte *** **fps *** **exop ***	Indirect Pointer C, A, B Q ALU Status Byte Pointer Funnel Shift Count Load/Store Count Remaining Floating-Point Environment Integer Environment Floating-Point Status Exception Opcode
spec	(combination of **prot** and **unprot** register classes)	Special-Purpose Registers
tlb	**tlb0..tlb127**	Translation Look-Aside Buffer

Table 4-1. Register Names and Classes (Cont'd)

< REGCLASS>	**< REGNAME>**	**Description**
coproc (Coprocessor Registers - only available if coprocessor is present and the "Use coprocessor" configuration item is set to "on".)	**instr** **i_temp** **r** **s** **r_temp** **s_temp** **status** **precis** **rf0-rf7** **mode** **mode_hi** **mode_lo** **f** **flags**	Instruction I-Temp R S R-Temp S-Temp Status Register Precision Register RF0-RF7 (register file) Mode Mode - high 32 bits Mode - low 32 bits F (display only) Flag Register (display only)
* **acc** (for 29050 only)	**acc0, acc1, acc2, acc3**	Floating point accumulator registers
all	(combination of **glob**, **loc**, **spec**, **tlb**, **coproc**, and **acc** register classes)	All Registers
(no class specified)	**gr1, gr64..gr111, lr0..lr47** and selected special registers,	
Commands	The **da** command displays the decode address value. The **ea** command displays the execute address value.	True decode address (may be different from **pc0** if processor is frozen) True execute address (may be different from **pc1** if processor is frozen)

When an interrupt or trap is taken, the freeze bit is set (thus the processor is frozen), and PCO and PC1 contain the addresses of the instructions in the decode and execute stages of the pipeline.

Making Coordinated Measurements

Coordinated measurements are synchronous measurements between multiple emulators or analyzers. Coordinated measurements can be made between HP 64700-Series emulators that communicate over the Coordinated Measurement Bus (CMB). Coordinated measurements can also be made between an emulator and some other instrument connected to the BNC connector.

This section describes coordinated measurements made from the PC Interface. These types of coordinated measurements are:

- Running the emulator on receipt of the CMB /EXECUTE signal.
- Using the analyzer trigger to break emulator execution into the monitor.

CMB Signals

Three signal lines on the CMB are active and serve the following functions:

/TRIGGER Active low. The analyzer trigger line on the CMB and on the BNC serve the same logical purpose. They provide a way for the analyzer to drive its trigger signal out of the system or for external trigger signals to arm the analyzer or break the emulator into its monitor.

READY Active high. This line is for synchronized, multi-emulator start and stop. When CMB run control interaction is enabled, all emulators must break to background on receipt of a false READY signal and will not return to foreground until this line is true.

/EXECUTE Active low. This line serves as a global interrupt signal. On receipt of an enabled /EXECUTE signal, each emulator is to interrupt whatever it is doing and execute a previously defined process. This process might run the emulator or start a trace measurement.

Running the Emulator at /EXECUTE

Before you can have the emulator respond to the /EXECUTE signal, you must enable CMB interaction. To do this, select:

Config **G**eneral

Use the arrow keys to move the cursor to the “CMB Interaction” field, and use the < **Tab**> key to select “on”. Use the < **Enter**> key to exit out of the lower right-hand field in the configuration display.

To specify that the emulator begin executing a program on receipt of the /EXECUTE signal, select:

Processor **C**MB **G**o

Now you may either select the current program counter, or you may select a specific address.

The command you enter is saved and is executed when the /EXECUTE signal becomes active. Also, you will see the message “ALERT: CMB execute; run started”.

Using the Analyzer Trigger to Break into the Monitor

To cause emulator execution to break into the monitor when the analyzer trigger condition is found, you must modify the trigger configuration. To access the trigger configuration, select:

Config **T**rigger

The trigger configuration display contains two diagrams, one for each of the internal TRIG1 and TRIG2 signals.

To use the internal TRIG1 signal to connect the analyzer trigger to the emulator break line, move the cursor to the highlighted “Analyzer” field in the TRIG1 portion of the display, and use the < **Tab**> key to select the arrow that points away from the analyzer and towards TRIG1. This causes the analyzer to drive TRIG1 when the trigger is found.

Next, move the cursor to the highlighted “Emulator” field and use the < **Tab**> key to select the arrow pointing towards the emulator. This specifies that emulator execution will break into the monitor when the TRIG1 signal is driven. Figure 4-6 shows the trigger configuration display.

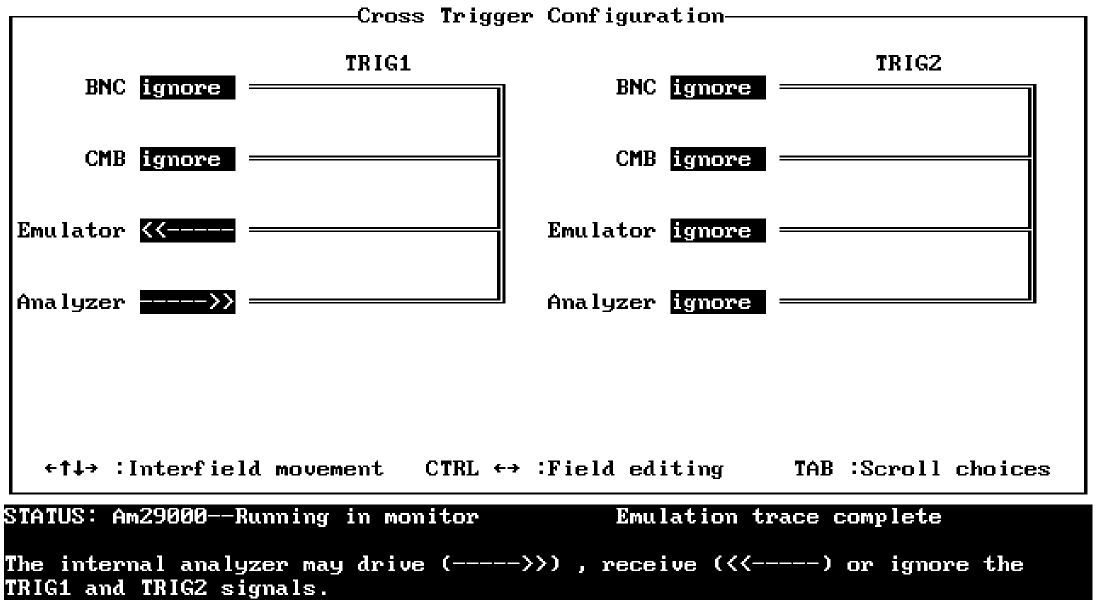

Figure 4-6. Trigger Configuration

5

Configuring the Emulator

Introduction

This chapter describes the HP 64774 emulator configuration options. To access the emulator configuration options, select:

Config **G**eneral

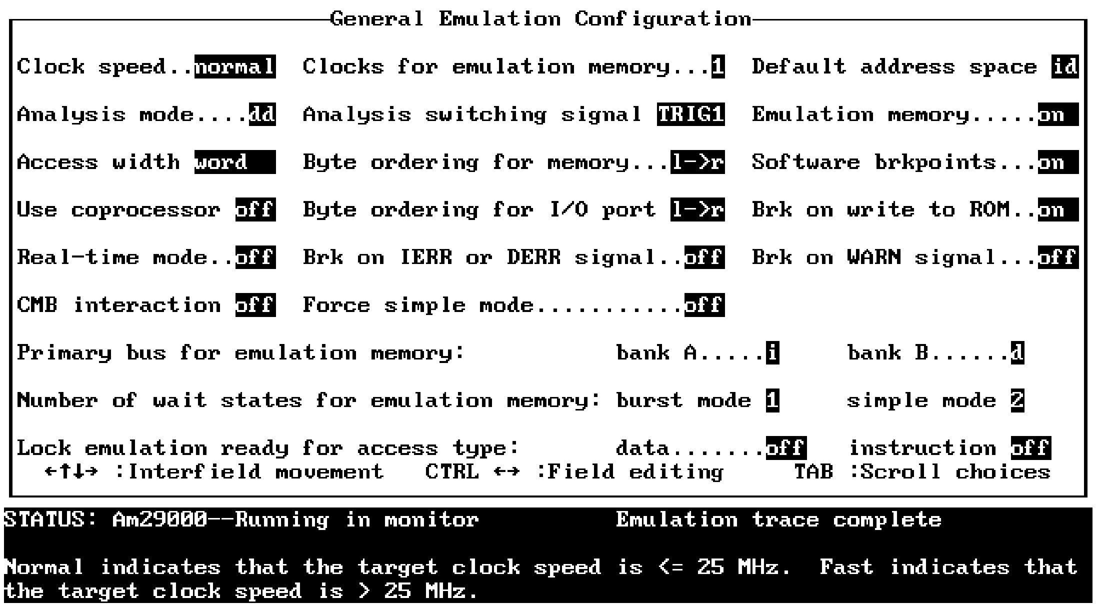

Figure 5-1. General Emulator Configuration

These configuration items are described in this chapter.

- Emulator Speed Configuration
  - Target clock speed selection
    - Emulation memory selection
    - Clock cycles for emulation memory accesses
    - Wait-states for emulation memory
  - Restrict to real-time runs
- Emulation Memory Configuration
  - Primary bus selection
  - Lock ready for bus accesses selections
- Analysis Mode Configuration
  - Analysis mode
    - Analysis mode switching signal
- Emulator Break Configuration
  - Software breakpoints
  - Break on writes to ROM selection
  - Break on IERR or DERR signals selection
  - Break on WARN signal selection
- General Emulator Configuration
  - Coprocessor access
  - Byte ordering for memory and I/O ports
  - Force simple mode accesses
  - Access width selection
  - Default address space
  - CMB interaction

When you position the cursor to a configuration item, a brief description of the item appears at the bottom of the display.

Note

You could use the **S**ystem **T**erminal window to modify the emulator configuration. However, if you do this, some PC Interface features may no longer work properly. You should only modify the emulator configuration by using the options presented in the PC Interface.

Emulator Speed Configuration

Note

The 29000/29050 emulator can execute in a target system at full clock speed (33 MHz); however, the analyzer may provide incorrect data above 25 MHz. As a result, Hewlett-Packard only supports operation of the HP 64774 emulator with analysis at clock speeds up to 25 MHz.

The emulator makes adjustments based on the speed of the target system clock, whether emulation memory is used or not, and, if emulation memory is used, the access time of the emulation memory modules installed. (The 2 Mbyte memory modules have a slower access time than the 0.5 Mbyte memory modules.) See table 5-1 to determine the correct settings for the “clock speed”, “emulation memory”, and “clocks for emulation memory” configuration items. These settings determine the minimum number of clocks required for any given cycle type (see table 5-2). Additional clock cycles (wait states) can be inserted using the “number of wait states for emulation memory” configuration items.

You can also specify whether or not the emulator should be restricted to real-time execution.

Clock Speed

The emulator will adjust the number of wait states based on whether the target system clock speed is less than or equal to 25 MHz or greater than 25 MHz. The wait states are inserted for mapper address translation (when the “emulation memory” configuration item is set to “on”).

normal (emulation memory = on)	Use this setting when the target clock speed is less than or equal to 25 MHz. No wait states are required for mapper address translation of emulation or target memory accesses.

fast (emulation memory = on)	Use this setting when the target clock speed is greater than 25 MHz. One wait state is required for mapper address translation of emulation or target memory accesses.

When emulation memory is disabled (the “emulation memory” configuration item is set to “off”), this configuration item does not affect the emulator.

Emulation Memory

The “emulation memory” configuration item allows you to enable or disable emulation memory and the memory mapper.

At or below 25 MHz (the “clock speed” configuration item is set to “normal”), the emulator will operate out of emulation memory without requiring any wait states for mapper address translation. In this case, there is no need to disable emulation memory.

When the clock speed is above 25 MHz (the “clock speed” configuration item is set to “fast”), mapper address translation requires one wait state when operating out of emulation or target memory.

If the emulator is operating out of target memory only, it can run at clock speeds above 25 MHz without any wait states when emulation memory and the memory mapper are disabled.

off (clock speed = fast)	**High speed, without mapper.** In this mode, no wait states are inserted, but all accesses are directed to the target system. No breaks on memory type (guarded, write to ROM, etc.) are available. Pipelined accesses are supported.
on (clock speed = fast, clocks for emulation memory = 2)	**High speed, with mapper.** In this mode, any new request (assertion of $\overline{\mathrm{IREQ}}$ or $\overline{\mathrm{DREQ}}$ causing a new access) will result in one clock cycle of dead time on that bus to allow for mapper address translation. This means that

the target system must insert at least one wait state during this type of access. Dead time only applies to the first access; subsequent burst cycles may operate with zero wait states.

Pipelined accesses are not allowed (the $\overline{\text{PEN}}$ signal is not driven to the emulation processor) because each access requires a separate cycle. The $\overline{\text{PEN}}$ (Pipeline Enable) signal allows devices that can support pipelined accesses to signal that a second access may begin while the first completes. The target system can drive the pipeline enable signal, but it will not reach the processor. See the Am29000/Am29050 microprocessor data book for more information.

When emulation memory is enabled, the most recently entered map is used.

Clocks for Emulation Memory

This configuration item specifies the number of clock cycles to use when accessing emulation memory. The valid settings for this configuration item are 1 and 2.

This configuration item is useful for slower emulation memory (for example, 256K x 4 RAMs in emulation memory may not be as fast as 64K x 4 RAMs).

Summary of Configuration Items Related to SYSCLK

Configuration items “clock speed”, “emulation memory”, “clocks for emulation memory ” interact (as described above) and must be set depending on SYSCLK, whether or not emulation memory is to be used, and, if emulation memory is being used, the access time of the emulation memory modules installed. (The 2 Mbyte memory modules have a slower access time than the 0.5 Mbyte memory modules.) Figure 5-1 shows the appropriate settings.

Table 5-1. SYSCLK Related Configuration Settings

Emulation memory usage		**Target system clock speed (in MHz)**			**Configuration settings**		
Are you going to use emulation overlay memory?	**Do you have any 2 Mbyte memory modules installed?**	**< = 20**	**> 20 and < = 25**	**> 25 and < = 33**	**emulation memory**	**clock speed**	**clocks for emulation memory**
no	X	X	X	X	off	X	X
yes	X	yes			on	normal	1
yes	no		yes		on	normal	1
yes	yes		yes		on	normal	2
yes	X			yes	on	fast	2

Clock Speed and the Analyzer. The analyzer must be informed of the SYSCLK rate via the “Clock Speed” field of the “Analysis Format” screen.

To configure the analyzer clock select **A**nalysis **F**ormat.

Use the arrow keys to move the cursor to the field next to the label “Clock Speed.” **Tab** to select **slow** if the analyzer data rate is less than or equal to 16.67 MHz. Select **fast** if the analyzer data rate is between 16.67 and 20 MHz. Select **very fast** if the analyzer data rate is between 20 and 25 MHz.

Press < **End**> , then < **Enter**> , to save your changes and exit the format form. Press < **Esc**> if you want to discard your changes and exit the format form.

The emulation analyzer can capture bus cycles at data rates up to 25 MHz. However, the trace state and time counters are limited to lower speeds. The 29000/29050 processor is set to **very fast** by default to ensure correct analyzer operation up to 25 MHz.

The analyzer can capture all types of bus cycles correctly up to the maximum clock rate of 25 MHz, but cannot correctly count states or time at higher speeds for certain bus cycle types.

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

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

To determine the correct setting for the “Clock Speed” field in the 29000/29050 emulator, calculate the maximum data rate by using the above equation. Remember that the emulator always inserts one wait state for all synchronous and burst accesses to emulation memory, and also must insert one wait state for synchronous and burst accesses to target memory when the external clock is greater than or equal to 25 MHz. Then choose the data rate option according to the data rate.

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

Analyzer clock rate	Clock Speed setting	Valid Count Qualifier options
clock ≤ 16.67 MHz	slow	Count < state> Count time
clock ≤ 20 MHz	fast	Count < state>
clock ≤ 25 MHz	very fast	Count none

Suppose that you are running the 29000/29050 processor at 30 MHz. You have enabled a wait state for target memory since target memory requires one wait state for synchronous/burst accesses over 25 MHz. The resulting data rate is 20 MHz, so you modify the “Clock Speed” field in the **A**nalysis **F**ormat form to **fast**. You are limited to counting states in the trace specification.

See the *PC Interface: Analyzer User’s Guide* for more information.

Number of Wait States for Emulation Memory

The “number of wait states for emulation memory” configuration items specify the minimum number of wait states on emulation memory accesses in burst mode and in simple mode.

Burst Mode. Specifies the minimum number of wait states for burst mode accesses (acceptable values are 1 through 4).

Simple Mode. Specifies the minimum number of wait states for simple mode access (acceptable values are 2 through 9).

The “clock speed” configuration items affect the number of clock cycles required for an emulation memory access. Also, an additional clock cycle is required for an access if there is a collision between I-bus and D-bus accesses and priority was given to the other access. See the “primary bus for emulation memory” configuration items.

The number of wait states required by an emulation memory access (clock cycles - 1) is compared to the number specified in the “number of wait states for emulation memory” configuration items. If the access doesn’t take the minimum number of wait states, then additional wait states are inserted.

For example, if “clock speed” is set to “fast” and “clocks for emulation memory” is set to “2”, a first access of emulation memory requires 4 clock cycles (1 for mapper address translation, 2 for the emulation memory access if there was no conflicting access from the other bus, and 1 resynchronization cycle). In other words, the access requires 3 wait states. Now, if “number of wait states for emulation memory: simple mode” is set to “5”, 2 wait states are inserted so that the minimum of 5 is met.

Wait State Summary

The *minimum* number of wait states depends on the settings for the configuration items “emulation memory”, “clock speed”, and “clocks for emulation memory”. Table 5-2 shows the minimum number of wait states generated for valid settings of these configuration items.

Table 5-2. Wait State Summary

Configuration settings			Number of wait states for cycle type				
emulation memory	clock speed	clocks for emulation memory	Target Memory		Emulation Memory		
			Initial	Burst	Initial	Burst	B-rsm *
on	normal	1	0	0	2	1	2
on	normal	2	0	0	3	2	3
on	fast	2	1	0	4	2	3
off	X	X	0	0	N/A	N/A	N/A

Note: If “clock speed” is set to “fast” and the target system attempts to respond with no wait states on an initial cycle (xREQ true) data may be lost.

* B-rsm - This is the number of wait states to resume burst mode to/from emulation memory after it has been suspended by the master (normally the Am29000 or Am29050).

The *actual* number of wait states for *target accesses* will be the greater of:

- The minimum wait states from the table above, or
- The number of wait states inserted by the target.

The *actual* number of wait states for *emulation memory accesses* will be the greater of:

- The minimum wait states from the table above, or
- The number of wait states specified by the “number of wait states for emulation memory” configuration items, or
- The number of wait states inserted by the target if emulation accesses are locked to the target (in other words, if the “lock emulation ready for access type” configuration items are set to “on”).

Note

If a bank of emulation memory is shared by both the I and D busses and a simultaneous access occurs, the minimum number of wait states for the lower priority bus will be increased by the number of clock cycles for an emulation memory access. See the “Primary bus for emulation memory bank A/B?” memory configuration questions and the “Number of clocks for emulation memory accesses?” emulator configuration question for more information.

Real-Time Mode

You may want to restrict emulator execution to real-time to prevent accidental breaks that might cause target system problems.

off	Disables the real-time restriction, and allows the system to accept commands normally.
on	Restricts the emulator to real-time execution.

When you restrict runs to real-time and the emulator is running user code, the system refuses all commands that require access to processor registers or target system memory.

These commands include:

- Register display/modification.
- Memory display/modification commands that access target system memory.
- Memory copy.
- Memory store.
- Memory find.

Note

Because the emulator contains multi-port emulation memory, commands which access emulation memory are allowed while runs are restricted to real-time.

Emulation Memory Configuration

In addition to the memory mapper and emulation memory wait state configuration described previously in the “Emulator Speed Configuration” section, there are options to configure the “primary bus selection” for the banks of emulation memory and the “lock ready” for the bus access selections.

Primary Bus for Emulation Memory

When a block of emulation memory is mapped with both I-bus and D-bus connections, the instruction bus and data bus could access the same memory location at the same time.

This configuration item lets you tell the emulation memory arbiter to give priority to either instruction bus accesses or data bus accesses. (Accesses to emulation memory made by the emulator have a lower priority than either the instruction or data bus.)

bank A = i bank B = i	Instruction bus accesses to emulation memory are given priority over data bus accesses.
bank A = d bank B = d	Data bus accesses to emulation memory are given priority over instruction bus accesses.

Lock Emulation Ready for Access Type

This configuration item lets you lock the emulation memory ready signal to that of the target system, in case other target system circuitry needs to be synchronized with emulation memory accesses.

data = on	The emulation ready signal for data bus accesses is locked with the target system ready.
data = off	Disables locking of the emulation ready signal for data bus accesses with the target system ready.
instruction = on	The emulation ready signal for instruction bus accesses is locked with the target system ready.

instruction = off	Disables locking of the emulation ready signal for instruction bus accesses with the target system ready.

When locked, the emulator will access emulation memory as configured. It will then wait until the target system signals ready before terminating the cycle. Target ready signals that occur before the emulation access is completed will be ignored.

Analysis Mode Configuration

In the HP 64774 emulator, the analyzer may operate in one of three different modes: instruction bus mode, data bus mode, and status bus mode. Also, you can have the analyzer switch modes at the trigger point. For example, you could capture information in data mode prior to the trigger and capture information in status mode after the trigger.

Analysis Mode

This configuration item lets you specify the analysis mode. You can also specify different capture modes before and after the trigger.

ii	Instruction/Data bus mode. All instruction accesses are recorded. Data accesses are recorded if they don't occur on the same clock cycle as instruction bus accesses.
dd	Data/Instruction bus mode. All data accesses are recorded. Instruction accesses are recorded if they don't occur on the same clock cycle as data bus accesses.
ss	Status mode. Status information for all transactions is stored.
id	Instruction/Data bus mode before trigger. Data/Instruction bus mode after trigger.
is	Instruction/Data bus mode before trigger. Status mode after trigger.

di	Data/Instruction bus mode before trigger. Instruction/Data bus mode after trigger.
ds	Data/Instruction bus mode before trigger. Status mode after trigger.
si	Status mode before trigger. Instruction/Data bus mode after trigger.
sd	Status mode before trigger. Data/Instruction bus mode after trigger.

Analysis Switching Signal

This configuration item specifies which one of the internal TRIG1 or TRIG2 signals is used when you have selected an analysis mode that switches at the trigger point.

TRIG1	The internal TRIG1 signal is used.
TRIG2	The internal TRIG2 signal is used.

Once this configuration item is set, you need only to modify the trigger configuration to specify that the selected signal be driven when the trigger occurs.

Emulator Break Configuration

There are a number of different ways that the emulator can break out of user program execution into the monitor. The following emulator configuration items allow you to enable or disable these types of emulator break conditions.

Software Breakpoints

This configuration option allows you to use the breakpoints feature. See the “Getting Started” chapter for information on using breakpoints.

on	The breakpoints feature is enabled by default. When you define or set a software breakpoint, the emulator replaces the opcode at the software breakpoint address with a HALT instruction. When the HALT instruction is executed, emulator execution breaks into the monitor, and the original opcode is restored. A subsequent **P**rocessor **G**o **P**c” or **P**rocessor **S**tep **P**c command will execute from the breakpoint address.
off	The breakpoints feature is disabled. When you add or set a software breakpoint, this feature is automatically enabled and correctly reflected in the configuration display. When choosing this item, any existing breakpoints are cleared.

Break on Write to ROM

Emulator execution can break into the monitor when the target (user) program writes data to a location mapped as ROM.

on	By default, emulator execution will break into the monitor when the target program writes to ROM locations.
off	Target program writes to ROM locations will not cause emulator execution to break into the monitor.

Break on $\overline{\text{IERR}}$ or $\overline{\text{DERR}}$ Signal

This configuration item specifies whether $\overline{\text{IERR}}$ or $\overline{\text{DERR}}$ signals during instruction/data fetches should break emulator execution into the monitor. You can enable a break on these signals as an alternative to the normal Instruction Access Exception or Data Access Exception traps.

on	Breaking into the monitor on $\overline{\text{IERR}}$ or $\overline{\text{DERR}}$ signals is enabled.
off	Disable breaking.

Break on $\overline{\text{WARN}}$ Signal

This configuration item specifies whether $\overline{\text{WARN}}$ signals should break emulator execution into the monitor. The emulator ignores $\overline{\text{WARN}}$ unless this break is enabled.

on	Breaking into the monitor on the $\overline{\text{WARN}}$ signal is enabled.
off	Disable breaking.

General Emulator Configuration

This section describes additional emulator configuration items not mentioned above.

Use Coprocessor

If there is a coprocessor in the target system, this configuration item specifies whether the emulator is allowed to access it for register displays. (The Am29027 coprocessor is supported.)

on	The emulator can access the target system coprocessor.
off	The emulator can’t access the target system coprocessor.

Byte Ordering for Memory and I/O Ports

The byte ordering configuration items specify the type of ordering (big endian or little endian) that is used when displaying memory or I/O port locations. This configuration selection does not affect the Byte Order bit in the processor’s configuration register.

l-> r	Bytes are numbered from left to right within a word (big endian).
r-> l	Bytes are numbered from right to left within a word (little endian).

Force Simple Mode

Normally, the emulator will use burst mode operation for its memory operations. You can force it to stay in simple mode by using the “force simple mode” configuration item. This will inhibit the IBACK and DBACK signals during emulation memory access cycles.

off	Allow burst mode operation.
on	Prevent burst mode operation.

This can be useful to prevent collisions between requests for the I bus and D bus, as they can never happen simultaneously when burst mode is disabled.

Access Width

This configuration item specifies the type of microprocessor cycles that are used by the monitor program to access target memory locations. When a command asks the monitor to read or write target system memory, the monitor program uses the access width setting to decide whether byte, half-word, or word instructions should be used.

byte	Byte accesses are used when the emulator accesses target memory. Target memory must support byte accesses for this option to work.
half	Half-word accesses are used when the emulator accesses target memory. Target memory must support half-word accesses for this option to work.

word	Word accesses are used when the emulator accesses target memory.

Default Address Space

This configuration item assigns the default address space designator. This allows you to refer to memory locations without having to specify the address space designator (for example, if the "default address space" configuration item is set to "i", entering the address "3000" really means "3000@i"). The options for this configuration item and the address spaces assumed when no designator is given are:

i	Instruction space.
d	Data space.
id	Instruction/data space.
r	Instruction ROM space.
a	Instruction ROM on data bus space.
ra	Instruction ROM (both busses) space.

CMB Interaction

Coordinated measurements are synchronous measurements between multiple emulators or analyzers. Coordinated measurements can be made between HP 64700-Series emulators that communicate over the Coordinated Measurement Bus (CMB).

Multiple emulator start/stop is one type of coordinated measurement. The CMB signals READY and /EXECUTE are used to perform multiple emulator start/stop.

This configuration item allows you to enable/disable interaction over the READY and /EXECUTE signals. (The third CMB signal, TRIGGER, is unaffected by this configuration item.)

on — Multiple emulator start/stop is enabled. If the

```
Processor CMB Go ...
```

command is entered, the emulator will start executing code when a pulse on the /EXECUTE line is received. The READY line is driven false while the emulator is running in the monitor. It goes true whenever execution switches to the user program.

off — The emulator ignores the /EXECUTE and READY lines, and the READY line is not driven.

Note

CMB interaction will also be enabled when the

Processor **C**MB **E**xecute

command is entered.

For more information, see the chapter “Using the Emulator — In Depth”.

Storing an Emulator Configuration

The PC Interface lets you store a particular emulator configuration so that it may be re-loaded later. The following information is saved in the emulator configuration.

- Emulator configuration items.
- Memory map.
- Break conditions.
- Trigger configuration.
- Window specifications.

To store the current emulator configuration, select:

Config **S**tore

Enter the name of a file to which the emulator configuration will be saved.

Loading an Emulator Configuration

If you have previously stored an emulator configuration and want to reload it into the emulator, select:

Config **L**oad

Enter the configuration file name and press < **Enter**> . The emulator will be reconfigured with the values specified in the configuration file.

Notes

A

Using the HP 64000 Reader

An HP 64000 “reader” is provided with the PC Interface. The HP 64000 reader converts the files into two files that are usable with your emulator. This means that you can use available language tools to create HP 64000 absolute files, then load those files into the emulator using the PC Interface.

The HP 64000 reader can operate from within the PC Interface or as a separate process. When operating the HP 64000 reader, you may need to execute it as a separate process if there is not enough memory on your computer to operate the PC Interface and HP 64000 reader simultaneously. You can also operate the reader as part of a “make file.”

What the Reader Does

Using the HP 64000 files (< file.X> , < file.L> , < scr1.A> , < scr2.A> , ...) the HP 64000 reader will produce two new files, an “absolute” file and an ASCII symbol file, that will be used by the PC Interface. These new files are named: “< file> .hpa” and “< file> .hps.”

The Absolute File

During execution of the HP 64000 reader, an absolute file (< file> .hpa) is created. This absolute file is a binary memory image that is optimized for efficient downloading into the emulator.

The ASCII Symbol File

The ASCII symbol file (< file> .hps) produced by the HP 64000 reader contains global symbols, module names, local symbols, and, when using applicable development tools such as a “C” compiler, program line numbers. Local symbols evaluate to a fixed (static, not stack relative) address.

Note

You must use the required options for your specific language tools to include symbolic (“debug”) information in the HP 64000 symbol files. The HP 64000 reader will only convert the symbol information present in the HP 64000 symbol files (< file.L> , < src1.A> , < src2.A> , ...).

The symbol file contains symbol and address information in the following form:

```
 module_name1
 module_name2
 ...
 module_nameN
global_symbol1  address@designator
global_symbol2  address@designator
...
global_symbolN  address@designator
|module_name1|# 1234    address@designator
|module_name1|local_symbol1     address@designator
|module_name1|local_symbol2     address@designator
...
|module_name1|local_symbolN     address@designator
```

The space preceding module names is required. A single tab separates symbol and address.

Each of the symbols is sorted alphabetically in the order: module names, global symbols, and local symbols.

Line numbers look like a local symbol except that “local_symbolX” will be replaced by “# NNNNN” where NNNNN is a five digit decimal line number. The addresses associated with global and local symbols are specific to the processor for which the HP 64000 files were generated.

Note

Because the 29000/29050 emulator can store symbols internally, symbols will appear in disassembly. When the line number symbol is displayed in the emulator, it appears in brackets. Therefore, the symbol “MODNAME: line 345” will be displayed as “MODNAME:[345]” in mnemonic memory and trace list displays.

Local symbols are scoped. To access a variable named "COUNT" in a source file module named "MAIN.C", you would enter "MAIN.C:COUNT". Because variables are case-sensitive, you must enter either upper- or lower-case letters, or use a combination of both to match the actual variable stored in the .HPS file. You can also display symbols to examine the variable.

Table A-1. How to Access Variables

Module Name	Variable Name	You Enter:
MAIN.C	COUNT	MAIN.C:COUNT
MAIN.C	line number 23	MAIN.C: line 23

Line number symbols are accessed by entering the following on one line in the order shown:

module name
colon (:)
space
the word "line"
space
the decimal line number

For example:

```
MAIN.C: line 23
```

Location of the HP 64000 Reader Program

The HP 64000 reader is located in the directory named \hp64700\bin by default, along with the PC Interface. This directory must be in the environment variable PATH for the HP 64000 reader and PC Interface to operate properly. This is usually defined in the "\autoexec.bat" file. *The following examples assume that you have "\hp64000\bin" included in your PATH variable. If not, you must supply the directory name when executing the reader program.*

Using the Reader from MS-DOS

The command name for the HP 64000 reader is **RHP64000.EXE**. To execute the reader from the command line, for example, enter:

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

-q — This option specifies the “quiet” mode, and suppresses the display of messages.

-f@fc — For emulators supporting function codes, this allows a function code to be supplied for the load addresses of data in the absolute file. This function code is not applied to symbols. For example, if your emulator supports a function code for program space (1000@p is a legal address), the option to load all absolute code into program space would be -f@p. For the complete list of applicable function codes, see the PC Interface **M**emory **L**oad command. If no function code override is desired, leave this option out of the command line and absolute data will be loaded into the default address space.

< filename> — This represents the name of the HP 64000 linker symbol file (file.L) for the absolute file to be loaded.

The following command will create the files “TESTPROG.HPA” and “TESTPROG.HPS”:

```
RHP64000 TESTPROG.L
```

Using the Reader from the PC Interface

The PC Interface has a file format option under the **M**emory **L**oad command. After you select HP64000 as the file format, the HP 64000 reader will operate on the file you specify. After this completes successfully, the PC Interface will accept the absolute and symbol files produced by the reader.

To use the reader from the PC Interface:

1. Start the PC Interface.

2. Check to make sure that you have mapped memory as appropriate for your system design. See the “Getting Started” chapter for information about mapping memory.

3. Select **M**emory, **L**oad. The memory load menu will appear.

4. The default file format will appear as “HP64000.” This is the file format you will use.

5. Use **Tab** and **Shift-Tab** to select the whether to load emulation memory, target system memory, or both. Press **< Enter>** to accept your choice.

6. Use **Tab** and **Shift-Tab** to select the function code space to be loaded. Press **< Enter>** to accept your choice.

7. Use **Tab** to select **yes** if you want the reader to re-read the absolute file and produce new .HPA and .HPS files. You would want to do this if you had changed any of the load options and needed to re-load the program in order to have the changes take effect. Press **< Enter>** to accept your choice.

8. Specify the name of an HP 64000 linker symbol file (TESTFILE.L for example).

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

Using the HP 64000 file that you specify (TESTFILE.L, for example), the PC Interface does the following:

- It checks to see if two files with the same base name and extensions .HPS and .HPA already exist (for example, TESTFILE.HPS and TESTFILE.HPA).
- If TESTFILE.HPS and TESTFILE.HPA don't exist, the HP 64000 reader produces them. The new absolute file, TESTFILE.HPA, is then loaded into the emulator.
- If TESTFILE.HPS and TESTFILE.HPA already exist but the create dates and times are earlier than the HP 64000 linker symbol file creation date/time, the HP 64000 reader recreates them. The new absolute file, TESTFILE.HPA, is then loaded into the emulator.
- If TESTFILE.HPS and TESTFILE.HPA already exist but the dates and times are later than the creation date and time for the HP 64000 linker symbol file, the HP 64000 reader will not recreate TESTFILE.HPA. The current absolute file, TESTFILE.HPA, is then loaded into the emulator.

Note

Date/time checking is done only within the PC Interface. When running the HP 64000 reader at the MS-DOS command line prompt, the HP 64000 reader will always update the absolute and symbol files.

When the HP 64000 reader operates on a file, a status message will be displayed indicating that it is reading an HP 64000 file. When the HP 64000 reader completes its processing, another message will be displayed indicating the absolute file is being loaded.

The PC Interface executes the reader with the **-q** (quiet) option by default. A field is supplied on the form allowing specification of the -f@fc option.

The memory type and function parameters work with your memory map. Each memory map term has a memory type and function code associated with it. Based on what you enter here as the memory type and function code, the PC Interface selects all memory map terms that match the specified type and function code, and comes up with a set of addresses that are eligible for loading. The PC Interface then reads your absolute file and loads only those addresses that are eligible. Addresses in your absolute file that are not eligible for loading are simply ignored.

If the Reader Won’t Run

If your program is very large, the PC Interface may run out of memory while attempting to create the database file. If this happens, you will need to exit the PC Interface and execute the program at the MS-DOS command prompt.

Including RHP64000 in a Make File

You may want to incorporate the “RHP64000” process as the last step in your “make file”, or as a step in your construction process, so as to eliminate the possibility of having to exit the PC Interface due to space limitations describe above. If the files with “.HPA” and “.HPS” extensions are not current, the process of loading an HP 64000 file will automatically create them.

Notes

B

Using the IEEE-695 Reader

An IEEE-695 MUFOM (Microprocessor Universal Format for Object Modules) “reader” is provided with the PC Interface. The IEEE-695 reader converts an IEEE-695 format file into two files that are usable with the HP 64774 emulator. This means you can use available language tools to create IEEE-695 absolute files, then load those files into the emulator from the PC Interface.

The IEEE-695 reader can operate from within the PC Interface or as a separate process. You may need to execute the reader as a separate process if there is not enough memory on your personal computer to run the PC Interface and the reader simultaneously.

You can also run the reader as part of a “make file.”

What the Reader Does

The IEEE-695 reader accepts an IEEE-695 format absolute file in the form “< file> .< ext> ” and creates two new files that are used by the PC Interface: an “absolute” file, and an ASCII symbol file.

The Absolute File

During execution of the IEEE-695 reader, an absolute file (< file> .HPA) is created. This absolute file is a binary memory image which is optimized for efficient downloading into the emulator.

The ASCII Symbol File

The ASCII symbol file (< file> .HPS) produced by the IEEE-695 reader contains global symbols, module names, local symbols, and, when using applicable development tools like a “C” compiler, program line numbers. Local symbols evaluate to a fixed (static, not stack relative) address.

Note

You must use the required options for your specific language tools to include symbolic ("debug") information in the IEEE-695 absolute file.

The symbol file contains symbol and address information in the following form:

```
 module_name1
 module_name2
 ...
 module_nameN
global_symbol1  address@designator
global_symbol2  address@designator
...
global_symbolN  address@designator
|module_name|local_symbol1       address@designator
|module_name|local_symbol2       address@designator
...
|module_name|local_symbolN      address@designator
|module_name|# 1234     address@designator
```

The space preceding module names is required. A single tab separates symbol and address.

Each of the symbols is sorted alphabetically in the order: module names, global symbols, and local symbols.

The local symbols are scoped. This means that to access a variable named "count" in a function named "foo" in a source file module named "main.c", you would enter "main.c:foo.count". See table B-1.

Table B-1. The Scope of Symbol Names

Module Name	Function Name	Variable Name	You Enter
main.c	foo	count	main.c:foo.count
main.c	bar	count	main.c:bar.count

Line numbers look like a local symbol except that "local_symbolX" is replaced by "# NNNNN" where NNNNN is a five digit decimal line number. Line numbers should appear in ascending order.

Note

When the line number symbol is displayed in the emulator, it appears as a bracketed number. Therefore, the symbol “modname: line 345” will be displayed as “modname:[345]” in mnemonic memory and trace list displays.

Location of the IEEE-695 Reader Program

The IEEE-695 reader is located in the directory named **\hp64700\bin** by default, along with the PC Interface. This directory must be in the environment variable PATH for the IEEE-695 reader and PC Interface to operate properly. This is usually defined in the “\autoexec.bat” file.

Using the IEEE-695 Reader from MS-DOS

The command name for the IEEE-695 reader is **RIEEE695.EXE**. You can execute the IEEE-695 reader from the command line with the following command syntax:

```
C:\HP64700\BIN\RIEEE695 [-u] [-q] [-f@fc]
<filename> <RETURN>
```

[-u] Specifies that the first leading underscore of a symbol is not removed.

[-q] Specifies the “quiet” mode. This option suppresses the display of messages.

[-f@fc] Where “fc” specifies the address space designator for the absolute and symbol files (“i”, “d”, “id”, “r”, “a”, or “ra”). Only one of the options can be selected. The default is no address space designator.

< filename> Specifies the name of the file containing the IEEE-695 absolute program.

Using the IEEE-695 Reader from the PC Interface

The 29000/29050 PC Interface has a file format option under the "**M**emory, **L**oad" command. After you select this option, the IEEE-695 reader will operate on the file you specify. After the reader completes successfully, the 29000/29050 PC Interface will load the absolute and symbol files produced by the Reader.

To use the reader from the PC Interface:

1. Start the PC Interface.
2. Check to make sure that you have mapped memory as appropriate for your system design. See the "Getting Started" chapter for information about mapping memory.
3. Select **M**emory, **L**oad. The memory load menu will appear.
4. Use **Tab** and **Shift-Tab** to select "IEEE-695."
5. Use **Tab** and **Shift-Tab** to select the whether to load emulation memory, target system memory, or both. Press **< Enter>** to accept your choice.
6. Use **Tab** and **Shift-Tab** to select the function code space to be loaded. Press **< Enter>** to accept your choice.
7. Use **Tab** to select **yes** if you want the reader to re-read the absolute file and produce new .HPA and .HPS files. You would want to do this if you had changed any of the load options and needed to re-load the program in order to have the changes take effect (for example, deleting leading underscore characters). Press **< Enter>** to accept your choice.

8. Use **Tab** and **Shift-Tab** to select whether to delete a leading underscore character from the symbol name.

9. Use **Tab** and **Shift-Tab** to specify the name of an IEEE-695 linker symbol file (TESTFILE.ABS for example).

Note

The file extension can be something other than ".ABS", but cannot be ".HPA", ".HPT", or ".HPS". The "< filename> .HPT" file is a temporary file used by the IEEE-695 reader to process the symbols.

---

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

    Using the IEEE-695 file that you specify (for example, TESTFILE.ABS), the PC Interface performs the following:

- Checks to see if two files with the same base name and extensions .HPS and .HPA already exist (for example, TESTFILE.HPS and TESTFILE.HPA).

- If TESTFILE.HPS and TESTFILE.HPA don't exist, the IEEE-695 reader produces them. The new absolute file, TESTFILE.HPA, is then loaded into the emulator.

- If TESTFILE.HPS and TESTFILE.HPA already exist but the create dates and times are earlier than the IEEE-695 file creation date/time, the IEEE-695 reader re-creates them. The new absolute file, TESTFILE.HPA, is then loaded into the emulator.

- If TESTFILE.HPS and TESTFILE.HPA already exist but the dates and times are later than the creation date/time for the IEEE-695 file, the current absolute file, TESTFILE.HPA, is then loaded into the emulator.

Note

Date/time checking is done only within the PC Interface. When running the IEEE-695 reader at the MS-DOS command line prompt, the reader will always update the absolute and symbol files.

When the IEEE-695 reader operates on a file, a status message will be displayed indicating that it is reading an IEEE-695 file. When the reader completes its processing, another message will be displayed indicating the absolute file is being loaded.

The memory type and function parameters work with your memory map. Each memory map term has a memory type and function code associated with it. Based on what you enter here as the memory type and function code, the PC Interface selects all memory map terms that match the specified type and function code, and comes up with a set of addresses that are eligible for loading. The PC Interface then reads your absolute file and loads only those addresses that are eligible. Addresses in your absolute file that are not eligible for loading are simply ignored.

If the IEEE-695 Reader Won't Run

If your program is very large, then the PC Interface may run out of memory while attempting to create the database file. If this happens, you will need to exit the PC Interface and execute the program at the command prompt.

Including RIEEE695 in a Make File

You may want to incorporate the "RIEEE695" process as the last step in your "make" file, or as a step in your construction process, to eliminate the possibility of having to exit the PC Interface due to space limitations. If the "-.HPA" and "-.HPS" files are not current, the process of loading an IEEE-695 file will automatically create them.

Index