HP 64758

70632 Emulator PC Interface

User's Guide

HEWLETT PACKARD

HP Part No. 64758-97005
Printed in U.S.A.
March, 1993

Edition 3

Notice

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

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

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

Printing History

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

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

Edition 1	64758-97001, August 1990
Editon 2	64759-97003, November 1990
Edition 3	**64758-97005, April 1993**

Using This manual

This manual introduces you to the HP 64758G/H 70632 Emulator as used with the PC Interface.

This manual:

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

This manual does not:

- Show you how to use every PC Interface command and option. See the *HP 64700 Emulators PC Interface: User's Reference* for further details.

Organization

Chapter 1 **Introduction.** This chapter lists the 70632 emulator features and describes how they can help you in developing new hardware and software.

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

Chapter 3 **Virtual Mode Emulation Topics.** This chapter shows you how to use emulator in virtual mode. The chapter describes a sample program and how to: load programs into the emulator, display on-chip MMU registers, privilege registers and TCB, set software breakpoints, and use the analyzer in virtual mode.

Chapter 4 **Configuring the Emulator.** You can configure the emulator to adapt it to your specific development needs. This chapter describes the options available when configuring the emulator, and how to save and restore particular configurations.

Chapter 5 **Using the Emulator.** This chapter describes emulation topics that are not covered in the "Getting Started" and "Virtual Mode Emulation Topics" chapters (for example, coordinated measurements and storing memory).

Chapter 6 **In-Circuit Emulation.** This chapter shows you how to plug the emulator into a target system, and how to use the "in-circuit" emulation features.

Appendix A **File Format Readers.** This appendix describes how to use the Readers from MS-DOS or PC Interface, load absolute files into the emulator, use global and local symbols with the PC Interface.

Contents

Illustrations

Tables

1

Introduction to the 70632 Emulator

Introduction

The topics in the chapter include:

- Purpose of the emulator
- Features of the emulator

Purpose of the 70632 Emulator

The 70632 emulator is designed to replace the NEC uPD70632 microprocessor in your target system to help you integrate target system software and hardware. The 70632 emulator performs just like the NEC uPD70632 microprocessor, but at the same time, it gives you information about the operation of the processor. The emulator gives you control over target system execution and allows you to view or modify the contents of processor registers and, target system memory.

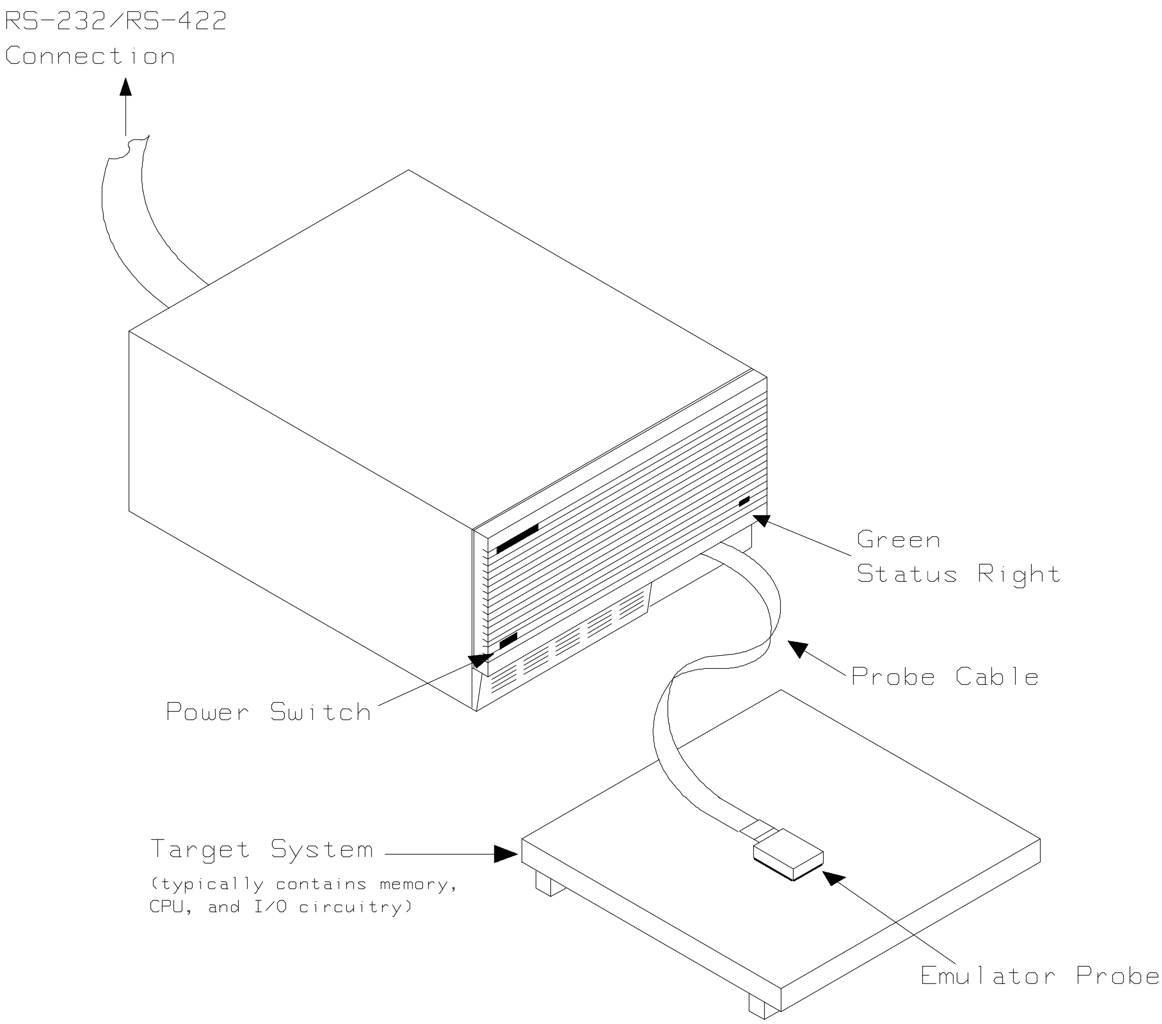


Figure 1-1. HP 64758 Emulator for the 70632

Features of the 70632 Emulator

Supported Microprocessor

The emulator probe has a 132-pin PGA connector. The HP 64758G/H emulator supports the NEC uPD70632 microprocessor.

Clock Speeds

Measurements can be made using the emulator's internal 20 MHz clock or an external clock from 8 MHz to 20 MHz with no wait states added to target memory.

Emulation Memory

Depending on the emulator model number, there are 512K/1M bytes of emulation memory. Memory mapping configuration maps physical memory only. If the MMU is enabled, the user is responsible for knowing user physical memory usage.

Dual-ported memory allows you to display or modify physical emulation memory without stopping the processor. Flexible memory mapping lets you define address ranges over the entire 4 Gbyte address range of the 70632. You can define up to 8 memory ranges (at 4 Kbyte boundaries and at least 4Kbytes in length). The monitor occupies 4K bytes leaving 508K or 1020K bytes of emulation memory which you may use. You can characterize memory ranges as emulation RAM, emulation ROM, target system RAM, target system ROM, or as guarded memory. The emulator generates an error message when accesses are made to guarded memory locations; additionally, you can configure the emulator so that writes to memory defined as ROM cause emulator execution to break out of target program execution. You can select whether the memory accesses honor /READY and /BERR signals from target system for each emulation memory range.

Analysis

The integrated emulation bus analyzer provides real-time analysis of all bus-cycle activity. You can define break conditions based on address and data bus cycle activity. In addition to hardware break, software breakpoints can be used for execution breakpoints.

The 70632 microprocessor has on-chip MMU which provides a 4 Giga-byte virtual space for each task. When you use the on-chip MMU, you will want to analyze either actual or virtual address space. You can configure which address space should be recognized by the emulation analyzer. Analysis functions include trigger, storage, count, and context directives. The analyzer can capture up to 1024 events, including all address, data, and status lines.

FPU

The emulation bus analyzer can capture bus states accessing to a Floating Point Processor.

MMU

The emulator will support development when using the internal Memory Management Unit.

FRM

The emulator supports the master mode of the 70632 FRM function. In the master mode, you can use the analyzer feature of the emulator. If signal is asserted by your target system, the emulator bus signals are held. So the emulator does not work as checker.

Registers

You can display or modify the 70632 internal CPU register contents. This includes the ability to modify the program counter (PC) value so you can control where the emulator starts a program run. You can also display or modify the 70632 MMU register contents.

Single-Step

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

Breakpoints

You can set the emulator/analyzer interaction so the emulator will break to the monitor program when the analyzer finds a specific state or states, allowing you to perform post-mortem analysis of the program execution. You can also set software breakpoints in your program. With the 70632 emulator, setting a software breakpoint inserts a 70632 BRK instruction into your program at the desired location.

Reset Support

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

Software Debugging

The HP 64758G/H Real-Time Emulator for 70632 microprocessors is a powerful tool for both software and hardware designers. Using the HP 64758G/H Emulator's emulation memory (up to 512 Kilo/1 Mega bytes), software debugging can be done without functional target system memory.

Configurable Target System Interface

You can configure the emulator so that it honors target system wait and retry requests when accessing emulation memory. Additionally, the processor signals /READY, /BERR, BFREZ, RT/EP, /NMI, INT, and /HLDRQ may be enabled or disabled independently of the 70632 processor.

Real-Time Operation

Real-time signifies continuous execution of your program at full rated processor speed without interference from the emulator. (Such interference occurs when the emulator needs to break to the monitor to perform an action you requested, such as displaying target system memory.) Emulator features performed in real time include: running and analyzer tracing. Emulator features not performed in real time include: display or modify of target system memory; load/dump of target memory, and display or modification of registers and some virtual related functionality.

Foreground or Background Emulation Monitor

The emulation monitor is a program executed by the emulation processor. It allows the emulation controller to access target system resources. For example, when you display target system memory, the monitor program executes 70632 instructions to read the target memory locations and send their contents to the emulation controller.

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

The monitor program also can execute in *background*, the emulator mode in which foreground operation is suspended so the emulation processor can access target system resources. The background monitor does not occupy processor address space.

Out-of-Circuit or In-Circuit Emulation

The 70632 emulator can be used for both out-of-circuit emulation and in-circuit emulation. The emulation can be used in multiple emulation systems using other HP 64700 Series emulators/analyzers.

2

Getting Started

Introduction

This chapter will lead you through a basic, step by step tutorial that shows how to use the HP 64758G/H 70632 emulaotor with the PC Interface.

This chapter will:

- Tell you what must be done before you can use the emulator as shown in the tutorial examples.
- Describe the sample program used for this chapter's examples.
- Briefly describe how PC Interface commands are entered and how emulator status is displayed.

This chapter will show you how to:

- Start up 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.

Before You Begin

Prerequisites

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

1. Connected the emulator to your computer. The *HP 64700 Series Installation/Service* manual shows you how to do this.

2. Installed 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: User's Reference* manual contains additional information on the installation and setup of the PC Interface.

3. In addition, it is recommended, although not required, that you read and understand the concepts of emulation presented in the *Concepts of Emulation and Analysis* manual. The *Installation /Service* also covers HP 64700 Series system architecture. A brief understanding of these concepts may help avoid questions later.

   You should read the *HP 64700 Emulators PC Interface: User's Reference* manual to learn how to use the PC Interface in general. For the most part, this manual contains information specific to the 70632 emulator.

A Look at the Sample Program

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

```
                .file   "cmd_rds.s"

                .equ    Dest_Size,0x30
                .equ    Stack_Size,0x100

                  .globl  Command_Input, Init, Message_Dest
                .data   "sbt" (RW) >0x00000000
                .org    .+0x34
                .word   Dummy_Text

                .text (RX) >0x00010000
                .align  4

Init:           mov.w   #Stack+Stack_Size,sp
                movea.w Command_Input,r0
                movea.w Message_Dest,r1
                mov.b   #’ ’,r26

Clear:          mov.b   #0x00,[r0]

Read_Input:     mov.b   [r0],r2
                cmp.b   #0x00,r2
                je      Read_Input

Process_Comm:   cmp.b   #’A’,r2
                je      Command_A
                cmp.b   #’B’,r2
                je      Command_B
                jr      Unrecognized

Command_A:      movea.w Message_A,r3
                mov.w   #Message_B-Message_A,r4
                jr      Output

Command_B:      movea.w Message_B,r3
                mov.w   #Invalid_Input-Message_B,r4
                jr      Output

Unrecognized:   movea.w Invalid_Input,r3
                mov.w   #Message_End-Invalid_Input,r4

Output:         movcfu.b [r3],r4,Message_Dest,#Dest_Size
Text_End:       jr      Clear

Dummy_Text:             halt


                .data (R) >0x00020000

Message_A:      .str    "THIS IS MESSAGE A"
Message_B:      .str    "THIS IS MESSAGE B"
Invalid_Input:  .str    "INVALID COMMAND"
Message_End:


                .bss (RW) >0x00030000
                .lcomm  Command_Input,  1,1
                .lcomm  Message_Dest,Dest_Size,4
                .lcomm  Stack,Stack_Size,4
```

Figure 2-1. Sample Program Source

System Base Table

The "sbt" section defines 70632 System Base Table containing the vectors for 70632 interrupts and exceptions. The sample program defines BRK instruction vector pointing to an address in the "text" section. This is requirement for emulation software breakpoints feature. Refer to "Using Software Breakpoints" section in this chapter for details.

Data Declarations

The "data" section defines the messages used by the program to respond to various command inputs. These messages are labeled **Message_A**, **Message_B**, and **Message_I**.

The Destination Area

The "bss" section declares memory storage for the command input byte (**Command_Input**), the destination area (**Message_Dest**), and the stack area.

Initialization

The program instructions from the **Init** label to the **Clear** label perform initialization. The stack pointer is set up and the addresses labeled **Command_Input** and **Message_Dest** are loaded into registers; R0 and R1.

Register R26 is set up to 20H for filling remaining locations after transferring a message to the destination area (**Message_Dest**) with blank.

Reading Input

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

Processing Commands

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

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

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

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

The instructions at **Command_A**, **Command_B**, and **Unrecognized** each load register R3 with the starting location of the appropriate message and register R4 with the length of the message to be displayed. Then, execution transfers to **Output** which writes the appropriate message to the destination location, **Message_Dest**. At the same time, the remaining locations are filled with blanks; the content of register R26.

Then, the program jumps back to read the next command.

Assembling and Linking the Sample Program

The sample program is written for the *HP 64879 70632 Assembler/Linker* hosted on HP-UX. You can use other software development tools to generate absolute files. When using these assembler/linker, a few changes must be made to the sample program. The PC Interface can load one of the following formats:

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

Following commands were used to generate the absolute file with *HP 64879 70632 Assembler/Linker*. The assembler and linker are hosted on HP-UX.

```
$as70616 -a cmd_rds.s >cmd_rds.lis<RETURN>
$ld70616 -o cmd_rds.x -m cmd_rds.o > cmd_rds.map<RETURN>
```

Starting Up the 70632 PC Interface

If you have set up the emulator device table and the HP64700 shell environment variable as shown in the *HP 64700 Emulators PC Interface: User's Reference* (this is done automatically when you use the **install** program to load the PC Interface software on your computer), you can start up the 70632 PC Interface by entering the following command from the MS-DOS prompt:

```
C> pcv70 <emulname>
```

where <emulname> is **emul_com1** if your emulator is connected to the COM1 port or **emul_com2** if it is connected to the COM2 port. If you edited the \hp64700\tables\64700tab file to change the emulator name, substitute the appropriate name for <emulname> in the above command.

In the command above, **pcv70** is the command to start the 70632 PC Interface; "**<emulname>**" is the logical emulator name given in the emulator device table. If this command is successful, you will see the display shown in figure 2-2. If this command is not successful, you will be given an error message and returned to the MS-DOS prompt. Error messages are described in the *PC Interface: User's Reference* manual.

Selecting PC Interface Commands

This manual tells you to "select" commands. You can select commands or command options by using the left and right arrow keys to highlight the option. Then press the **Enter** key. Or, you can simply type the first letter of that option. If you select the wrong option, press the **ESC** key to retrace the command tree.

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

Emulator Status

The emulator status is shown on the line above the command options. The PC Interface periodically checks the status of the emulator and updates the status line.

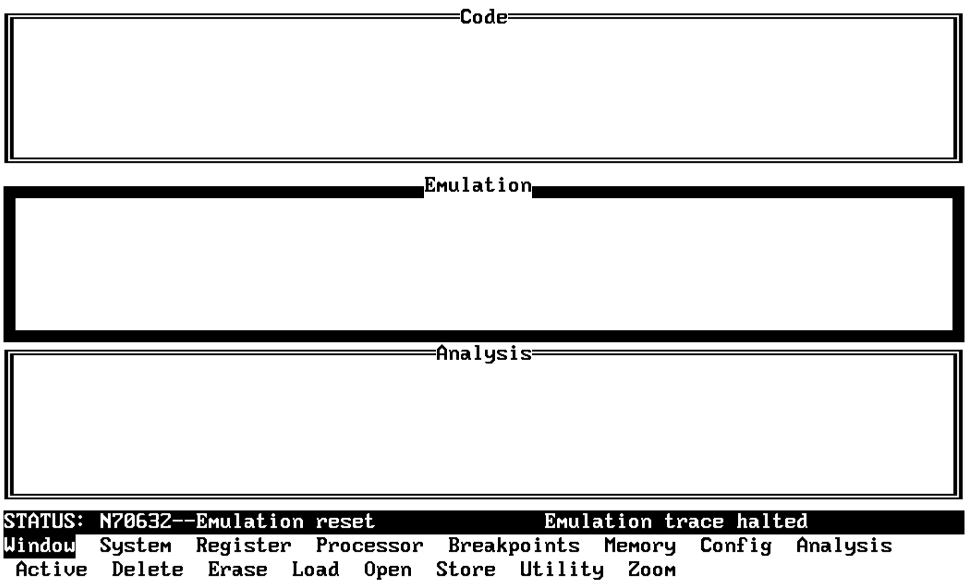


Figure 2-2. PC Interface Display

Mapping Memory

Depending on the emulator model number, user mappable emulation memory consists of 508 or 1020 kilobytes, mappable in 4 Kbyte blocks. The emulation memory system does not introduce any wait states.

The memory mapper allows you to characterize memory locations. It allows you specify whether a certain range of memory is present in the target system or whether you will be using emulation memory for that address range. You can also specify whether the target system memory is ROM or RAM, and you can specify that emulation memory be treated as ROM or RAM. If you are using the emulator in in-circuit, additionally; you can choose whether the emulation accesses honor /READY or /BERR signals from the target system (wait or retry cycles are inserted if requested).

Note

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

Blocks of memory can also be characterized as guarded memory. Guarded memory accesses will generate "break to monitor" requests. Writes to ROM will generate "break to monitor" requests if the "Break on ROM write" configuration item is enabled (see the "Configuring the Emulator" chapter).The memory mapper allows you to define up to 8 different map terms.

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 locations your program will occupy in memory. For example, the HP 64879 linker load map listing for the sample program is shown in figure 2-3.

```
                LINK EDITOR MEMORY MAP

output          input           virtual
section         section         address         size

sbt                             00000000        00000038
                sbt             00000000        00000038 cmd_rds.o

*avail*                         00000038        0000ffc8

.text                           00010000        00000070
                .text           00010000        00000070 cmd_rds.o

*avail*                         00010070        0000ff90

.data                           00020000        00000034
                .data           00020000        00000034 cmd_rds.o

*avail*                         00020034        0000ffcc

.bss                            00030000        00000134        uninitialized
                .bss            00030000        00000134 cmd_rds.o

*avail*                         00030134        fffcfecb
```

Figure 2-3. Load Map Listing for the Sample Program

From the load map listing, you can see that the sample program occupies locations in three address ranges. The system base table area, which contains the breakpoint instruction trap vector, occupies locations 0H through 0fffH. The program area, which contains the opcodes and operands which make up the sample program, occupies locations 10000H through 1006fH. The data area, which contains the ASCII values of the messages the program displays, is occupies locations 20000H through 20033H. The destination area, which contains the command input byte and the locations of the message destination and the stack, occupies locations 30000H through 30133H.

Four mapper terms will be 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
```

By default, unmapped area attribute is defined as target RAM. However, when emulation without plugging the emulator into your target system, unmapped area should be defined as "guarded" to detect the illegal accesses to the area.

As the cursor is in the "Unmapped memory type" field now, press the **TAB** key to select the **grd** (guarded memory) type.

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

```
0..0fff
```

Move the cursor to the "memory type" field of term 1, and press the **TAB** key to select the **eram** (emulation RAM) type.

Move the cursor to the "address range" field of term 2 and enter:

```
10000..10fff
```

Move the cursor to the "memory type" field of term 2, and press the **TAB** key to select the **erom** (emulation ROM) type.

Move the cursor to the "address range" field of term 3 and enter:

```
20000..20fff
```

Move the cursor to the "memory type" field of term 3, and press the **TAB** key to select the **erom** (emulation ROM) type.

Move the cursor to the "address range" field of term 4 and enter:

`30000..30fff`

Move the cursor to the "memory type" field of term 4, and press the **TAB** key to select the **eram** (emulation RAM) type.

To save your memory map, use the right arrow key or 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.) The memory configuration display is shown as follows.

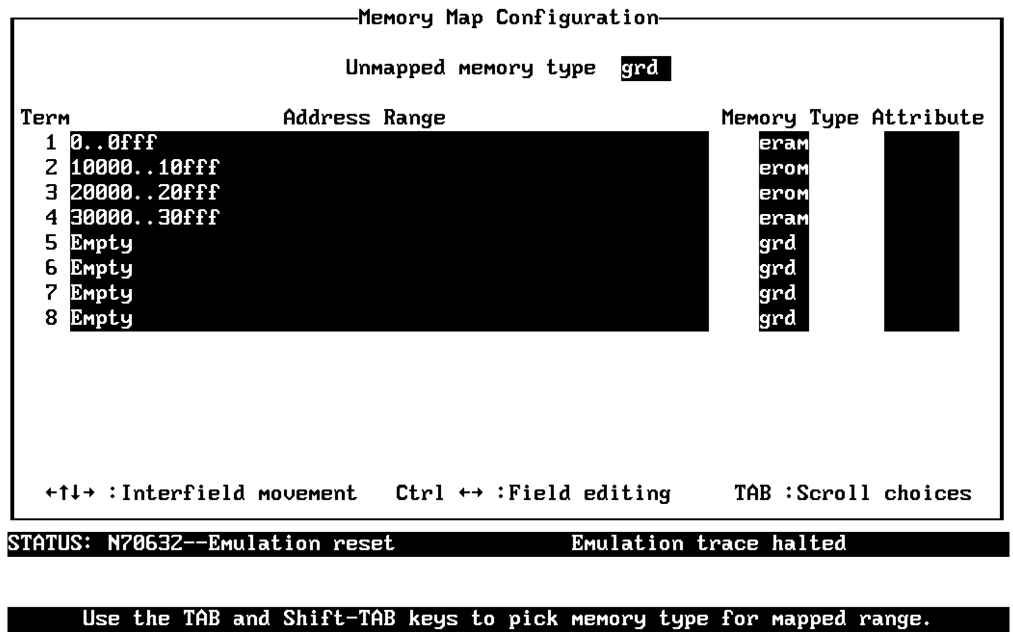


Figure 2-4. Memory Map Configuration

When mapping memory for your target system programs, you may wish to characterize 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.

Loading Programs into Memory

If you have already assembled and linked the sample program, you can load the absolute file by selecting:

```
Memory, Load
```

File Format

Use **Tab** and **Shift-Tab** to select the format of your absolute file. The emulator accepts absolute files in the following formats:

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

For this tutorial, choose the **NEC_COFF** format.

Memory Type

The second field allows you to selectively load the portions of the absolute file which reside in emulation memory, target system memory, or both.

Since emulation memory is mapped to for sample program locations, you can select either "**Emulation**" or "**Both**".

Force Absolute File Read

This option is only available for the HP64000 and NEC COFF formats. It forces the file format readers to regenerate the emulator absolute file (.hpa) and symbol database (.hps) before loading the code. Normally, these files are only regenerated whenever the file you specify (the output of your language tools) is newer than the emulator absolute file and symbol database.

For more information, refer to the File Format Readers appendix.

File Format Options

Some of the formats, such as the NEC COFF format, have special options. Refer to the File Format Readers appendix of this manual for more information. For this example, you do not need to change this configuration.

Absolute File Name

Enter the name of your absolute file ("**cmd_rds.x**" in this example) in the last field, and press **Enter** to start the memory download.

Displaying Symbols

The following pages show you how to display global and local symbols for the sample program. For more information on symbol display, refer to the *PC Interface Reference*.

Displaying Global Symbols

When you load HP64000 or NEC COFF format absolute files into the emulator, the corresponding symbol database is also loaded.

The symbol database also can be loaded with the "**S**ystem, **S**ymbols, **G**lobal, **L**oad" command. Use this command when you load multiple absolute files into the emulator. You can load the various symbol databases corresponding to each absolute file. When you load a symbol database, information from a previous symbol database is lost. That is, only one symbol database can be present at a time.

After a symbol database is loaded, both global and local symbols can be used when entering expressions. You enter global symbols as they appear in the source file or in the global symbols display.

To display global symbols, select:

```
System, Symbols, Global, Display
```

The symbols window automatically becomes the active window because of this command. You can press <CTRL>**z** to zoom the window. The resulting display follows.

```
                                    Symbols
 Modules
---------
 cmd_rds

Address      Symbol
----------   ------
000030000    Command_Input
000010000    Init
000030004    Message_Dest
000020034    edata
000030134    end
000000038    esbt
000010070    etext

STATUS: N70632--Emulation reset              Emulation trace halted
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Command_file  Wait  MS-DOS  Log  Terminal  Symbols  Exit
```

The global symbols display has two parts. The first part lists all the modules that were linked to produce this object file. These module names are used by you when you want to refer to a local symbol, and are case-sensitive. The second part of the display lists all global symbols in this module. These names can be used in measurement specifications, and are case-sensitive. For example, if you wish to make a measurement using the symbol **Cmd_Input**, you must specify **Cmd_Input**. The strings **cmd_input** and **CMD_INPUT** are not valid symbol names here.

Loading and Displaying Local Symbols

To display local symbols, select:

`System Symbols Local Display`

Enter the name of the module you want to display (from the first part of the global symbols list; in this case, **cmd_rds**) and press **Enter**. The resulting display follows.

```
                                  Symbols
Address      Symbol
----------   ------
000010019    Clear
000010033    Command_A
000010043    Command_B
00001006D    Dummy_Text
000020022    Invalid_Input
000020000    Message_A
000020011    Message_B
000020031    Message_End
000010061    Output
000010025    Process_Comm
00001001D    Read_Input
000030034    Stack
00001006B    Text_End
000010053    Unrecognized

STATUS: N70632--Emulation reset              Emulation trace halted
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Command_file  Wait  MS-DOS  Log  Terminal  Symbols  Exit
```

After you display local symbols with the “**S**ystem **S**ymbols **L**ocal **D**isplay” command, you can enter local symbols as they appear in the source file or local symbol display. When you display local symbols for a given module, that module becomes the default local symbol module.

If you have not displayed local symbols, you can still enter a local symbol by including the name of the module:

```
module_name:symbol
```

Remember that the only valid module names are those listed in the first part of the global symbols display, and are case-sensitive for compatibility with other systems (such as HP-UX).

When you include the name of an source file with a local symbol, that module becomes the default local symbol module, as with the “**S**ystem **S**ymbols **L**ocal **D**isplay” command.

Local symbols must be from assembly modules that form the absolute whose symbol database is currently loaded. Otherwise, no symbols will be found (even if the named assembler symbol file exists and contains information).

One thing to note: It is possible for a symbol to be local in one module and global in another, which may result in some confusion. For example, suppose symbol “XYZ” is a global in module A and a local in module B and that these modules link to form the absolute file. After you load the absolute file (and the corresponding symbol database), entering “XYZ” in an expression refers to the symbol from module A. Then, if you display local symbols from module B, entering “XYZ” in an expression refers to the symbol from module B, **not the global symbol**. Now, if you again want to enter “XYZ” to refer to the global symbol from module A, you must display the local symbols from module A (since the global symbol is also local to that module). Loading local symbols from a third module, if it was linked with modules A and B and did not contain an “XYZ” local symbol, would also cause “XYZ” to refer to the global symbol from module A.

Transfer Symbols to the Emulator

You can use the emulator’s symbol-handling capability to improve measurement displays. You do this by transferring the symbol database information to the emulator. To transfer the global symbol information to the emulator, use the command:

```
System Symbols Global Transfer
```

Transfer the local symbol information for all modules by entering:

```
System Symbols Local Transfer All
```

You can find more information on emulator symbol handling commands in the *Emulator PC Interface Reference*.

Displaying Memory in Mnemonic Format

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

`Memory, Display, Mnemonic`

Enter the address range "**10000..1006B**". (You could also specify this address range using symbols, for example, "**Init..cmd_rds:Text_End**".) The Emulation window automatically becomes the active window as a result of this command. You can press <CTRL>**z** to zoom the window. Use the **Home** key to view the memory contents from the top of the display. The resulting display follows.

```
                                  Emulation
Address       Symbol             Mnemonic
-----------   ----------------   ------------------------------------------------
000010000@r   Init               MOV.W       #00030134H,SP
000010007@r   -                  MOVEA.W     Command_Input,R0
00001000e@r   -                  MOVEA.W     Message_Dest,R1
000010015@r   -                  MOV.B       #20H,R26
000010019@r   cmd_rds:Clear      MOV.B       #0H,[R0]
00001001d@r   _rds:Read_Input    MOV.B       [R0],R2
000010020@r   -                  CMP.B       #0H,R2
000010023@r   -                  BE/Z        cmd_rds:Read_Input
000010025@r   ds:Process_Comm    CMP.B       #41H,R2
000010029@r   -                  BE/Z        cmd_rds:Command_A
00001002b@r   -                  CMP.B       #42H,R2
00001002f@r   -                  BE/Z        cmd_rds:Command_B
000010031@r   -                  BR          cmd_rds:Unrecognized
000010033@r   d_rds:Command_A    MOVEA.W     cmd_rds:Message_A,R3
00001003a@r   -                  MOV.W       #00000011H,R4
000010041@r   -                  BR          cmd_rds:Output
000010043@r   d_rds:Command_B    MOVEA.W     cmd_rds:Message_B,R3

STATUS: N70632--Emulation reset              Emulation trace halted
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Display  Modify  Load  Store  Copy  Find  Report
```

Stepping Through the Program

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

`Processor, Step, Address`

Enter a step count of 1, enter the symbol **Init** (defined as a global in the source file), and press **Enter** to step from program's first address, 10000H. The Emulation window remains active. Press <CTRL>**z** to view a full screen of information. The executed instruction, the program counter address, and the resulting register contents are displayed as shown in the following.

```
                                   Emulation
00001004a@r  -                 MOV.W      #00000011H,R4
000010051@r  -                 BR         cmd_rds:Output
000010053@r  ds:Unrecognized   MOVEA.W    cmd_rds:Invalid_Inpu,R3
00001005a@r  -                 MOV.W      #0000000fH,R4
000010061@r  cmd_rds:Output    MOVCFU.B   [R3],R4,Message_Dest,#30H
00001006b@r  md_rds:Text_End   BR         cmd_rds:Clear

000010000@r  Init             MOV.W      #00030134H,SP
PC = 000010007@r
  pc= 00010007  sp= 00030134  fp= 00000000  ap= 00000000 psw=10000000
  r0= 00000000  r1= 00000000  r2= 00000000  r3= 00000000  r4=00000000
  r5= 00000000  r6= 00000000  r7= 00000000  r8= 00000000  r9=00000000
 r10= 00000000 r11= 00000000 r12= 00000000 r13= 00000000 r14=00000000
 r15= 00000000 r16= 00000000 r17= 00000000 r18= 00000000 r19=00000000
 r20= 00000000 r21= 00000000 r22= 00000000 r23= 00000000 r24=00000000
 r25= 00000000 r26= 00000000 r27= 00000000 r28= 00000000 r29=00000000
 r30= 00000000 r31= 00030134

STATUS: N70632--Running in monitor           Emulation trace halted
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Go  Break  Reset  I/O  CMB  Step
```

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, **S**tep, **P**c

After selecting the command above, you have to opportunity to change the previous step count. If you wish to step the same number of times, you can press **Enter** to start the step.

To save time when single-stepping, you can use the function key macro <F1>, which executes the command:

Processor **S**tep **P**c **1**

For more information, see the *Emulator PC Interface Reference* chapter on Function Key Macros.

To repeat the previous command, you can press <CTRL>**r**.

Specifying a Step Count

If you wish to continue to step a number of times from the current program counter, select:

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

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 times to step. Type **5** into the field, and press Enter. The resulting display follows.

```
                                   Emulation
 r30= 00000000 r31= 00030134


000010015@r  -                 MOV.B      #20H,R26
000010019@r  cmd_rds:Clear     MOV.B      #0H,[R0]
00001001d@r  _rds:Read_Input   MOV.B      [R0],R2
000010020@r  -                 CMP.B      #0H,R2
000010023@r  -                 BE/Z       cmd_rds:Read_Input
PC = 00001001d@r
  pc= 0001001d  sp= 00030134  fp= 00000000  ap= 00000000 psw=10000001
  r0= 00030000  r1= 00030004  r2= 00000000  r3= 00000000  r4=00000000
  r5= 00000000  r6= 00000000  r7= 00000000  r8= 00000000  r9=00000000
 r10= 00000000 r11= 00000000 r12= 00000000 r13= 00000000 r14=00000000
 r15= 00000000 r16= 00000000 r17= 00000000 r18= 00000000 r19=00000000
 r20= 00000000 r21= 00000000 r22= 00000000 r23= 00000000 r24=00000000
 r25= 00000000 r26= 00000020 r27= 00000000 r28= 00000000 r29=00000000
 r30= 00000000 r31= 00030134


STATUS: N70632--Running in monitor           Emulation trace halted
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Go  Break  Reset  I/O  CMB  Step
```

Modifying Memory

The preceding step commands show the sample program is executing in the **Read_Input** loop, where it continually reads the command input byte to check if a command has been entered. To simulate the entry of a sample program command, you can modify the command input byte by selecting:

`Memory, Modify, Bytes`

Now enter the address of the memory location to be modified, an equal sign, and new value of that location, for example, "**Command_Input=41**". (The **Command_Input** label was defined as a global symbol in the source file.)

To verify that 41H was indeed written to **Command_Input** (30000H), select:

`Memory, Display, Bytes`

Type the address 30000 or the symbol **Command_Input**, and press **Enter**. This command will automatically activate the Emulation window. The resulting display is shown below.

```
                                 Emulation
000010019@r  cmd_rds:Clear    MOV.B      #0H,[R0]
00001001d@r  _rds:Read_Input  MOV.B      [R0],R2
000010020@r  -                CMP.B      #0H,R2
000010023@r  -                BE/Z       cmd_rds:Read_Input
PC = 00001001d@r
  pc= 0001001d  sp= 00030134  fp= 00000000  ap= 00000000 psw=10000001
  r0= 00030000  r1= 00030004  r2= 00000000  r3= 00000000  r4=00000000
  r5= 00000000  r6= 00000000  r7= 00000000  r8= 00000000  r9=00000000
 r10= 00000000 r11= 00000000 r12= 00000000 r13= 00000000 r14=00000000
 r15= 00000000 r16= 00000000 r17= 00000000 r18= 00000000 r19=00000000
 r20= 00000000 r21= 00000000 r22= 00000000 r23= 00000000 r24=00000000
 r25= 00000000 r26= 00000020 r27= 00000000 r28= 00000000 r29=00000000
 r30= 00000000 r31= 00030134


Address      Data (hex)                                      Ascii
-----------  ----------------------------------------------  ----------------
000030000@r  41                                              A

STATUS: N70632--Running in monitor          Emulation trace halted
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Display  Modify  Load  Store  Copy  Find  Report
```

You can continue to step through the program as shown earlier in this chapter to view the instructions which 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 for the allowed commands. The command "A" (41H) was entered above, so the "THIS IS MESSAGE A" message should have been written to the **Message_Dest** locations. Because you must search for hexadecimal values, you will want to search for a sequence of characters which uniquely identify the message, for example,
" A" or 20H and 41H. To search the destination memory location for this sequence of characters, select:

```
Memory, Find
```

Enter the range of the memory locations to be searched, "**30004..30033**" (You can also enter "**Message_Dest..Message_Dest+2f**".), and enter the data "**20,41**" (or, **" ","A"**). The resulting message shows you that the message was indeed written as it was supposed to have been.

To verify that the sample program works for the other allowed commands, you can modify the command input byte to "B" and search for " B" (20H and 42H), or you can modify the command input byte to "C" and search for "D C" (44H, 20H, and 43H).

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 (dependent on the type of instruction being executed and whether the processor is in a hold state).

Using Software Breakpoints

Software breakpoints are realized by the 70632 BRK instruction. When you define or enable a software breakpoint, the emulator will replace the opcode at the software breakpoint address with a breakpoint interrupt instruction (BRK).

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

Note

When using software breakpoints feature of the emulator, you must define the BRK instruction vector to point to an address where instruction fetches is allowed; typically in the program code area. In this sample program, the BRK instruction vector points to a "HALT" instruction. When a software breakpoint occurs, the emulator reads the BRK interrupt vector, push the next PC and PSW to stack, fetch one word of instruction pointed by the vector same as the real CPU. And then, break occurs but the instruction, "HALT" in this example, will never be executed.

There are some notices to use the software breakpoints features. Refer to the "Software Breakpoints" section of the "Using the Emulator" chapter.

Defining a Software Breakpoint

Now that software breakpoints are enabled, you can define software breakpoints. To define a breakpoint at the address of the **Unrecognized** label of the sample program (10053H), select:

```
Breakpoints, Add
```

Enter the local symbol "**cmd_rds:Unrecognized**". After the breakpoint is added, the breakpoint window becomes active and shows that the breakpoint is set. You can add multiple breakpoints in a single command by separating each one with a semicolon. For example, you could type 2010;2018;2052 to set three breakpoints. Run the program by selecting:

Processor, **G**o, **P**c

The status line shows that the emulator is running the user program. Modifying the command input byte to an invalid command (for example, 75H).

Memory, **M**odify, **B**yte

Type "**Command_Input=75**". The following messages will be displayed.

```
ALERT:  Software breakpoint: 000010053@r
STATUS: Running in monitor
```

To continue program execution, select:

Processor, **G**o, **P**c

Displaying Software Breakpoints

To view the status of the breakpoint, select:

Breakpoints, **D**isplay

The resulting display shows that the breakpoint has been cleared.

```
                                   Emulation
 r20= 00000000 r21= 00000000 r22= 00000000 r23= 00000000 r24=00000000
 r25= 00000000 r26= 00000020 r27= 00000000 r28= 00000000 r29=00000000
 r30= 00000000 r31= 00030134


Address      Data (hex)                                              Ascii
-----------  ----------------------------------------------  ----------------
000030000@r  41                                                      A

pattern match at address: 000030013@r

Status   Address
------   ----------
 Set     000010053@r

Status   Address
------   ----------
 Clear   000010053@r

STATUS: N70632--Running user program         Emulation trace halted
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Display  Add  Remove  Set  Clear
```

Setting a Software Breakpoint

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

Breakpoints, **S**et, **S**ingle

The address of the breakpoint you just added is still in the address field; to set this breakpoint again, press **Enter**. As with the "**B**reakpoints **A**dd" command, the Emulation window becomes active and shows that the breakpoint is set.

Clearing a Software Breakpoint

If you wish to clear a software breakpoint that does not get hit during program execution, you can select:

Breakpoints, **C**lear, **S**ingle

The address of the breakpoint set in the previous section is still in the address field; to clear this breakpoint again, press **Enter**.

Using the Analyzer

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.

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 wish to trace the states of the sample program which follow the read of a "B" (42H) command from the command input byte. To do this, you must modify the default analysis specification by selecting:

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

The emulation analysis specification is shown. Use the right arrow key to move to the “Trigger on” field. Type “**a**” and press **Enter**.

You’ll enter the pattern expression menu. Press the up arrow key until the **addr** field directly opposite the pattern **a=** is highlighted. Type the address of the command input byte, using either the global symbol **Command_Input** or address 30000 hex, and press **Enter**.

The “Data” field is now highlighted. Type **0XXXXXX42** hex and press **Enter**. 42H is the value of the “B” command and the “X”s specify “don’t care” values. When 42H is read from the command input byte (30000H), a lower byte read is performed because the address is a multiple of four.

The status qualifiers are defined as follows.

70632 Analysis Status Qualifiers

This trace command example uses the status qualifier “**read**”. The following analysis status qualifiers also can be used with the 70632 emulators.

```
fetch           0x1xxxxxxxxxx011x     code fetch
fetchbr         0x1xxxxxxxxxx0111     code fetch after branch
read            01xxxxxxxxxxxxxxx     read
write           00xxxxxxxxxxxxxxx     write
data            0xxxxxxxxxxxx0011     data access (read/write)
io              0xxxxxxxxxxxx1011     i/o access (read/write)
exec            0xxxxxxxxxxxx0000     execution state
sdata           0xxxxxxxxxxxx0010     data access (read/write) with short path
sysbase         0xxxxxxxxxxxx0100     system base table access
trans           0xxxxxxxxxxxx0101     translation table access (read/write)
coproc          0xxxxxxxxxxxx1000     co-processor access(read/write)
shortrd         0x1xxxxxxxxxx0010     data access read with short path
shortwr         0x0xxxxxxxxxx0010     data access write with short path
iord            0x1xxxxxxxxxx1011     i/o access read
iowr            0x0xxxxxxxxxx1011     i/o access write
transrd         0x1xxxxxxxxxx0101     translation table access read
transwr         0x0xxxxxxxxxx0101     translation table access write
coprocrd        0x1xxxxxxxxxx1000     co-processor access read
coprocwr        0x0xxxxxxxxxx1000     co-processor access write
fault           0xxxxxxxxxxxx1100     machine fault acknowledge
halt            0xxxxxxxxxxxx1101     halt acknowledge
intack          0xxxxxxxxxxxx1110     interrupt acknowledge
grdacc          0xxxxxxxxxx0x0xxx     guarded memory access
wrrom           0x0xxxxxxx0xx0xxx     write to ROM
monitor         0xxxxxxxxxxx0xxxx     background monitor cycle
lock            0xxxxxxxx0xxxxxxx     bus lock
retry           00xxxxxxxxxxxxxxx     retry
hold            0xxxxxxxxxxxx0001     bus hold
```

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

2 Store         any state

   Branches off          Count  off      Prestore off    Trigger position
                                                         start   of 1024
  ←↑↓→ :Interfield movement   Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: N70632--Running user program         Emulation trace halted

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

Figure 2-5. Modifying the Trace Specification

```
-------------------- Internal State Trace Specification --------------------
 ----------------------------------- Set 1 --------------------------------
 Range (r) Label addr   =                       thru
 Pat    ------------addr-------------------------data---------------stat---
  a =  |              Command_Input|                0xxxxxx42|          read|
  b =  |                           |                         |              |
  c =  |                           |                         |              |
  d =  |                           |                         |              |
 ----------------------------------- Set 2 --------------------------------
  e =  |                           |                         |              |
  f =  |                           |                         |              |
  g =  |                           |                         |              |
  h =  |                           |                         |              |
 arm
 --------------------------------- Expression -----------------------------
 Expressions have the form: <set1> and/or <set2>. Where set1 consists of <a,
 b,c,d,r,!r> and set2 consists of <e,f,g,h,arm>. Patterns within a set can be
 joined with ¦(or) or ~(nor), but not both. Example: !r ~ a or e ¦ f ¦ g ¦ h
 Pattern Expression: a
STATUS: N70632--Running user program         Emulation trace halted
Enter-> read

   Use TAB to view choices or enter a status expression. (ex.  read && word)
```

Figure 2-6. Modifying the Pattern Specification

Note

You can combine qualifiers to form more specific qualifiers. For example, the expression **read&data** matches only data reads. See the *Emulator PC Interface Reference* for more information.

Select the **read** status and press **Enter**. Figure 2-5 and figure 2-6 show the resulting analysis specification. To save the new specification, use **End Enter** to exit the field in the lower right corner. You'll return to the trace specification. Press **End** to move to the trigger spec field. Press **Enter** to exit the trace specification.

Starting the Trace

To start the trace, select:

```
Analysis, Begin
```

A message on the status line will show you that the trace is running. You would not expect the trigger to be found because no commands have been entered. Modify the command input byte to "B" by selecting:

```
Memory, Modify, Bytes
```

Enter "**Command_Input=42**". The status line now shows that the trace is complete. (If you have problems, you may be running in monitor. Select **P**rocessor **G**o **P**c to return to the user program.)

Displaying the Trace

To display the trace, select:

```
Analysis, Display
```

You are now given 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 **40** into the ending state field, press **Enter**, and use <CTRL>**z** to zoom the Analysis window.

Note

If you choose to dump a complete trace into the trace buffer, it will take a few minutes to display the trace.

The resulting trace is similar to trace shown in the following display. (You may need to press the **Home** key to get to the top of the trace.)

```
                                    Analysis
   Line    addr,H     data,H     uPD70632 Mnemonic,H                        count,
  -----   --------   --------   -------------------------------------    --------
     -1   ad_Input   ffffff42    No fetch cycle found                     ********
      0   nd_Input   ffffff42      ......42H data  read                   ********
      1   00010020   64226a14    No fetch cycle found                     ********
      2   00010030   44226a14                fetch                        ********
      3   00010023   44226a14    No fetch cycle found                     ********
      4   00010034   ffcdf223                fetch                        ********
      5   ess_Comm   ffcdf223    No fetch cycle found                     ********
      6   00010029   ffcdf223    No fetch cycle found                     ********
      7   00010038   242d0000                fetch                        ********
      8   0001002b   242d0000    No fetch cycle found                     ********
      9   0001003c   000011f4                fetch                        ********
     10   0001002f   44ffffff    No fetch cycle found                     ********
     11   ommand_B   44ffffff                fetch aft br                 ********
     12   00010044   ffcef223                fetch                        ********
     13   00010048   242d0000                fetch                        ********
     14   0001004c   000011f4                fetch                        ********
     15   00010050   44106a00                fetch                        ********

STATUS:  N70632--Running user program          Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Begin  Halt  CMB  Format  Trace  Display
```

Line 0 in the trace list above shows the state which triggered the analyzer. The trigger state is always on line 0. The other states show the exit from the **Command_Input** loop, the **Process_Comm** and **Command_B** instructions.

Press the **PgDn** key to see more lines of the trace.

The resulting display shows the **Command_B** instructions and the branch to **Out_put** and the beginning of the instructions which move the "THIS IS MESSAGE B" message to the destination locations (**Message_Dest**).

```
                              Analysis
      15   00010050  44106a00                fetch                       ********
      16   00010054  ffcff223                fetch                       ********
      17   ommand_B  242d0000   MOVEA.W    cmd_rds:Message_B,R3            ********
      18   00010058  242d0000                fetch                       ********
      19   0001004a  00000ff4   MOV.W      #00000011H,R4                   ********
      20   0001005c  00000ff4                fetch                       ********
      21   s:Output  638a58ff                fetch aft br                ********
      22   00010051  ffa3f284   BR         cmd_rds:Output                  ********
      23   00010064  ffa3f284                fetch                       ********
      24   00010068  6a300001                fetch                       ********
      25   0001006c  000000ae                fetch                       ********
      26   etext     efffffb7                fetch                       ********
      27   00010074  dfffe9f3                fetch                       ********
      28   s:Output  dfffe9f3   MOVCFU.B   [R3],R4,Message_Dest,#30H       ********
      29   00010078  d5655757                fetch                       ********
      30   essage_B  ffff54ff     ....54..H data  read                   ********
      31   00020012  ff48ffff     ..48....H data  read                   ********
      32   age_Dest  00000054     ......54H data  write                  ********
      33   00020013  49ffffff     49......H data  read                   ********

STATUS:  N70632--Running user program         Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Begin  Halt  CMB  Format  Trace  Display
```

Changing the Trace Format

You can modify the trace list format to suit your needs. You can:

- Widen the address column to accommodate longer symbol names.
- Change the port base; to octal; for example.
- Change the count from relative to absolute.

To change the trace format, select:

```
Analysis, Format
```

The bottom part of the "**A**nalysis, **F**ormat" display is used to specify which trace labels appear in which column of the display. Several trace labels are predefined for the internal emulation analyzer.

The default trace display format for the 70632 emulator includes the trace line number (which is always displayed), the hexadecimal address, the 70632 mnemonic and counter field. The default width of the address column is eight characters. The counter field is for displaying either time or occurrence of bus states. By default, analyzer counter is turned off because the clock rate of bus sampling is high-speed. Therefore, the counter field is displayed with "********". You may want to widen the address column to accommodate longer symbol names instead of the senseless counter field.

You can change this by specifying the address column with the optional width parameter. A width of 18 characters is often wide enough to accommodate most symbol names.

Move the cursor to the "Width" field of the **addr** trace label. Enter **18**.

Use the down arrow key to move to the field labeled "**count**." Press **Tab** until it says "**--OFF--**" and press **End**, then **Enter.**

If you display the trace list with the "**A**nalysis, **D**isplay" command, the trace will look like the following.

```
---------------------- Internal State Format Specification ------------------------

     Qualify States   Clock Speed
       user            very fast



 Label Pol Base Width                      Label Pol Base Width
addr       hex   18                       --OFF--
data       hex                            --OFF--
mne                                       --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: N70632--Running user program         Emulation trace complete


      Use the TAB and Shift-TAB keys to select a label or enter a new one.
```

Figure 2-7. Modifying the Trace Format

```
                              Analysis
 Line   addr,H              data,H    uPD70632 Mnemonic,H
-----   -----------------   --------  --------------------------------------
    0   Command_Input       ffffff42    ......42H data  read
    1   00010020            64226a14   No fetch cycle found
    2   00010030            44226a14              fetch
    3   00010023            44226a14   No fetch cycle found
    4   00010034            ffcdf223              fetch
    5   d_rds:Process_Comm  ffcdf223   No fetch cycle found
    6   00010029            ffcdf223   No fetch cycle found
    7   00010038            242d0000              fetch
    8   0001002b            242d0000   No fetch cycle found
    9   0001003c            000011f4              fetch
   10   0001002f            44ffffff   No fetch cycle found
   11   cmd_rds:Command_B   44ffffff              fetch aft br
   12   00010044            ffcef223              fetch
   13   00010048            242d0000              fetch
   14   0001004c            000011f4              fetch
   15   00010050            44106a00              fetch

STATUS: N70632--Running user program         Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Begin  Halt  CMB  Format  Trace  Display
```

Trigger Position

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

- **Start**
- **Center**
- **End**

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

When **Center** trigger position is selected, the trigger is positioned at the center of the trace list. You can trace the states around the trigger state.

When **End** trigger position is selected, the trigger is positioned at the end of the trace list. You can trace the states before the trigger.

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

For example, if you wish to trace the transition to the command A process, change the trigger position to "**Center**".

To reset the analyzer specification defined in previous section by selecting:

`Analysis, Trace, Reset`

Modify the analysis specification by selecting:

`Analysis, Trace, Modify`

The emulation analysis specification is shown. Use the right arrow key to move to the “Trigger on” field. Type “a” and press **Enter**.

You’ll enter the pattern expression menu. Press the up arrow key until the **addr** field directly opposite the pattern **a=** is highlighted. Type the address of the command A process, using either the global symbol **Command_A** or address 10033, and press **Enter**.

To skip the data specification, press **Enter** key.

Now the "Status" field is highlighted. Use the **TAB** key to select the status "**exec**". The following display shows the resulting analysis specification.

```
                     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
                                                        center of 1024
  ←↑↓→ :Interfield movement   Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: N70632--Running user program        Emulation trace complete

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

To save the new specification, use **End Enter** to exit the field in the lower right corner. You'll return to the trace specification. Move the cursor to the "Trigger Position" field. Select the "**Center**" by pressing the **TAB** key.

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

STATUS: N70632--Running user program          Emulation trace complete

TAB selects a simple pattern or enter an expression or move up to edit patterns.
```

Press **End** to move to the trigger spec field. Press **Enter** to exit the trace specification.

To save the new specification, use the **Enter** key to exit out of the field in the lower right corner.

To start the trace, select:

`Analysis, Begin`

A message on the status line will show you that the trace is running. You would not expect the trigger to be found because no commands have been entered. Modify the command input byte to "A" (41H) with "**M**emory, **M**odify, **B**yte" command.

The status line now shows that the trace is complete.

To display the trace, select:

`Analysis, Display`

and press **End Enter**.

```
                                  Analysis
  Line    addr,H              data,H    uPD70632 Mnemonic,H
 -----   ------------------  --------  ---------------------------------------
    -7   00010029            44fdffff   BE/Z      cmd_rds:Command_A
    -6   cmd_rds:Command_A   44ffffff               fetch aft br
    -5   00010034            ffcdf223               fetch
    -4   00010038            242d0000               fetch
    -3   0001003c            000011f4               fetch
    -2   00010040            44206a00               fetch
    -1   00010044            ffcef223               fetch
     0   cmd_rds:Command_A   242d0000   MOVEA.W   cmd_rds:Message_A,R3
     1   00010048            242d0000               fetch
     2   0001003a            000011f4   MOV.W     #00000011H,R4
     3   0001004c            000011f4               fetch
     4   cmd_rds:Output      638a58ff               fetch aft br
     5   00010041            ffa3f284   BR        cmd_rds:Output
     6   00010064            ffa3f284               fetch
     7   00010068            6a300001               fetch
     8   0001006c            000000ae               fetch

STATUS:  N70632--Running user program         Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Begin  Halt  CMB  Format  Trace  Display
```

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

For a Complete Description

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

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 if 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
```

For program code area, enter 10000H through 1006dH (**10000..1006d**) as the source memory range to be copied, and enter **10800** as the destination address. Repeat the "**M**emory, **C**opy" command. For message data area, enter 20000H through 20031H (**20000..20031**) as the source memory range to be copied, and enter **20800** as the destination address.

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

```
Processor, Go, Address
```

Enter **10800**. The status line shows that the emulator is "Running user program". You may wish to trace program execution or enter valid and invalid commands and search the message destination area (as shown earlier in this chapter) to further 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 "processor break", "processor go", or "processor step" command is entered. A CMB execute signal will also cause the emulator to run if reset.

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 two different ways to exit the PC Interface. You can exit the PC Interface using the “locked” option, which restores the current configuration next time you start the PC Interface. You can select this option as follows.

```
System Exit Locked
```

Another way to execute the PC Interface is with the “unlocked” option, which presents the default configuration next time you start the PC Interface. Select this option with the following command.

```
System Exit Unlocked
```

Or , you can exit the PC Interface without saving the current configuration using the command:

```
System Exit No_Save
```

See the *Emulator PC Interface Reference* for a complete description of the system exit options and their effect on the emulator configuration.

Notes

3

Virtual Mode Emulation Topics

Introduction

The on-chip Memory Management Unit (MMU) of the 70632 microprocessor translates virtual addresses to physical (actual) addresses that are placed on the processor address bus. This chapter shows you how to use the emulator when the 70632 MMU is active.

This chapter:

- Describes the sample program used for this chapter's examples.
- Shows you how to enter emulation commands to view execution of the sample program. The commands described in this chapter include:
  - loading the sample program files into the emulator.
  - using virtual address expression in command lines.
  - displaying on-chip MMU registers and privilege registers.
  - displaying address translation tables.
  - displaying TCB.
  - using software breakpoint.
  - using XMMU function.
  - using the analyzer.
  - specifying address mode with suffix.

Sample Program for Virtual Mode Emulation

The sample program consists of the following files.

- *os.s*
- *task_a.s*
- *task_b.s*

These are listed in Figure 3-1 through 3-3. The sample program is a multi-task system which consists of a simple operating system and two tasks. Each of two tasks transfers its own message from an independent area to a common area.

```
.file   "os.s"

.equ    isp,0
.equ    l0sp,1
.equ    l1sp,2
.equ    l2sp,3
.equ    l3sp,4
.equ    sbr,5
.equ    tr,6
.equ    sycw,7
.equ    tkcw,8
.equ    pir,9
.equ    psw2,15
.equ    atbr0,16
.equ    atlr0,17
.equ    atbr1,18
.equ    atlr1,19
.equ    atbr2,20
.equ    atlr2,21
.equ    atbr3,22
.equ    atlr3,23
.equ    trmod,24
.equ    adtr0,25
.equ    adtr1,26
.equ    adtmr0,27
.equ    adtmr1,28

.equ    Stack_Size,0x1000
.equ    Dest_Size,0x20

.globl  Sys_SBT, Current_Task, Num_Of_Task, TCB_Entry, ATE0
.globl  ATE1_A, ATE1_B, PTE0, PTE!_A, PTE1_B, Sys_Stack, Sys_Init
.globl  Setup_Task, Start_Ini_Task, Sys_Trap, Switch_Task, Message_Dest
```

Figure 3-1. Sample Program Source os.s

```
                .data   "sys_sbt" (RW) >0x00000000
Sys_SBT:
                .org    0x34
                .word   Dummy_Text
                .org    0xc0
                .word   Sys_Trap


                .data   "sys_tcb" (RW) >0x00001000
Current_Task:   .word   0
Num_Of_Task:    .word   2

TCB_Entry:      .word   TCB_A
                .word   0x7f000000
                .word   0x00000000
                .word   0x40000000
                .word   0x40002000
                .word   0


                .word   TCB_B
                .word   0x7f000000
                .word   0x00000000
                .word   0x40000000
                .word   0x40002000
                .word   0

TCB_A:          .word   0x0000e000
                .space  8*4
                .word   0x00002009,0x00000000

TCB_B:          .word   0x0000e000
                .space  8*4
                .word   0x00002011,0x00000000

                .data   "sys_ate" (RW) >0x00002000
ATE0:           .word   0x00003003,0x00000500

ATE1_A:         .word   0x00003103,0x00000300
ATE1_B:         .word   0x00003203,0x00000300

                .data   "sys_pte" (RW) >0x00003000
PTE0:           .word   0x00000e05
                .word   0x00001e05
                .word   0x00002e05
                .word   0x00003e05
                .word   0x00004e05
                .word   0x00005e05

                .org    0x100
PTE1_A:         .word   0x00007e05
                .word   0x00008e05
                .word   0x00009e05
                .word   0x00006e05

                .org    0x200
PTE1_B:         .word   0x0000ae05
                .word   0x0000be05
```

Figure 3-1. Sample Program Source os.s (Cont'd)

```
                .word   0x0000ce05
                .word   0x00006e05


                .bss    "sys_stk" (RW) >0x00004000
                .lcomm  Sys_Stack,Stack_Size,4


                .text   "sys_text" (RX) >0x00005000
                .align  4

Sys_Init:       mov.w   #Sys_Stack+Stack_Size,sp
                ldpr    #Sys_SBT,#sbr

                ldpr    #0x2001,#atbr0
                ldpr    #0x00000000,#atlr0
                ldpr    #0,#atbr1
                ldpr    #0,#atbr2
                ldpr    #0,#atbr3

                ldpr    #0x2171,#sycw

Setup_Task:     mov.w   Num_Of_Task,r0
                mov.w   #TCB_Entry,r1
Setup_Task_0:   ldtask  4[r1],[r1]
                mov.w   0x10[r1],r2
                mov.w   #0,[-r2]
                mov.w   8[r1],[-r2]
                mov.w   12[r1],[-r2]
                mov.w   r2,4[[r1]]
                add.w   #0x18,r1
                dbr     r0,Setup_Task_0

                ldtask  TCB_Entry+4,TCB_Entry
Start_Ini_Task: retis   #4

                .align  4
Sys_Trap:       mov.w   Current_Task,r0
                mov.w   r0,r2
                mul.w   #0x6,r2
                mov.w   #TCB_Entry,r1
                sttask  4[r1](r2)
                inc.w   r0
                cmp.w   r0,Num_Of_Task
                jnz     Sys_Trap_0
                xor.w   r0,r0
Sys_Trap_0:     mov.w   r0,Current_Task
                mul.w   #0x6,r0
                ldtask  4[r1](r0),[r1](r0)
Switch_Task:    retis   #4

Dummy_Text:     halt

                .bss    "shr_mem" (RW) >0x40003000
                .lcomm  Message_Dest, Dest_Size,4
```

Figure 3-1. Sample Program Source os.s (Cont’d)

```
                .file   "task_a.s"

                .equ    Stack_Size,0x1000
                .equ    Message_Dest,0x40003000
                .equ    Dest_Size,0x20

                .globl  Transfer_A, Stack_A, Message_A

                .text   "text_a" (RW) >0x40000000
Transfer_A:     mov.w   #' ',r26
                mov.w   #Message_A_End-Message_A,r24
                movcfu.b Message_A,r24,/Message_Dest,#Dest_Size
                trap    #0xa0
Trans_A_End:    jmp     Transfer_A

                .bss    "stack_a"       (RW) > 0x40001000
                .lcomm  Stack_A,Stack_Size,4

                .data   "data_a" (RW) >0x40002000
Message_A:      .str    "THIS IS TASK A MESSAGE."
Message_A_End:
```

Figure 3-2. Sample Program Source task_a.s

```
                .file   "task_b.s"

                .equ    Stack_Size,0x1000
                .equ    Message_Dest,0x40003000
                .equ    Dest_Size,0x20

                .globl  Transfer_B, Stack_B, Message_B

                .text   "text_b" (RW) >0x40000000
Transfer_B:     mov.w   #' ',r26
                mov.w   #Message_B_End-Message_B,r24
                movcfu.b Message_B,r24,/Message_Dest,#Dest_Size
                trap    #0xa0
Trans_B_End:    jmp     Transfer_B

                .bss    "stack_b"       (RW) > 0x40001000
                .lcomm  Stack_B,Stack_Size,4

                .data   "data_b" (RW) >0x40002000
Message_B:      .str    "Task B : Running..."
Message_B_End:
```

Figure 3-3. Sample Program Source task_b.s

os.s

System Base Table The "sys_sbt" section defines the 70632 Break-point instruction trap vector and the Software trap 0 vector. The break-point instruction vector is required for the software breakpoint feature of the emulator. The software trap 0 vector is used for aborting task and transfering execution to the operating system.

Task Context Block The "sys_tcb" section defines task context block. The operating system manages tasks with this block.

The address labeled **Current_Task** contains a task number which is currently executed. Tasks are numbered from 0. This address initialized to 0 when the program is started. First, the task numbered 0 will be executed.

The address labeled **Num_Of_Task** contains the number of tasks the operating system manages. This program has two tasks, which are alternately executed. So this address contains the value "2".

The address labeled **TCB_Entry** contains task control blocks for each task. Each block consists of pointer and register list of TCB managed under the 70632 processor, and the initial values of registers PSW, PC and SP, and a word of flags.

The address labeled **TCB_A** contains the TCB, managed under the processor, for one of the tasks. This task will be called as "*Task A*" in this example. The task number mentioned above is "0".

The address labeled **TCB_B** contains the TCB for the other task, which will be called as "*Task B*". The task number is "1".

Area Table Entry The "ate" section defines the 70632 Area Table Entry.

The address labeled **ATE0** contains Area Table Entry (ATE) in Section 0. In this example, Section 0 is a common part between *Task A* and *Task B.* Section 0 has one area.

The address labeled **ATE1_A** and **ATE1_B** contains ATE in Section 1 for each task. Section 1 is independent between *Task A* and *Task B.* **ATE1_A** is for *Task A*, and **ATE1_B** is for *Task B.* Section 1 has one area each other.

Page Table Entry The "pte" section defines the 70632 Page Table Entry.

The address labeled **PTE0** contains Page Table Entry (PTE) in Section 0, Area 0. This area has six pages.

The addresses labeled **PTE1_A** and **PTE1_B** contains PTE in Section 1, Area 0 for each task. **PTE1_A** is for *Task A*, and **PTE1_B** is for *Task B*. Each area has four pages.

System Stack The "sys_stk" section defines a stack for the operating system. The stack is pointed by the register ISP.

System Program Code The "sys_text" section defines program codes for the operating system.

The program instructions from the **Sys_Init** label to the **Setup_Task** perform initialization of the operating system. The privilege registers are set up and the processor address mode is switched to virtual mode.

The instructions from the **Setup_task** to **Start_Ini_Task** perform initialization for the tasks. The stack for each task is set up with initial PC and PSW.

The instructions from **Start_Ini_Task** transfer the execution to initial task (*Task A*).

The instructions from **Sys_Trap** perform switching task. When a task aborts the execution, the processor executes from the address labeled **Sys_Trap**. The instructions store the task execution environment of the aborted task to corresponding TCB, update the **Current_Task** to the another task number to be switched, load the TCB, and switch the execution.

Common Destination Area The "common" section defines common destination of message from both *Task A* and *Task B* tasks.

The real location is at 00006000H through 00006fffH. Both locations of the *Task A* and *Task B* virtual space are at the same address range 40003000H through 40003fffH.

task_a.s

Task A Program Code The "text_a" section defines program codes of *Task A*. This section is located at 40000000H through 40000fffH by the on-chip MMU.

Task A transfers a message of character string data from the address labeled **Message_A** to the address labeled **Message_Dest**. After the transfer, the processor executes trap instruction. The trap instruction causes the execution aborting into the operating system. At this time, the execution of *Task A* is stopped until next dispatch by the operating system.

Task A Stack The "stack_a" section defines stack of *Task A*. The location of the *Task A* virtual space is at 40001000H through 40001fffH.

Task A Data The "data_a" section defines the message transferred by *Task A*.

The virtual location is 40002000H through 40002fffH.

task_b.s

Task B Program Code The "text_b" section defines program codes of *Task B*.

The virtual location is 40000000H through 40000fffH as same as *Task A*. *Task B* does same as *Task A* except for the message data, which is located at address labeled **Message_B**.

Task B Stack The "stack_b" section defines stack of *Task B*. The location of the *Task B* virtual space is at 40001000H through 40001fffH as same as *Task A*.

Task B Data The "data_b" section defines the message transferred by *Task B*.

This virtual location is 40002000H through 40002fffH as same as *Task A*.

Assembling and Linking the Sample Program

The sample program source files *os.s*, *task_a.s* and *task_b.s* are written for the *HP 64879 70632 Assembler/Linker* hosted on HP-UX.

Following commands were used to generate the absolute files with *HP 64879 70632 Assembler/Linker*.

```
$as70616 -a os.s> os.lis<RETURN>
$as70616 -a task_a.s> task_a.lis<RETURN>
$as70616 -a task_b.s> task_b.lis<RETURN>
$ld70616 -m -o os os.o >os.map<RETURN>
$ld70616 -m -o task_a task_a.o > task_a.map<RETURN>
$ld70616 -m -o task_b task_b.o >task_b.map<RETURN>
$cf70616 -m -o mul_task.a mul_task.cfc >mul_task.cfm<RETURN>
$ar70616 -x mul_task.a os.cf<RETURN>
$ar70616 -x mul_task.a task_a.cf<RETURN>
$ar70616 -x mul_task.a task_b.cf<RETURN>
```

The assembler, linker and the other tools are hosted on HP-UX. File *mul_task.cfc* is a command file for the configurator *cf70616.* The command file is shown in Figure 3-4.

```
SPACE(OS) 0x0  < {os}
SPACE(TASK_A)  < {task_a}
SPACE(TASK_B)  < {task_b}
```

Figure 3-4. Configurator Command File

Setting up the Emulator

Start up the 70632 PC Interface by entering the following command from MS-DOS prompt:

```
C>pcv70 <emulname>
```

The following tasks are required to set up the emulator for the sample program *mul_task*.

Mapping Memory

To map memory for the sample program, select:

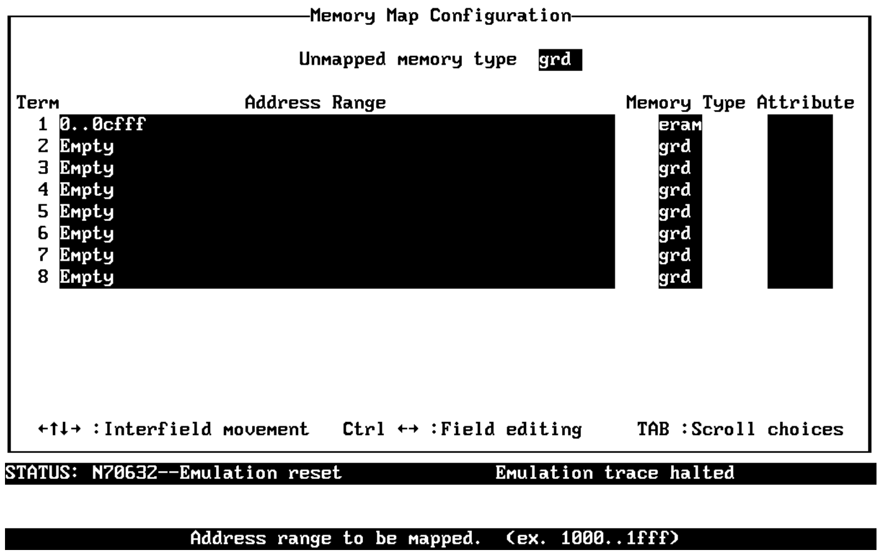
```
Config, Map, Modify
```

To change the unmapped area attribute to "guarded", select **grd** by using the **TAB** key. The sample program *mul_task* occupies address range 0 through 0cfffH of actual memory.

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

```
0..0cfff
```

Move the cursor to the "memory type" field of term 1, and press the **TAB** key to select the **eram** (emulation RAM) type. To save your memory map, use the right arrow key or 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.)



Loading Program into Memory

Load the absolute file by selecting:

```
Memory, Load
```

Select the "**NEC_COFF**" for the format of the absolute files. Select the either "**Emulation**" or "**Any**" for the memory type to be loaded. The absolute files to be loaded into memory are *os.cf*, *task_a.cf* and *task_b.cf*. First, you must load *os.x*.

To load *os.cf*, select **yes** in the "Generate load address information in real addresses" field, and select **yes** in the "Add address attributes to the symbol file" field.

Type **os.cf** in the last field and press **Enter**.

The *task_a.cf* and *task_b.cf* should be loaded after its virtual space is set up.

```
                         Memory Load Configuration

   File Format                                                  NEC_COFF

   Target memory type for memory load                           Both

   Force the absolute file to be read                           no

   Delete a leading underscore character from symbol names      yes

   Generate load address information in real addresses          yes

   Add address attributes to the symbol file                    yes

   Absolute file name
   os.cf

 ←↑↓→ :Interfield movement    Ctrl ←→ :Field editing     TAB :Scroll choices
STATUS: N70632--Emulation reset             Emulation trace halted


Enter the name of an NEC COFF file.
```

Figure 3-5. Loading the Sample Program into Memory

Displaying and Transferring the Symbols for os.x

The sample program is executed from the address **Sys_Init**. Display the global symbols by selecting:

`System, Symbol, Global, Display`

Type <CTRL>**z** to zoom the symbol window.

```
                                      Symbols
 Modules
---------
 os

Address     Symbol
----------  ------
000002000@r  ATE0
000002008@r  ATE1_A
000002010@r  ATE1_B
000001000@r  Current_Task
000006000@r  Message_Dest
000001004@r  Num_Of_Task
000003000@r  PTE0
000003100@r  PTE1_A
000003200@r  PTE1_B
000005043@r  Setup_Task
000005084@r  Start_Ini_Task
0000050BF@r  Switch_Task
000005000@r  Sys_Init

STATUS: N70632--Emulation reset              Emulation trace halted
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Command_file  Wait  MS-DOS  Log  Terminal  Symbols  Exit
```

The *os.x* includes the module *os*, display the local symbols of the module *os*.

`System, Symbol, Local, Display`

Specify the module name to be displayed, by typing **os** and pressing **Enter**.

```
                              Symbols
Address      Symbol
----------   ------
0000050C1@r  Dummy_Text
000005051@r  Setup_Task_0
0000050AE@r  Sys_Trap_0
000001038@r  TCB_A
000001064@r  TCB_B

STATUS: N70632--Emulation reset              Emulation trace halted
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Command_file  Wait  MS-DOS  Log  Terminal  Symbols  Exit
```

Getting into Virtual Mode

Before starting the program, define software breakpoint at the address **Start_Ini_Task**. This address is the exit of the operating system.

`Breakpoints, Add`

Enter **Start_Ini_Task** and press **Enter**.

Then start the program from the address **Sys_Init**.

`Processor, Go, Address`

Enter **Sys_Init** and press **Enter**.

You will see the following line in the status lines.

```
ALERT:  Software breakpoint: 0000005084@v
```

The processor executed the following tasks from **Sys_Init** to **Start_Ini_Task**.

- Initializing privilege registers (stack pointer and area table registers)
- Initializing Task Context Blocks for *Task A* and *Task B.*
- Switching to *Task A*

Loading the Absolute File in Virtual Mode (task_a.cf)

The emulator broke just before the transition from operating system to *Task A*. Since the current virtual space is for *Task A*, you can load the absolute file *task_a.cf*. Before loading the file, you must change the configuration option "Load real address", select:

`Config, General`

Move the cursor to the "Load real address" field, and enter "**n**". Move the cursor to the bottom field and press **Enter** to save this configuration.

```
──────────────────────General Emulation Configuration─────────────────────────
Internal clock        [y] Real-time mode      [n] Break on ROM writes   [y]
Software brkpoints    [y] Trace HOLD Tag      [y] Load real address     [n]
Trace execution cycle [y] Trace real address  [y] CMB interaction       [n]
V70 inverse assemble  [y] Respond to HLDRQ    [y] Respond to target NMI [y]
Target interrupts     [y] Target bus freeze   [y] Drive backgrnd cycles [n]
Target memory access size       bytes  Address bits A31-A8     0
Monitor type               background

  ←↑↓→ :Interfield movement   Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: N70632--Running in monitor           Emulation trace halted
When 'yes' is selected, object files will be loaded in real address mode.
Otherwise, object files will be loaded in virtual address mode.
```

To load the file *task_a.cf*, select:

`Memory, Load`

Move the cursor to the "Generate load address information in real address" field and select "**no**", and move the cursor to the "Absolute file name" field, and type the **task_a.cf** then press **Enter**.

```
                    Memory Load Configuration

  File Format                                         NEC_COFF

  Target memory type for memory load                  Both

  Force the absolute file to be read                  no

  Delete a leading underscore character from symbol names  no

  Generate load address information in real addresses  no

  Add address attributes to the symbol file           yes

  Absolute file name
  task_a.cf

 ←↑↓→ :Interfield movement   Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: N70632--Running in monitor          Emulation trace halted


Enter the name of an NEC COFF file.
```

Since loading an absolute file clears previous loaded symbols, you should reload the symbols for *os.cf*, select:

System, **S**ymbol, **G**lobal, **L**oad

Press **Enter** to load the symbol file **os.hps**.

Define software breakpoint at the address **Switch_Task**. This address is the exit of the task dispatcher.

Breakpoints, **A**dd

Enter **Switch_Task** and press **Enter**.

Continue the execution.

Processor, **G**o, **P**c

You will see the following line in the status lines.

```
ALERT:  Software breakpoint: 00000050bf@v
```

The processor executed the following tasks.

- Transferring the message of *Task A*
- Exiting the execution of *Task A*
- Storing the Task Context for *Task A*
- Loading the Task Context for *Task B*
- Switching to *Task B*

Loading the Absolute File, task_b.cf

The emulator broke just before the transition from task dispatcher to *Task B*. Since the current virtual space is for *Task B*, you can load the absolute file *task_b.cf*.

To load the file *task_b.cf*, select:

```
Memory, Load
```

Move the cursor to the "Absolute file name" field, and type the **task_b.cf** then press **Enter**.

Reload the symbols for *os.cf*, select:

```
System, Symbol, Global, Load
```

Press **Enter** to load the symbol file **os.hps**.

Transfer the symbols to the emulator by selecting.

```
System, Symbol, Global, Transfer
System, Symbol, Local, Transfer, All
```

Displaying Registers

Display basic registers by selecting:

```
Register, Display, Basic
```

Type <CTRL>**z** to zoom the Emulation window.

You can also display privilege and on-chip MMU registers, select:

```
Register, Display, Class
```

Use the **TAB** key to select **priv** and press **Enter**.

```
Register, Display, Class
```

Use the **TAB** key to select **mmu** and press **Enter**.

```
                                   Emulation
 Clear    000005084@r
 Set      0000050bf@r

  pc= 000050bf  sp= 40001ff4  fp= 00000000  ap= 00000000 psw=80000000
  r0= 00000006  r1= 00001008  r2= 00000000  r3= 00000000  r4=00000000
  r5= 00000000  r6= 00000000  r7= 00000000  r8= 00000000  r9=00000000
 r10= 00000000 r11= 00000000 r12= 00000000 r13= 00000000 r14=00000000
 r15= 00000000 r16= 00000000 r17= 00000000 r18= 00000000 r19=00000000
 r20= 00000000 r21= 00000000 r22= 00000000 r23= 00000000 r24=00000000
 r25= 00000000 r26= 00000000 r27= 00000000 r28= 00000000 r29=00000000
 r30= 00000000 r31= 40001ff4

   sbr= 00000000    tr= 00001064  sycw= 00002171  tkcw= 0000e000
   pir= 00007006   isp= 00005000  psw2= 0000f002
  l0sp= 40001ff4  l1sp= 00000000  l2sp= 00000000  l3sp= 00000000

  atbr0= 00002001  atbr1= 00002011  atbr2= 00000000  atbr3= 00000000
  atlr0= 00000000  atlr1= 00000000  atlr2= 00000000  atlr3= 00000000

STATUS: N70632--Running in monitor           Emulation trace halted
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Display  Modify  Float
```

Tracing the Program Execution

Suppose that you wish to trace the program from the current address.

Modifying Trace Format

For widening the address column of the trace listing, do the following.

To change the trace format, select:

`Analysis, Format`

Move the cursor to the "Width" field of the **addr** trace label. Enter **18**.

Use the down arrow key to move to the field labeled **count**. Press **Tab** until it says "**--OFF--**" and press **End**, then **Enter.**

Resetting Trace Specification

To be sure that the analyzer is in its default state, select:

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

The default trace specification triggers the analyzer as soon as possible if the program is running user program. The emulator is running in monitor because the software breakpoint has hit. To trace the program execution from the current address, you do not have to change the initial trace specifications. Start the trace and continue the program.

Analysis, **B**egin

Processor, **G**o, **P**c

The status line shows that the emulation trace is completed.

To display the trace listing, select:

Analysis, **D**isplay

Use the right arrow key to move the cursor to the "Ending state to display" field. Type **100** into the ending state field, press **Enter** and use <CTRL>**z** to zoom the analysis window. Press the **Home** key to get to the top of the display. The resulting trace is similar to the following display.

The trace listing shows the beginning of the execution by task dispatcher. Press the **PgDn** key to see more lines of the trace.

```
                                          Analysis
   Line   addr,H               data,H    uPD70632 Mnemonic,H
  -----   ------------------   --------  --------------------------------------
      0   ATE0                 00003003     00003003H trans table  read
      1   00002004             00000500     00000500H trans table  read
      2   00003014             00005e85     00005e85H trans table  read
      3   Switch_Task          faffffff               fetch aft br
      4   000050c0             000000e4               fetch
      5   esys_text__          020b4001               fetch
      6   000050c8             aaa92eba               fetch
      7   000050cc             0aa2b8aa               fetch
      8   ATE1_B               00003203     00003203H trans table  read
      9   00002014             00000300     00000300H trans table  read
     10   00003204             0000bf85     0000bf85H trans table  read
     11   0000bff4             40000000     40000000H data  read
     12   Switch_Task          00000000   RETIS     #4H
     13   0000bff8             00000000     00000000H data  read
     14   ATE1_B               00003203     00003203H trans table  read
     15   00002014             00000300     00000300H trans table  read
     16   PTE1_B               0000ae05     0000ae05H trans table  read

STATUS: N70632--Running user program         Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Begin  Halt  CMB  Format  Trace  Display
```

The display shows the execution of *Task B*, and you can find that the address fields of the trace are displayed in real address. Regardless of

```
                                    Analysis
     16   PTE1_B              0000ae05     0000ae05H trans table  read
     17   PTE1_B              0000ae85     0000ae85H trans table  write
     18   0000a000            20f43a2d               fetch aft br
     19   0000a004            2d000000               fetch
     20   0000a008            0013f438               fetch
     21   0000a00c            8a580000               fetch
     22   0000a000            8a580000   MOV.W     #00000020H,R26
     23   0000a010            001ff2f2               fetch
     24   0000a014            00f39800               fetch
     25   0000a007            20400030   MOV.W     #00000013H,R24
     26   0000a018            20400030               fetch
     27   0000a01c            d6a0f4f8               fetch
     28   0000a020            ffffe1f2               fetch
     29   0000a024            000000ff               fetch
     30   0000a028            5755d755               fetch
     31   ATE1_B              00003203     00003203H trans table  read
     32   00002014            00000300     00000300H trans table  read
     33   00003208            0000ce05     0000ce05H trans table  read
     34   00003208            0000ce85     0000ce85H trans table  write

STATUS:  N70632--Running user program         Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Begin  Halt  CMB  Format  Trace  Display
```

address mode, addresses which the analyzer captures are real addresses by default.

Changing Symbols

The program executes *Task A* and *Task B* alternately. Suppose that you wish to note to *Task B*. In this case, you should load the symbols for *Task B*.

Load the symbols for *Task B*, select:

`System, Symbol, Global, Load`

Type the symbol file name **task_b.hps** and press **Enter**.

Display the global symbols you have loaded, select:

`System, Symbol, Global, Display`

Press <CTRL>**z** key to zoom the symbol window.

The global symbols for *Task B* are displayed.

```
                                 Symbols
 Modules
---------
 task_b

Address     Symbol
----------  ------
040002000@v  Message_B
040001000@v  Stack_B
040000000@v  Transfer_B
040002014@v  edata_b
040002000@v  estack_b
040000028@v  etext_b

STATUS:  N70632--Running user program         Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Command_file  Wait  MS-DOS  Log  Terminal  Symbols  Exit
```

When loading new global symbols, the old global symbols are removed. This means that only one symbol file can be stored in the emulator.

To display local symbols, select:

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

Enter **task_b**. The resulting display follows.

```
                                 Symbols
Address     Symbol
----------  ------
040002013@v  Message_B_End
04000001F@v  Trans_B_End

STATUS:  N70632--Running user program         Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Command_file  Wait  MS-DOS  Log  Terminal  Symbols  Exit
```

Transfer the symbols to the emulator, select:

```
System, Symbol, Global, Transfer
System, Symbol, Local, Transfer, All
```

Specifying Virtual Space

The emulator uses the current value of the 70632 address table register pairs by default when you specify an address in virtual address in a command.

Suppose that you would like to debug a certain task executed in multiple virtual space without stopping the execution. You will be unable to specify the virtual address in desired virtual space, because the address space is dynamically changed.

The XMMU function provides you to specify a desired virtual address space. Regardless of the current virtual space, you can specify the address space you want to note to. The emulator has the optional XMMU class registers. These registers consist of eight XMMU register pairs and one XMMU mode register. The XMMU register pairs correspond to the actual 70632 area table register pairs. You can specify a virtual address space by modifying the XMMU class registers. These registers are not actual registers of the 70632 processor.

When you set the contents of the XMMU class registers and activate the XMMU function, the XMMU class registers are used for the address translation of the virtual address you specify in a command, instead of the actual area table register pairs of the 70632 microprocessor.

The XMMU class registers consist of the following registers.

```
XMMU class registers          corresponded actual registers
xatbr0                        atbr0
xatlr0                        atlr0
xatbr1                        atbr1
xatlr1                        atlr1
xatbr2                        atbr2
xatlr2                        atlr2
xatbr3                        atbr3
xatlr3                        atlr3

mmumod                        --None--
```

If you set the value of the **mmumod** register in the above table to "1", the emulator translates the virtual address in a command line with the contents of the XMMU class registers instead of the actual area table register pairs. Oppositely, if you want to make the emulator to translate the virtual address in a command line with the actual table register pairs, in other words the virtual address in the current address space, reset the value of the **mmumod** register to "0".

To display the XMMU class registers, select:

```
Register, Display, Class
```

Select **xmmu** by using the **TAB** key, and press **Enter**.

Press <CTRL>**z** key to zoom the Emulation window.

```
                                   Emulation
  r0= 00000006  r1= 00001008  r2= 00000000  r3= 00000000  r4=00000000
  r5= 00000000  r6= 00000000  r7= 00000000  r8= 00000000  r9=00000000
 r10= 00000000 r11= 00000000 r12= 00000000 r13= 00000000 r14=00000000
 r15= 00000000 r16= 00000000 r17= 00000000 r18= 00000000 r19=00000000
 r20= 00000000 r21= 00000000 r22= 00000000 r23= 00000000 r24=00000000
 r25= 00000000 r26= 00000000 r27= 00000000 r28= 00000000 r29=00000000
 r30= 00000000 r31= 40001ff4

   sbr= 00000000    tr= 00001064  sycw= 00002171  tkcw= 0000e000
   pir= 00007006   isp= 00005000  psw2= 0000f002
  l0sp= 40001ff4  l1sp= 00000000  l2sp= 00000000  l3sp= 00000000

  atbr0= 00002001  atbr1= 00002011  atbr2= 00000000  atbr3= 00000000
  atlr0= 00000000  atlr1= 00000000  atlr2= 00000000  atlr3= 00000000

  mmumod= 00000000
  xatbr0= 00000000  xatbr1= 00000000  xatbr2= 00000000  xatbr3= 00000000
  xatlr0= 00000000  xatlr1= 00000000  xatlr2= 00000000  xatlr3= 00000000

STATUS: N70632--Running user program         Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Display  Modify  Float
```

The resulting display shows the contents of XMMU class registers. The display also includes the contents of on-chip MMU registers, you displayed in previous section, and these values define virtual space for *Task B*.

Since you want to note to *Task B*, modify the XMMU class registers as the same value as the value of on-chip MMU registers in the display. Select:

```
Register, Modify
```

Use the **TAB** key to select (or enter) **xatbr0** and press **Enter**. Type **2001**, press **Enter**.

```
Register, Modify
```

Use the **TAB** key to select **xatbr1** and press **Enter**. Type **2011**, press **Enter**.

To make the emulator use the configured address space you entered, select:

`Register, Modify`

Use the **TAB** key to select **mmumod** and press **Enter**. Type **1**, press **Enter**.

To confirm the XMMU class registers you have modified, select:

`Register, Display, Class`

Use the **TAB** key to select **xmmu**, and press **Enter**.

```
                                   Emulation
 r20= 00000000 r21= 00000000 r22= 00000000 r23= 00000000 r24=00000000
 r25= 00000000 r26= 00000000 r27= 00000000 r28= 00000000 r29=00000000
 r30= 00000000 r31= 40001ff4

   sbr= 00000000    tr= 00001064  sycw= 00002171  tkcw= 0000e000
   pir= 00007006   isp= 00005000  psw2= 0000f002
  l0sp= 40001ff4  l1sp= 00000000  l2sp= 00000000  l3sp= 00000000

  atbr0= 00002001  atbr1= 00002011  atbr2= 00000000  atbr3= 00000000
  atlr0= 00000000  atlr1= 00000000  atlr2= 00000000  atlr3= 00000000

  mmumod= 00000000
  xatbr0= 00000000  xatbr1= 00000000  xatbr2= 00000000  xatbr3= 00000000
  xatlr0= 00000000  xatlr1= 00000000  xatlr2= 00000000  xatlr3= 00000000

  mmumod= 00000001
  xatbr0= 00002001  xatbr1= 00002011  xatbr2= 00000000  xatbr3= 00000000
  xatlr0= 00000000  xatlr1= 00000000  xatlr2= 00000000  xatlr3= 00000000

STATUS: N70632--Running user program         Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Display  Modify  Float
```

To display the contents of memory at address range **Transfer_B** through **Trans_B_End**. The label **Trans_B_End** is local symbol.

Select:

`Memory, Display, Mnemonic`

Enter the address range "**Transfer_B..task_b:Trans_B_End**", and press **Enter**. Enter <CTRL>**z** to zoom the window.

```
                              Emulation
  atbr0= 00002001  atbr1= 00002011  atbr2= 00000000  atbr3= 00000000
  atlr0= 00000000  atlr1= 00000000  atlr2= 00000000  atlr3= 00000000

  mmumod= 00000000
  xatbr0= 00000000  xatbr1= 00000000  xatbr2= 00000000  xatbr3= 00000000
  xatlr0= 00000000  xatlr1= 00000000  xatlr2= 00000000  xatlr3= 00000000

  mmumod= 00000001
  xatbr0= 00002001  xatbr1= 00002011  xatbr2= 00000000  xatbr3= 00000000
  xatlr0= 00000000  xatlr1= 00000000  xatlr2= 00000000  xatlr3= 00000000

Address      Symbol           Mnemonic
-----------  ---------------  ----------------------------------------------
040000000@v  Transfer_B       MOV.W      #00000020H,R26
040000007@v  -                MOV.W      #00000013H,R24
04000000e@v  -                MOVCFU.B   Message_B,R24,/40003000H,#2
04000001c@v  -                TRAP       #a0H
04000001f@v  k_b:Trans_B_End  JMP        Transfer_B

STATUS: N70632--Running user program        Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Display  Modify  Load  Store  Copy  Find  Report
```

Breakpoints

Before defining the breakpoint, break the emulator by selecting:

`Processor, Break`

To define a breakpoint at the address of **Transfer_B**, select:

`Breakpoints, Add`

Enter the address label **Transfer_B**, and press **Enter**.

Now that the software breakpoint is set, start the execution. Select:

`Processor, Go, Pc`

The status line shows as follows.

```
ALERT:  Software breakpoint: 040000000@v
```

Displaying Address Translation Tables

You can display the 70632 Area Table Entry (ATE) and Page Table Entry (PTE). These features are provided with Terminal Interface. To entering the Terminal Interface, select:

`System, Terminal`

To display the ATE corresponding with address **Transfer_B**, enter:

`ate Transfer_B`

```
1:000 at 000002010  Present
      PTB=000003200  Limit=003  Growth=positive
      Execute level=0  Write level=0  Read level=0
```

To display the PTE corresponding with address **Transfer_B**, enter:

pte Transfer_B

```
1:000:000 at 000003200  Present
          Page base=00000a000  Executable  Writable  Readable
          Not modified  Accessed  User=0  Not locked
```

Displaying TCB

You can display TCB contents of current task by using the **tcb** Terminal Interface command. Specify the register list with **-l** option. The register list specifies registers to be stored to or loaded from TCB when the task is switched. The format of the register list is same as the 70632 processor's LDTASK or STTASK instruction operand. Since the register list of current task is 7f000000H, enter:

tcb -l 7f000000

To exit the Terminal Interface window, type <CTRL>\.

```
                                  Terminal
ate Transfer_B
1:000 at 000002010  Present
      PTB=000003200  Limit=003  Growth=positive
      Execute level=0  Write level=0  Read level=0
M$pte Transfer_B
1:000:000 at 000003200  Present
          Page base=00000a000  Executable  Writable  Readable
          Not modified  Accessed  User=0  Not locked
M$tcb -l 7f000000
  tkcw ATT=7 OTM=0 FIT=0 FZT=0 FVT=0 FUT=0 FPT=0 RDI=0 RD=0
  l0sp=40001ff4
  r24=00000013  r25=00000000  r26=00000020  r27=40003020  r28=40002013
  r29=00000000  r30=00000000
  atrp1 ATB=000002010  Limit=000  Growth=positive  Valid
M$

STATUS: Unknown - polling disabled

                                                       Press Ctrl \ to abort
```

Tracing Virtual Address

The analyzer can capture virtual address by modifying configuration.

To configure to make the analyzer capture the virtual address, select:

Config, **G**eneral

Use the arrow key to the "Trace real address" configuration option field. Enter "**n**" to change trace address from real to virtual. Move the cursor to the bottom field, and press **Enter** to save the configuration.

Specifying Trigger

To trigger the analyzer when reading from the address **Message_B**.

`Analysis, Trace, Modify`

The emulation analysis specification is shown. Use the right arrow key to move to the “Trigger on” field. Type “a” and press **Enter**.

You’ll enter the pattern expression menu. Press the up arrow key until the **addr** field directly opposite the pattern **a=** is highlighted. Type the global symbol **Message_B** and press **Enter**.

To skip the data specification, press **Enter** key.

Now the "Status" field is highlighted. Use the **TAB** key to select the status "**read**".

The following display shows the resulting analysis specification. To save the new specification, use **End Enter** to exit the field in the lower right corner.

```
                    Internal State Trace Specification
                                   Set 1
Range (r) Label addr    =                        thru
Pat              addr                    data                  stat
 a =                   Message_B                                   read
 b =
 c =
 d =
                                   Set 2
 e =
 f =
 g =
 h =
arm
                                 Expression
Expressions have the form: <set1> and/or <set2>. Where set1 consists of <a,
b,c,d,r,!r> and set2 consists of <e,f,g,h,arm>. Patterns within a set can be
joined with |(or) or ~(nor), but not both. Example: !r ~ a or e | f | g | h
Pattern Expression: a
STATUS: N70632--Running in monitor          Emulation trace complete

 The TAB key selects whether the pattern matches the values or not the values.
```

You’ll return to the trace specification. Move the cursor to the "Trigger Position" field. Select the "**Center**" by pressing the **TAB** key.

```
─────────────── 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
                                                      center of 1024
  ←↑↓→ :Interfield movement   Ctrl ←→ :Field editing    TAB :Scroll choices
STATUS: N70632--Running in monitor           Emulation trace complete

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

Press **End** to move to the trigger spec field. Press **Enter** to exit the trace specification.

Move the cursor to "Trigger Position" field, and select **Center**. Move the cursor to the lower right corner, and press **Enter** to save the trace specification.

To save the new specification, use the **Enter** key to exit out of the field in the lower right corner.

To start the trace, select:

`Analysis, Begin`

The status line shows that the trace is running.

To continue the execution, select:

`Processor, Go, Pc`

The trace status changes to "Emulation trace complete".

To display the trace, select:

`Analysis, Display`

Press **End Enter**, and use <CTRL>**z** to zoom the window.

```
                                   Analysis
  Line   addr,H              data,H    uPD70632 Mnemonic,H
 -----   ------------------  --------  --------------------------------------
    -7   ATE1_B              00003203     00003203H trans table  read
    -6   00002014            00000300     00000300H trans table  read
    -5   00003208            0000ce85     0000ce85H trans table  read
    -4   ATE1_B              00003203     00003203H trans table  read
    -3   00002014            00000300     00000300H trans table  read
    -2   0000320c            00006f85     00006f85H trans table  read
    -1   4000000e            00006f85   MOVCFU.B  Message_B,R24,/40003000H,
     0   Message_B           ffffff54     ......54H data  read
     1   40002001            ffff61ff     ....61..H data  read
     2   40003000            00000054     ......54H data  write
     3   40002002            ff73ffff     ..73....H data  read
     4   40003001            00006100     ....61..H data  write
     5   40002003            6bffffff     6b......H data  read
     6   40003002            00730000     ..73....H data  write
     7   40002004            ffffff20     ......20H data  read
     8   40003003            6b000000     6b......H data  write

STATUS: N70632--Running user program         Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Begin  Halt  CMB  Format  Trace  Display
```

The resulting display shows the transfer of the *Task B* message.

Press the <CTRL>**r** key to see more lines. Then you will see the transition from *Task B* to the task dispatcher.

```
                                   Analysis
  Line   addr,H              data,H    uPD70632 Mnemonic,H
 -----   ------------------  --------  --------------------------------------
     9   40002005            ffff42ff     ....42..H data  read
    10   40003004            00000020     ......20H data  write
    11   40002006            ff20ffff     ..20....H data  read
    12   40003005            00004200     ....42..H data  write
    13   40002007            3affffff     3a......H data  read
    14   40003006            00200000     ..20....H data  write
    15   40002008            ffffff20     ......20H data  read
    16   40003007            3a000000     3a......H data  write
    17   40002009            ffff52ff     ....52..H data  read
    18   40003008            00000020     ......20H data  write
    19   4000200a            ff75ffff     ..75....H data  read
    20   40003009            00005200     ....52..H data  write
    21   4000200b            6effffff     6e......H data  read
    22   4000300a            00750000     ..75....H data  write
    23   4000200c            ffffff6e     ......6eH data  read
    24   4000300b            6e000000     6e......H data  write

STATUS: N70632--Running user program         Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Begin  Halt  CMB  Format  Trace  Display
```

Press the <CTRL>**r** key several times until the Task A execution is displayed (The states of Task B will be stored from line 134).

```
                                 Analysis
   Line   addr,H              data,H    uPD70632 Mnemonic,H
  -----   ------------------  --------  -------------------------------------
    121   0000105c            00002009     00002009H data  read
    122   00001060            00000000     00000000H data  read
    123   00001064            0000e000     0000e000H data  read
    124   000050bf            faffffff               fetch aft br
    125   000050c0            000000e4               fetch
    126   000050c4            020b4001               fetch
    127   000050c8            aaa92eba               fetch
    128   000050cc            0aa2b8aa               fetch
    129   ATE1_A              00003103     00003103H trans table  read
    130   0000200c            00000300     00000300H trans table  read
    131   00003104            00008f85     00008f85H trans table  read
    132   40001ff4            4000001f     4000001fH data  read
    133   000050bf            00000000   RETIS     #4H
    134   40001ff8            00000000     00000000H data  read
    135   ATE1_A              00003103     00003103H trans table  read
    136   0000200c            00000300     00000300H trans table  read

STATUS:  N70632--Running user program         Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Begin  Halt  CMB  Format  Trace  Display
```

```
                                 Analysis
   Line   addr,H              data,H    uPD70632 Mnemonic,H
  -----   ------------------  --------  -------------------------------------
    137   PTE1_A              00007e85     00007e85H trans table  read
    138   task_b:Trans_B_End  d6ffffff               fetch aft br
    139   40000020            ffffe1f2               fetch
    140   40000024            000000ff               fetch
    141   etext_b             5550c511               fetch
    142   4000002c            75315575               fetch
    143   Transfer_B          20f43a2d               fetch aft br
    144   task_b:Trans_B_End  2d000000   JMP     Transfer_B
    145   40000004            2d000000               fetch
    146   40000008            0017f438               fetch
    147   4000000c            8a580000               fetch
    148   Transfer_B          8a580000   MOV.W   #00000020H,R26
    149   40000010            001ff2f2               fetch
    150   40000014            00f39800               fetch
    151   40000007            20400030   MOV.W   #00000017H,R24
    152   40000018            20400030               fetch

STATUS:  N70632--Running user program         Emulation trace complete
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Begin  Halt  CMB  Format  Trace  Display
```

As you can see some address are replaced with the symbols for *Task B*. You may confuse the states with Task B's one. The reason is because *Task A* and *Task B* occupy the same virtual address (not the same virtual space) each other.

Address Mode Suffixes

When you issue a command, the emulator displays the result of the command. According to circumstance, the resulting display includes address information such as "00004000@r" or "00008000@v".

The suffix "@r" indicates that the address is displayed in real address mode. The suffix "@v" indicates that the address is displayed in virtual address. When the emulator displays an address information, the address mode will be different as the case may be.

Specifying An Address Mode

When you designate addresses, you can also select either real or virtual address by adding a suffix. The following suffixes are allowed.

- "**@r**" real address
- "**@v**" virtual address

You can also designate addresses with no suffix. In this case, the address mode, which is required to evaluate the addresses, is determined as follows.

1. When the processor is reset, the addresses are evaluated as real address.

2. When the processor never runs in virtual mode after reset, the addresses are evaluated as real address.

3. Once the processor has run in virtual mode after reset, the addresses are evaluated as virtual address.

Note

If the processor has ever run in virtual mode since the processor was reset, the address expression without suffix is evaluated as virtual address, even if the processor is running in real mode.

If you specify a virtual address in a command, the emulator has to translate the virtual address, which you have specified,to the real address. The method of the address translation is same as the actual 70632 microprocessor. In this case, the emulator use the current value of the 70632 address table register pairs, ATBR0, ATLR0, ATBR1 ...., to translate the address by default. The details of the address translation are shown in chapter 4.

Symbols supported by the PC Interface can include the suffixes. To include the suffixes in symbols, use the "**-a**" option for the *rdnec70* converter.

Note

The following commands do not accept inconsistent suffix.

```
"Processor, Go, Address"
"Processor, Step, Address"
"Processor, Go, CMB, Address"
```

For example, when the emulator is in real mode, you can not specify a virtual address (with "@v" suffix) in one of the above command.

Notes

4

Configuring the 70632 Emulator

Introduction

Your 70632 emulator can be used in all stages of target system development. For instance, you can run the emulator out-of-circuit when developing target system software, or you can use the emulator in-circuit when integrating software with target system hardware. Emulation memory can be used in place of, or along with, target system memory. You can use the emulator's internal clock or the target system clock. You can execute target programs in real-time or allow emulator execution to be diverted into the monitor when commands request access of target system resources (target system memory, register contents, etc.) The emulator is a flexible instrument and it may be configured to suit your needs at any stage of the development process. This chapter describes the options available when configuring the 70632 emulator.

This chapter will:

- Show you how to access the emulator configuration options.
- Describe the emulator configuration options
- Show you how to save a particular emulator configuration, and load it again at a later time.

Prerequisites

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

General Configuration

Select:

Config, **G**eneral

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

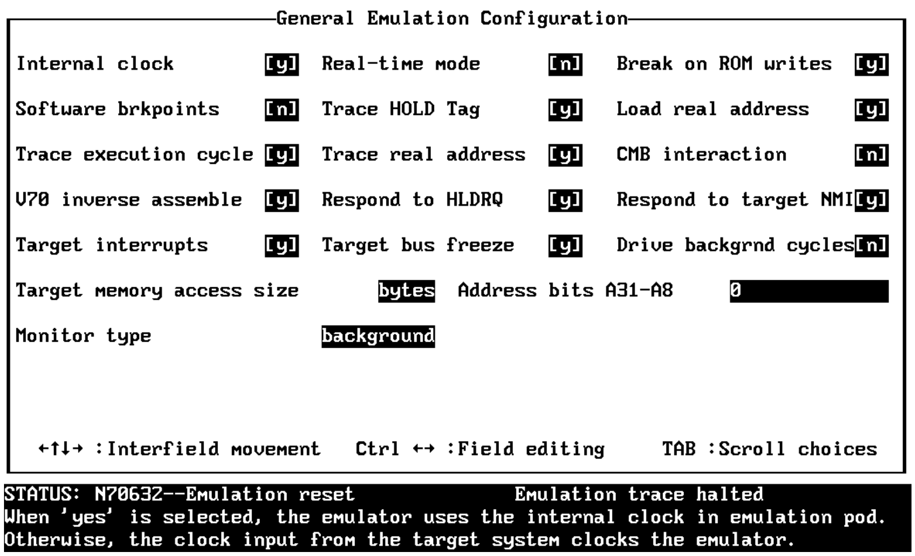


Figure 4-1. General Emulator Configuration

Note

It is possible to use the System Terminal window to modify the emulator configuration. However, if you do this, some PC Interface features may no longer work properly. We recommend that you only modify the emulator configuration by using the options presented in the PC Interface.

Internal Clock

This configuration item allows you to select whether the emulator will be clocked by the internal clock source or by a target system clock source.

Yes — The internal 20 MHz clock (system clock speed) is the emulator clock source. This is the default.

No — An external target system clock is the emulator clock source. External clock sources must be within the range of 8-20 MHz.

Real-Time Mode

If it is important that the emulator execute target system programs in real-time, you can enable the real-time emulator mode. In other words, when you execute target programs (with the "**P**rocessor, **G**o" command), the emulator will execute in real-time.

No — The default emulator configuration disables the real-time mode. When the emulator is executing the target program, you are allowed to enter emulation commands that require access to target system resources (display/modify: registers, target system memory, or target system I/O etc.). If one of these commands is entered, the system controller will temporarily break emulator execution into the monitor. These command are described in the "Target Memory Access" section of chapter 5.

Yes — If your target system program requires real-time execution, you should enable the real-time mode in order to prevent temporary breaks that might cause target system problems.

Break on ROM Writes

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

Yes — Emulator execution will break into the monitor when the target program writes to ROM locations.

No — Target program writes to ROM locations will not cause emulator execution to break into the monitor.

Software Breakpoints

The software breakpoint feature uses the BRK instruction. When you add or set a software breakpoint (and software breakpoints are enabled), the emulator will replace the opcode at the software breakpoint address with the BRK instruction. When the emulator executes the BRK instruction, execution breaks into the monitor.

If your target program uses BRK instruction and contains a breakpoint interrupt routine, you may wish to disable the software breakpoints feature so that BRK instructions do not cause breaks to the monitor. Refer to the "Software Breakpoints" section of chapter 5 for information to use software breakpoints.

No — The software breakpoints feature is disabled. This is specified by the default emulator configuration, so you must change this configuration item before you can use software breakpoints.

Yes — The software breakpoints feature is enabled. The emulator breaks to the monitor when an BRK instruction is executed. If the interrupt instruction is a software breakpoint, the original opcode is restored in the user program. A subsequent run or step from the instruction pointer (program counter) will execute from the breakpoint address.

If the BRK instruction is not a software breakpoint, an "undefined breakpoint" status message is displayed. To continue with program execution, you must run or step from the target program’s breakpoint interrupt vector address.

Trace Hold Tag

This configuration option specifies whether or not the analyzer traces target bus hold sequence.

Yes	The analyzer traces bus hold cycles.
No	The analyzer traces no bus hold cycles.

Load Real Address

This configuration option specifies whether the emulator should load absolute files into virtual address or real address when you use the "**M**emory, **L**oad" command. In other words, you can specify that in which address space the address location information are recorded in the absolute files.

Yes	The emulator interprets the location address information in the absolute files as real address.
No	The emulator interprets the location address information in the absolute files as virtual address.

Trace Execution Cycles

This configuration option specifies whether or not the analyzer traces instruction execution cycles.

Yes	Both exec states and bus states are captured by the emulation analyzer. You will see the disassembles of executed instructions in trace listing. Lines with disassembles indicate exec states of the instructions.
No	Only bus states are captured by the emulation analyzer. When you display trace listing, the emulator disassembles with "fetch" states, and their disassembled processor mnemonics is displayed at the "fetch" states which are the first byte of the instructions. In this mode, the analyzer can trace with time tagging or # of states counter.

Refer to the "Using the Emulator" chapter for more details of the analyzer features.

Trace Real Address

This configuration option specifies whether the analyzer should trace virtual address or real address.

Yes — The analyzer captures real address bus which is the same that the actual microprocessor outputs to.

No — The analyzer captures virtual address.

CMB Interaction?

Coordinated measurements are measurements synchronously made in multiple emulators or analyzers. Coordinated measurements can be made between HP 64700 Series emulators which communicate over the Coordinated Measurement Bus (CMB).

Multiple emulator start/stop is one type of coordinated measurement. 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.)

No — The emulator ignores the /EXECUTE and READY lines, and the READY line is not driven.

Yes — 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.

Note

CMB interaction will also be enabled when the

```
Processor, CMB, Execute
```

command is entered.

V70 Inverse Assemble

This configuration specifies inverse assembler for either 70108/70116 or 70632 microprocessor. The 70632 microprocessor has the 70108/70116 emulation mode. In this mode, the 70632 executes the instruction as 70108/70116 microprocessor's one. The emulator provides both inverse assemblers for 70108/70116 and 70632.

Yes — The 70632 inverse assembler is used when you display memory in mnemonic format.

No — The 70108/70116 inverse assembler is used when you display memory in mnemonic format.

Respond to HLDRQ

This configuration option specifies whether /HLDRQ signal from target system is accepted or ignored by the emulator.

Yes — The emulator accepts Hold Request from target system.

No — The emulator ignores Hold Request (/HLDRQ signal input) from target system.

Respond to Target NMI

This configuration option specifies whether or not the emulation processor accepts to /NMI signal generated by the target system.

Yes — The emulator accepts NMI signal generated by the target system. When the NMI signal is accepted, the emulator calls the NMI procedure as actual microprocessor.

No — The emulator ignores NMI signal from target system completely.

Target Interrupts

This configuration option specifies whether or not the emulation processor accepts to INT signal generated by the target system.

Yes	The emulator accepts INT signal generated by the target system. When the INT signal is accepted, the emulator calls the INT procedure as actual microprocessor.
No	The emulator ignores INT signal from target system completely.

Target Bus Freeze

This configuration option specifies whether BFREZ from target system is accepted or ignored by the emulator.

Yes	The emulator accepts BFREZ signal from target system.
No	The emulator ignores BFREZ signal input from target system.

Drive Background Cycles

This configuration option specifies whether or not the emulator's bus cycles are driven to your target system bus when the emulator is in background cycle. If your target system requires bus cycle activities constantly, you will need to drive the emulation bus cycles to your target system bus.

No	The emulator does not drive any bus cycles to target system bus in background operation.
Yes	The emulator drives its bus cycles to target system bus whether or not the emulator executed in the background cycles. If your target system have some circuitry which monitors bus activities, you may need to enable this configuration. related configuration .

Target Memory Access Size

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

This configuration option specifies the type of microprocessor cycles that are used by the monitor program to access target memory. When a command requests the monitor to read or write target system memory or I/O, the monitor program will look at the access mode setting to determine whether byte, halfword, or word instructions should be used.

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

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

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

Address Bits A31-A8

This configuration option specifies the location of the background monitor program. The monitor may be located on any 4K byte boundary. If the address specified is not on a 4K boundary, the 4K boundary below the address is used. The location of background monitors may be important because background cycles of the 70632 emulator are always visible to the target system. In default, the monitor is located on 00000H through 00FFFH (actual address).

This configuration does not make sense when the "Drives Background Cycles" question is answered "No".

Monitor Type

This configuration option allows you to select and use a foreground emulation monitor program. The default monitor is background monitor.

Background Specify monitor type as background monitor. When you select background monitor, you can specify the background monitor location.

Foreground Specify monitor type as foreground monitor. When you select foreground monitor, you must specify correct foreground monitor start address with next configuration question (foreground monitor address). After you completed the configuration setting, you need to load foreground monitor program to the emulator with "**M**emory, **L**oad" feature. The foreground monitor program must already assembled and linked with appropriate location specification. Refer to the *HP 64758 70632 Terminal Interface User's Guide* for more information.

Note

If you select a foreground monitor, a 4 kilobyte block is automatically mapped at the address specified by the next question.

Foreground Monitor Location?

You can relocate the monitor from the default monitor location to any 4K byte boundary. The location of the foreground monitor is important because it will occupy part of the processor address space. Foreground monitor location must not overlap the location of target system programs. The default foreground monitor location is "00000000H". When entering monitor block addresses, you must only specify addresses on 4K byte boundaries; otherwise, an invalid syntax message is displayed.

Note

Relocating the monitor causes all memory mapper terms to be removed.

Specify the real memory location of foreground monitor in the "[real address]" field. When using the foreground monitor in virtual mode, you must also specify the virtual location in the "[virtual address]" field.

Storing an Emulator Configuration

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

- Emulator configuration items.
- Key macro specifications.
- Memory map.
- Break conditions.
- Trace specification and format.
- Window specifications.

To store the current emulator configuration, select:

Config **S**tore

Enter the name of a file in which to save the configuration.

Loading an Emulator Configuration

If you want to reload a previously stored emulator configuration, 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

5

Using The Emulator

Introduction

Many of the important topics described in this chapter involve the commands or features which relate to using the emulator. The "Getting Started" and "Virtual Mode Emulation Topics" chapters shows you how to use the basic features of the 70632 emulator. This chapter describes more information or notices of the emulator.

This chapter contains the following topics.

- Register Manipulation
  - Stack Pointer and Program Status Word Modification.
  - Floating-Point Format Display or Modification
- Analyzer Topics
  - Analyzer Status Labels
  - Analyzer Trigger Condition
  - Trace Listing Disassembler
  - Execution States
  - Analyzer Data Bus Condition
  - Analyzer Clock Speed
  - Cause of Monitor Break
- Hardware Breakpoints
- Software Breakpoints
- Target Memory Access
- FPU Support
- MMU Support
- Coordinated Measurement
- Unfamiliar Prompts
- 70118/70116 Emulation Mode
- FRM Support
- Real-time Emulation Memory Access
- Virtual Address Translation
- Storing the Contents of Memory into Absolute File
- Register Names and Classes
- Foreground Monitor
- Restrictions and Considerations

Prerequisites

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

Register Manipulation

Stack Pointer Modification

In the 70632 microprocessor, one of the five privileged registers (L0SP, L1SP, L2SP, L3SP, ISP) is selected as stack pointer according to the EL and IS flags of the PSW, and the stack pointer is cached by SP. The contents of the stack pointer corresponding to the execution level are not always the same as the stack pointer (SP). The stack pointer corresponding to the execution level is updated only when the execution level is changed.

The emulation monitor is executed in execution level 0. When the emulator returns from emulation monitor to user program, for example when you issue "**P**rocessor **G**o" command, the emulator changes execution level from 0 to user program’s execution level which is determined by the IS flag and EL field in the program status word (PSW).

For this reason, in emulation monitor, the stack pointer (SP) and the stack pointer corresponding to the execution level need to have the same value. The monitor intends to keep the stack pointer (SP) and the current level stack pointer to have the same value.

When breaking into monitor, the current level stack pointer is modified to the value of SP.

If you modify registers PSW, L0SP, L1SP, L2SP, L3SP or SP in monitor, note the following.

- When you modify the EL or IS flag of the PSW, the SP is modified to the value of the stack pointer corresponding to the execution level which is determined by the EL or IS flag of the PSW you have modified.
- When you modify the stack pointer corresponding to the current execution level (L0SP, L1SP, L2SP, L3SP, ISP), the stack pointer SP is modified to the same value.
- When you modify the stack pointer SP, the stack pointer corresponding to the execution level (L0SP, L1SP, L2SP, L3SP or ISP; the one selected depending on the contents of the PSW) is modified with the same value.

Displaying/Modifying Registers In Floating-Format

You can display/modify general purpose registers (R0 through R31) in floating-point format with "**R**egister, **F**loat" command. The IEEE-754 standard data type is supported.

Display register R0 in floating-point format by selecting:

```
Register, Float, Display
```

Use the **TAB** key to select the **r0**.

Modify register R4 to the value 12345.678, by selecting:

```
Register, Float, Modify
```

Use the **TAB** key to select the **r4** and press **Enter**. Type the value **12345.678** and press **Enter**.

Analyzer Topics

Analyzer Status Qualifiers

The following are the analyzer status labels which may be used in the "**A**nalysis, **T**race, **M**odify" analyzer commands.

```
fetch           0x1xxxxxxxxxx011x     code fetch
fetchbr         0x1xxxxxxxxxx0111     code fetch after branch
read            01xxxxxxxxxxxxxxx     read
write           00xxxxxxxxxxxxxxx     write
data            0xxxxxxxxxxxx0011     data access (read/write)
io              0xxxxxxxxxxxx1011     i/o access (read/write)
exec            0xxxxxxxxxxxx0000     execution state
sdata           0xxxxxxxxxxxx0010     data access (read/write) with short path
sysbase         0xxxxxxxxxxxx0100     system base table access
trans           0xxxxxxxxxxxx0101     translation table access (read/write)
coproc          0xxxxxxxxxxxx1000     co-processor access(read/write)
shortrd         0x1xxxxxxxxxx0010     data access read with short path
shortwr         0x0xxxxxxxxxx0010     data access write with short path
iord            0x1xxxxxxxxxx1011     i/o access read
iowr            0x0xxxxxxxxxx1011     i/o access write
transrd         0x1xxxxxxxxxx0101     translation table access read
transwr         0x0xxxxxxxxxx0101     translation table access write
coprocrd        0x1xxxxxxxxxx1000     co-processor access read
coprocwr        0x0xxxxxxxxxx1000     co-processor access write
fault           0xxxxxxxxxxxx1100     machine fault acknowledge
halt            0xxxxxxxxxxxx1101     halt acknowledge
intack          0xxxxxxxxxxxx1110     interrupt acknowledge
grdacc          0xxxxxxxxxx0x0xxx     guarded memory access
wrrom           0x0xxxxxxx0xx0xxx     write to ROM
monitor         0xxxxxxxxxxx0xxxx     background monitor cycle
lock            0xxxxxxxx0xxxxxxx     bus lock
retry           00xxxxxxxxxxxxxxx     retry
hold            0xxxxxxxxxxxx0001     bus hold
```

Specifying Trigger Condition at Desired Instruction Execution

In the "Using the Analyzer" section of the "Getting Started" chapter, you used the analyzer to trace the states of the program after that the instruction located at address 10033H was executed.

As you know, the 70632 processor has the prefetch unit (PFU) to prefetch the instruction string to be executed.

If you had not specified the "exec" status label in the trigger status field, unexpected trigger would have occurred at the prefetch state of the address 10033H.

This discussion is significant when you specify the trigger condition at the execution of the instruction which follows a branch instruction like:

```
000020012@r  -              CMP.B      #00H,R2
000020016@r  -              BZ         00020000H
000020018@r  -              MOV.W      #0000000fH,R0
```

Assume that the processor executes instructions at address range 20000H through 20016H normally, and the instruction at address 20018H is executed at long intervals.

If you wish to trigger the analyzer at the execution of the address 20018H, you should specify the "exec" status label into status field.

If you would not specify the "exec" status label, the trigger will always occur at the prefetch of the address 20018H whether or not the branch condition at address 20016H is satisfied.

Disassembles In Trace Listing

As you can see disassembles in analyzer trace listing, the emulator has disassemble capability in trace listing. When the emulator disassembles instructions in stored trace information, the prefetch cycles of each instruction are required.

In the "Using the Analyzer" section of the "Getting Started" chapter, you configured the analyzer to trace the states of the program after the read states from the **Command_Input** byte and 0xxxxxx42H in the data field.

When you displayed the results of analyzer trace, some lines which include "**No fetch cycle found**" messages were displayed. Each line was instruction execution cycle at the address in the left side of the line. However, the disassembles of these instructions were not displayed because the prefetch states for the instructions were not stored by the analyzer.

The trigger position was at the start of the trace listing, because you wished to trace the states of the program after the triggered state.

Note that the "**No fetch cycle found**" messages may be displayed around line 0 (trigger point) when the "Trigger Position" is "Start".

To display complete disassembles in trace listing, you should modify location of trigger state in trace list, referred to as the "trigger position".

Execution States Location in Trace Listing

The emulation analyzer stores execution states of the program in addition to actual bus cycles, if configuration option "Trace execution cycle" is answered "**yes**".

When the processor executes an instruction, the execution state of the instruction is generated before its bus state(s) by the execution of the instruction.

However, it is possible that the execution states are inserted after or between the actual bus states of these activities, since the clock rate of bus sampling is high-speed.

The following trace listing shows the example that the execution state, numbered 64, falls behind its bus activity.

```
61   00003004  00001e05     00001e05H trans table  read           *********
62   00003004  00001e85     00001e85H trans table  write          *********
63   00001004  00000002     00000002H data  read                  *********
64   00005043  00000002   MOV.W     00001004H,R0                  *********
65   0000504a  00000002   MOV.W     #00001008H,R1                 *********
66   00005060  2da20801               fetch                       *********
```

Specifying Data For Trigger Condition or Store Condition

The analyzer captures the data bus of the 70632 microprocessor. When you specify a data in the analyzer trigger condition or store condition, the ways of the analyzer data specifications differ according to the data size and the address. Suppose that you wish to trigger the analyzer when the processor accesses to the byte data 41H in a 1000H address. You should not specify the data field of the trigger condition like "41H".

The data condition will be considered as 00000041H. The bit 31 through bit 8 of data bus is unpredictable because of the byte data. You will unable to trigger as you desire. You should have specified the data field with "0xxxxxx41h".

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

When the address that you want to trigger is not a multiple of 4, the data bus specification is different from the above. If you trigger the analyzer at the address 1001H instead of the address 1000H, the data 41H will be output to the bit 7 through bit 4 of the data bus. You should specify the data field with "0xxxx41xxh".

Analyzer Clock Speed

The emulation analyzer can capture both the exec states and bus states.

Bus states show actual processor's bus activity.

Exec states indicate the address of the first byte of an executed opcode. Only the address and processor status fields are valid during these states.

The analyzer has a counter which allows to count either time or occurrence of bus states. Tracing both bus cycles and exec states, effectively doubles the clock rate to the analyzer.

By default, the analyzer time counter is turned off because the analyzer time counter cannot be used at high-speed clock rate. If it is desired to use the analyzer counter, configure the analyzer to trace only bus cycles. The clock speed can be effectively halved if execution states are NOT traced. To do this, you should answer "no" at the "Trace execution cycle" question. Refer to the "Trace Execution Cycle" section of the "Configuring the Emulator" chapter for more information.

Finding Out the Cause of a Monitor Break

If the emulator breaks into monitor unwillingly, you can examine the cause of the break by using the analyzer. When you issue the following commands, you can capture the behavior of the program just before the monitor break.

Reset the trigger condition, select:

```
Analysis, Trace, Reset
```

Specify the trigger condition that the analyzer is **never** triggered.

```
Analysis, Trace, Modify
```

Use the right arrow key to move to the “Trigger on” field. Use the **TAB** key to select "**no state**" , and press **Enter**. To save the trace specification, move the cursor to the lower right corner, and press **Enter**.

Start the trace.

```
Analysis, Begin
```

After starting your program, the unexpected break will occur. To show the cause of the break, stop the trace and display the trace listing.

```
Analysis, Halt
Analysis, Display
```

Specify the trace lines to be displayed as -19 through 0.

The trace listing displays will show the cause of the break. If you cannot find the cause of the break, display the previous states. If the trace listing does not include the fundamental problem, you need to change the trigger condition to capture the problem, and then restart the trace and the program.

This is also useful to detect the causes other than monitor breaks like a processor halt.

Hardware Breakpoints

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

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, Trigger
```

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 which shows that the analyzer is driving TRIG1.

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.

The trigger configuration display is shown in figure 5-1.

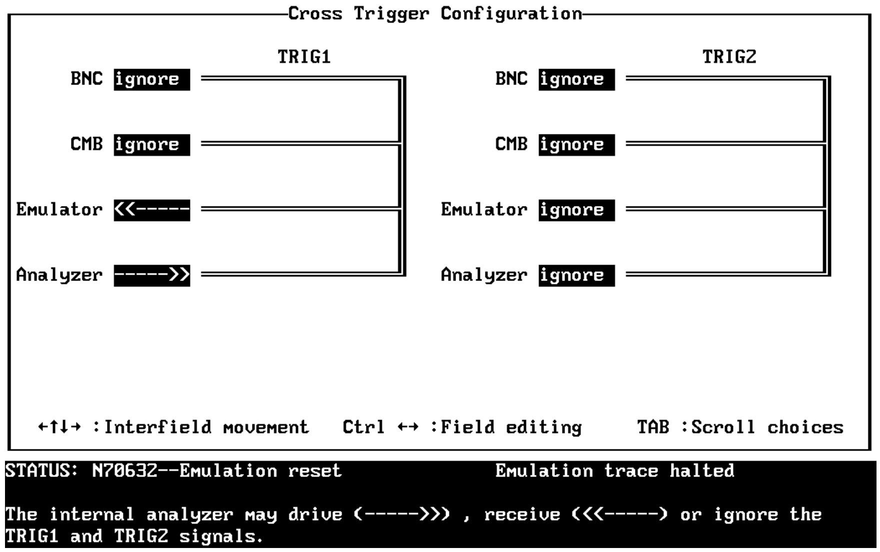


Figure 5-1. Cross Trigger Configuration

Software Breakpoints

Software breakpoints are realized by the 70632 BRK instruction. When you define or enable a software breakpoint, the emulator will replace the opcode at the software breakpoint address with a breakpoint interrupt instruction (BRK). When the BRK instruction is executed, the emulator breaks into monitor and compares the address that the break occurred.

If the address is defined as software breakpoint, the emulator displays that the breakpoint hit. The emulator disable the breakpoint and replace the BRK instruction with the original opcode.

If the BRK interrupt was generated by a BRK interrupt instruction in the target system, execution still breaks to the monitor, and an "undefined breakpoint" status message is displayed. To continue with program execution, you must run or step from the target program’s breakpoint interrupt vector address.

There are some attentions when you use the software breakpoint features.

1. Software Breakpoints Should Be Set at only Locations which Contain Instruction Opcodes

   You must only set software breakpoints at memory locations which contain instruction opcodes (not operands or data). If a software breakpoint is set at a memory location which is not an instruction opcode, the software breakpoint instruction will never be executed and the break will never occur.

2. Software Breakpoints Should Be Set When The Emulator Is Running In Monitor

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

3. Software breakpoints cannot be set in target ROM

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

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

4. BRK instruction vector must be set up

   You must define the 70632 break-point instruction trap vector to point to an address which is allowed instruction fetch; typically in the program code area.

   When a software breakpoint occurred, the emulator breaks into the monitor after the BRK instruction has been executed. However the instruction which is pointed by the BRK instruction vector is never executed.

   If you didn’t set up the vector and a software break has occurred, an access to the address pointed by the vector may drive the emulator into unpredictable state. The 70632 break-point instruction vector is defined in the 70632 system base table. The vector is located at 0XXXXXX34H; where "XXXXXX" is determined by the contents of the privilege register SBR (defaults is "000000").

This table location depends on the content of 70632 SBR register.

5. More Three Words Of The Stack Area Must Be Prepared

When the BRK instruction is executed, the emulator stores the exception information to stack as the same as the 70632 microprocessor does.

So, you should prepare more three words (12 bytes) for stack in addition. The stack, which is used when the breakpoint occurs, is normally the level 0 stack which is pointed by L0SP. When the software breakpoint occurs, if the program uses interrupt stack, the three words of the interrupt stack pointed by ISP is modified by the emulator instead of level 0 stack.

6. Software Breakpoint Manipulation In Virtual Mode

When you enable disable or remove a software breakpoint which you have set by using virtual address, you must issue its command in same virtual space when you have set.

The notices related to software breakpoint manipulation in virtual mode are described in chapter 3.

Target Memory Access

Commands Not Allowed when Real-Time Mode is Enabled

When emulator execution is restricted to real-time and the emulator is running in user code, the system refuses all commands that require access to processor registers or target system memory or I/O. The following commands are not allowed when runs are restricted to real-time:

- Register display/modification (except for XMMU class registers).
- Target system memory display/modification. Because the emulator contains dual-port emulation memory, commands which access emulation memory do not require breaks and are allowed while runs are restricted to real-time.
- I/O display/modification.
- Step.
- Area Table Entry display (which is in target system memory).
- Page Table Entry display (when the PTE or the dependent ATE is/are in target system memory).
- Any other commands with virtual address designation (which cause target system memory accesses for address translation). When you specifies virtual addresses in commands, the emulator will refer to the address translation tables to translate the virtual addresses to the corresponded real addresses. If the address translation tables which are required to translate the specified virtual addresses is in target system memory, the address translation will be failed.

If the real-time mode is enabled, these resources can only be displayed or modified while running in the monitor.

Breaking out of Real-Time Execution

The only commands which are allowed to break real-time execution are:

```
"Processor, Reset", "Processor, Go",
"Processor, Break"
```

FPU Support

The emulation analyzer can capture co-processor cycles. FPU register display and modification are not supported.

There are following considerations to display co-processor mnemonics in trace or memory display.

FMOVCR instruction

FMOVCR instruction will be displayed as follows:

```
FMOVCTW     instead of FMOVCR   OP1, FCTW
FMOVPTW     instead of FMOVCR   OP1, FPTW
FMOVSTW     instead of FMOVCR   OP1, FSTW
```

Instructions with no operand

Dummy operands are displayed when dis-assembling instructions without any operand. As a sign, "**#**" is displayed just after Opcode mnemonics as follows.

```
0000fe86a@r -               FRPUSH #    FR0,FR0
```

Two "FR0"s are dummy operands. The following instructions relate this.

```
FADD3M.S FADD3M.L FADD4M.S FADD4M.L
FSUB3M.S FSUB3M.L FSUB4M.S FSUB4M.L
FMUL3M.S FMUL3M.L FMUL4M.S FMUL4M.L
FRPUSH   FRPOP    FAFFECT
```

Instructions with one operand

Dummy operand is displayed when dis-assembling instructions with only one operand. As a sign, "*****" is displayed just after Opcode mnemonics as follows.

```
0000fe87a@r -               FRREL *     /00000100H,FR0
```

The "FR0" is a dummy operand. The following instructions relate this.

```
FIPV.S   FIPV.L   FRPINC   FRREL
```

MMU Support

The **ate** and **pte** Terminal Interface commands allow you to display Area Table Entry and Page Table Entry for an address you specified in the commands. These commands are useful to examine in which address space the program are executed, and detect the address translation error of the program. Examples of these command usages are described in chapter 3. These command syntax are described in the *HP 64758 70632 Emulator Terminal Interface User's Guide.*

Making Coordinated Measurements

Coordinated measurements are measurements synchronously made in multiple emulators or analyzers. Coordinated measurements can be made between HP 64700 Series emulators which 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 chapter will describe coordinated measurements made from the PC Interface which involve the emulator. These types of coordinated measurements are:

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

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 means 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 are

required to break to background upon reception of a false READY signal and will not return to foreground until this line is known to be in a true state.

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

Unfamiliar Status

When you are using the emulator, one of the following message is displayed in the status line normally.

```
N70632--Emulation reset
N70632--Running user program
N70632--Running in monitor
```

If your target system has a defect or you does not configure the emulator appropriately, the following prompts may be displayed.

- N70632--Waiting for ready
- N70632--Halted

Waiting for Target Ready

The status "Waiting for ready" indicates that the emulator is waiting for target ready signal.

If you map the unused memory locations as target memory and your program accesses to these locations by a defect (in case of in-circuit, also if a target memory is accessed by an emulation command), the emulator is waiting for an impossible ready signal infinitely because the /READY signal is internally pulled up. When you encounter this status, the emulator cannot break into monitor. All you can do is to reset the processor.

If you are using the emulator in in-circuit mode, the reason is that the emulator intends to access to a memory location for which your target system does not generate ready signal.

If you are using the emulator in out-of-circuit mode, the reason is that the emulator intends to access to a target memory location by your program. To prevent this, all of memory locations, which are not used, should be mapped as guarded memory. When you direct the emulator to access a target memory location, the emulator will return an error message.

Halt or Machine Fault

The status "Halted" indicates that the emulator is halted or in machine fault.

In case of machine fault, all you can do will be to reset the processor because the emulator cannot break into monitor.

One of the causes is the exception by a address translation failure. In this case, one of the solution is to use the analyzer. The analyzer will capture states which causes the emulator to halt. Refer to the "Finding out the Cause of a Monitor Break" description of the "Analyzer Topics" section in this chapter, for the analyzer configuration.

70108/70116 Emulation Mode

The 70632 microprocessor has the 70108/70116 emulation mode. In this mode, the 70632 executes instructions as 70108/70116 microprocessor's ones.

The emulator provides the following functions for both 70108/70116 and 70632.

- Display memory contents in processor mnemonic format.
- Single-stepping
- Analyzer trace

Displaying Memory In 70108/70116 Mnemonic Format

The emulator can display contents of memory in mnemonic format for both 70108/70116 and 70632. The emulator provides both inverse assemblers for 70108/70116 and 70632. You can select one of the inverse assemblers to display memory contents by using configuration option "**V70 inverse assemble** ".

To display memory contents in 70108/70116 mnemonic, change the disassembler by answering:

no

To display memory contents in 70632 mnemonic (default), answer:

yes

Single-stepping

You can also single-step the instructions in the 70108/70116 emulation mode. When you single-step the instructions, mnemonics of the executed instruction is displayed in corresponded processor's ones.

However, when you modify the contents of PSW to change the mode with the "**R**egister, **M**odify" command, a mnemonic of next one instruction is displayed in wrong processor mnemonic.

Tracing States In Both Mode

You can also trace the bus states and exec states in the 70108/70116 emulation mode. When tracing the execution of the program, mnemonics of the executed instructions are included in trace listing. The corresponded processor mnemonics are displayed automatically.

Real-time Emulation Memory Access

The dual-port memory for the emulation memory allows emulation displays and modifications of emulation memory without breaking the processor into the monitor during emulation.

This is referred to as the Real-time Emulation Memory Access capability.

If you issue emulation memory display/modification command while the emulation program is running, HP 64700 emulation controller, not the emulation processor, intends to access the dual-port emulation memory with the cycle-stealing method. The emulation memory accesses without breaking the processor into the monitor are accomplished for this reason.

When cycle-stealing to access to the emulation memory, the emulation controller watches for idle cycles in the 70632 bus cycles. When the idle cycles are found, the emulation controller can access to the emulation memory at the interval of the 70632 bus cycles with cycle-stealing.

However, in case that the emulation controller cannot find any idle cycles, the emulation controller holds the 70632 bus cycles (not but breaking into the monitor) in order to access to the emulation memory.

If your target system inserts some wait states to access to memory, no idle cycle may be generated. It is depended on WHAT instructions are executed when the emulation memory access command is issued, or HOW much wait states are inserted.

When there is no idle cycle within 160 mS, the hold request will be generated to the emulation processor except that the emulator is held, bus-frozen or reset.

Virtual Address Translation

When you specify virtual addresses in emulation commands, the emulator intends to translate these virtual addresses to actual memory addresses in order to manipulate contents of these memory locations.

For the address translation, the 70632 microprocessor uses its area table register pairs, which define a virtual address space. Similarly, the emulator requires values which corresponds to the 70632 area table register pairs.

Using the Caches of Area Table Register Pairs

The emulator has the caches of the area table register pairs, which allow the emulator to refer the corresponded area table for the address translations even if the emulator cannot to or is not allowed to break into the monitor.

Each time the emulator breaks into monitor, the caches are updated by the contents of the 70632 area table register pairs.

By default, the emulator uses the caches to translate the addresses which you specify in emulation commands. The caches contain the base addresses and the lengths of the area tables as the same as the 70632 area table register pairs. The emulator refers to the corresponded area table and page table by using the caches.

If the emulator is restricted to real-time runs by the "**Real-time mode** " configuration option, the caches will keep the values while you do not break the emulator into the monitor intentionally. Only when you issue "**P**rocessor, **B**reak", "**P**rocessor, **S**tep" or "**P**rocessor, **R**eset" command or a break condition (such as software breakpoint) is satisfied, the caches are updated.

If the emulator is not restricted to real-time runs (default), the caches are updated by the contents of the area table register pairs every time the emulator breaks into monitor whether with or without your intention. When you issue commands with virtual addresses, the emulator breaks into the monitor to access the area table register if possible. As the result, the emulator will use the current virtual address space for address translations.

In the both cases, when the emulator cannot break into monitor, for example the processor is reset, the emulator uses the caches for the address translation.

Specifying Virtual Address Space

When you specify virtual addresses in emulation commands, the emulator translates the virtual address to corresponded real addresses. The translated real addresses depends on a virtual address space. The virtual address space can be defined by the values of area table base and length for each section. In 70632 microprocessor, these informations are stored in its area table register pairs.

In case that the caches mentioned above are used for the address translation, it is difficult to specify an virtual address in your desirable virtual address space during running user program. If your program performs in multiple virtual space, you may want to specify a virtual address space for address translations in order to watch for the execution of a certain task.

This is accomplished by using the XMMU function. The XMMU function allows you to fix a virtual address space for address translations. The emulator has the optional XMMU class registers. These registers consist of eight XMMU register pairs and one XMMU mode register. The XMMU register pairs correspond to the actual 70632 area table register pairs. You can specify a virtual address space by modifying the XMMU class registers. The format of the XMMU class registers is the same as the 70632 actual area table register pairs. The XMMU class registers also include the XMMU mode register (mmumod), which determines whether the caches or the contents of the XMMU register pairs are used for address translations. By default, the caches are selected.

If you activate the XMMU function, the emulator uses the contents of the XMMU register pairs for address translations whether or not the emulator is restricted to real-time runs.

The XMMU class registers consist of the following registers.

```
XMMU class registers          corresponded actual registers
xatbr0                        atbr0
xatlr0                        atlr0
xatbr1                        atbr1
xatlr1                        atlr1
xatbr2                        atbr2
xatlr2                        atlr2
xatbr3                        atbr3
xatlr3                        atlr3

mmumod                        --None--
```

To specify a virtual address space which is used for address translations, modify the contents of the XMMU register pairs corresponded to the area table registers by using the "**R**egister, **M**odify" command or the **cpmmu** (copy current virtual address space to XMMU registers) Terminal Interface command. See also the "Using the XMMU function" section of chapter 3. For the "**cpmmu**" command syntax, refer to the *HP 64758 70632 Terminal Interface User's Guide*.

After you have modify the contents of the XMMU register pairs, activate the XMMU function by changing the contents of XMMU mode register (mmumod) to the value 1.

```
Register, Modify
```

Select **mmumod** for the register name, and type **1** for the value to be modified.

To use the caches of the area table register pairs for address translations, modify mmumod register to 0 (default).

```
Register, Modify
```

Select **mmumod** for the register name, and type **0** for the value to be modified.

Besides by using the "**R**egister **M**odify" command, the **mmumod** register is reset when the emulator breaks into monitor in the following causes.

- Break by software breakpoint
- Break by single-stepping
- Break by writing to ROM
- Break by access to guarded memory

In these case, the **mmumod** register is reset to "0". As the result, the address translation of the virtual address in a command uses the actual area table register pairs.

Storing Memory Contents to an Absolute File

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

The first character of the absolute file name must be a letter. You can name the absolute file with a total of 8 alphanumeric characters, and optionally, you can include an extension of up to 3 alphanumeric characters. If the file is stored in HP 64000 format, its extension must be ".X".

Caution

The "Memory Store" command writes over an existing file if it has the same name that is specified with the command. You may wish to verify beforehand that the specified filename does not already exist.

Register Names and Classes

The following register names and classes may be used with the "**R**egister, **D**isplay/**M**odify" commands.

BASIC Class

Register Name	Description
pc psw r0 r1 r2 r3 r4 r5 r6 r7 r8 r9 r10 r11 r12 r13 r14 r15 r16 r17 r18 r19 r20 r21 r22 r23 r24 r25 r26 r27 r28 r29 r30 r31 ap fp sp	All basic registers. The **ap** and **r29**, **fp** and **r30**, **sp** and **r31** have same values because of only difference of their register mnemonics.

priv (Privilege registers)

isp l0sp l1sp l2sp
l3sp sbr tr sycw
tkcw pir psw2

mmu (MMU registers)

atbr0 atlr0 atbr1 atlr1 atbr2 atlr2 atbr3 atlr3	Area Table Register Pairs

debg (Debug registers)

trmod adtr0 adtr1
adtmr0 adtmr1

xmmu (XMMU function registers)

mmumod **xatbr0 xatlr0** **xatbr1 xatlr1** **xatbr2 xatlr2** **xatbr3 xatlr3**	XMMU function registers. These registers are **not actual 70632 registers**. Refer to the XMMU function section of the "Using the Emulator" chapter for the detail.

Foreground Monitor

This section describe the PC Interface specific notification for using the foreground monitor. Refer to the *HP 64758 70632 Emulator Terminal Interface User's Guide*, for more information of the foreground monitor.

Foreground Monitor Configuration

Before loading the foreground monitor into the emulator, you must answer some configuration option questions.

Select:

```
Config, General
```

Move the cursor to "Monitor type" field, select **foreground**, and press **Enter**.

In the next field, you must specify the foreground monitor location (real address), type the address of the monitor, and press **Enter**. The address must be specified in 4 Kbyte boundary. The monitor location is automatically mapped as emulation ram (eram).

In case of virtual mode application, you must also specify the virtual location of the monitor. Specify the address in the "[virtual address]" field.

Loading the Monitor into Emulator

To load the monitor, select:

```
Memory, Load
```

Specify the file format and press **Enter**.

Now the "Memory Type" field is high-lighted, select "**F_Monitor**" to load the foreground monitor, and press **Enter**.

Specify the file name and press **Enter**.

The emulation monitor will have been loaded into emulator.

After you have loaded the monitor program, map the memory and load your program.

Restrictions and Considerations

When the microprocessor accesses data which are not aligned, the microprocessor generates more than twice memory access cycles.
If the microprocessor accepts interrupt while microprocessor reads the data which are not aligned, the microprocessor stop accessing the data and generates invalid memory write cycle.
But, memory is not changed because bus enable signals(BS0-BS3) are inactive, and stopped memory read cycles are reexecuted after interrupt routine.

If you specify that the emulator break into the monitor upon attempts to write to memory mapped as ROM and if microproccessor generates invalid memory write cycle described above in user’s program, the emulator break into the monitor.

Notes

6

In-Circuit Emulation Topics

Introduction

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

This chapter will:

- Describe the issues concerning the installation of the emulator probe into target systems.
- Show you how to install the emulator probe.
- Show you how to use features related to in-circuit emulation.

Prerequisites

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

Installing the Emulator Probe into a Target System

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

Caution

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

Caution

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

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

Caution **Protect your target system CMOS components.** If you target system contains any CMOS components, turn ON the target system first, then turn ON the emulator. Likewise, turn OFF your emulator first, then turn OFF the target system.

Pin Protector

The target system probe has a pin protector that prevents damage to the prove when inserting and removing the probe from the target system microprocessor socket. Do not use the probe without a pin protector installed. If the target system probe is installed on a densely populated circuit board, there may not be enough room to accommodate the plastic shoulders of the probe socket. If this occurs, another pin protector may be stacked onto the existing pin protector.

Conductive Pin Guard

HP emulators are shipped with a conductive plastic or conductive foam pin guard over the target system probe pins. This guard is designed to prevent impact damage to the pins and should be left in place while you are not using the emulator. However, when you do use the emulator, either for normal emulation tasks, or to run performance verification on the emulator, you must remove this conductive pin guard to avoid intermittent failures due to the target system probe lines being shorted together.

Caution

Always use the pin protectors and guards as described above. Failure to use these devices may result in damage to the target system probe pins. Replacing the target system probe is expensive; the entire probe and cable assembly must be replaced because of the wiring technology employed.

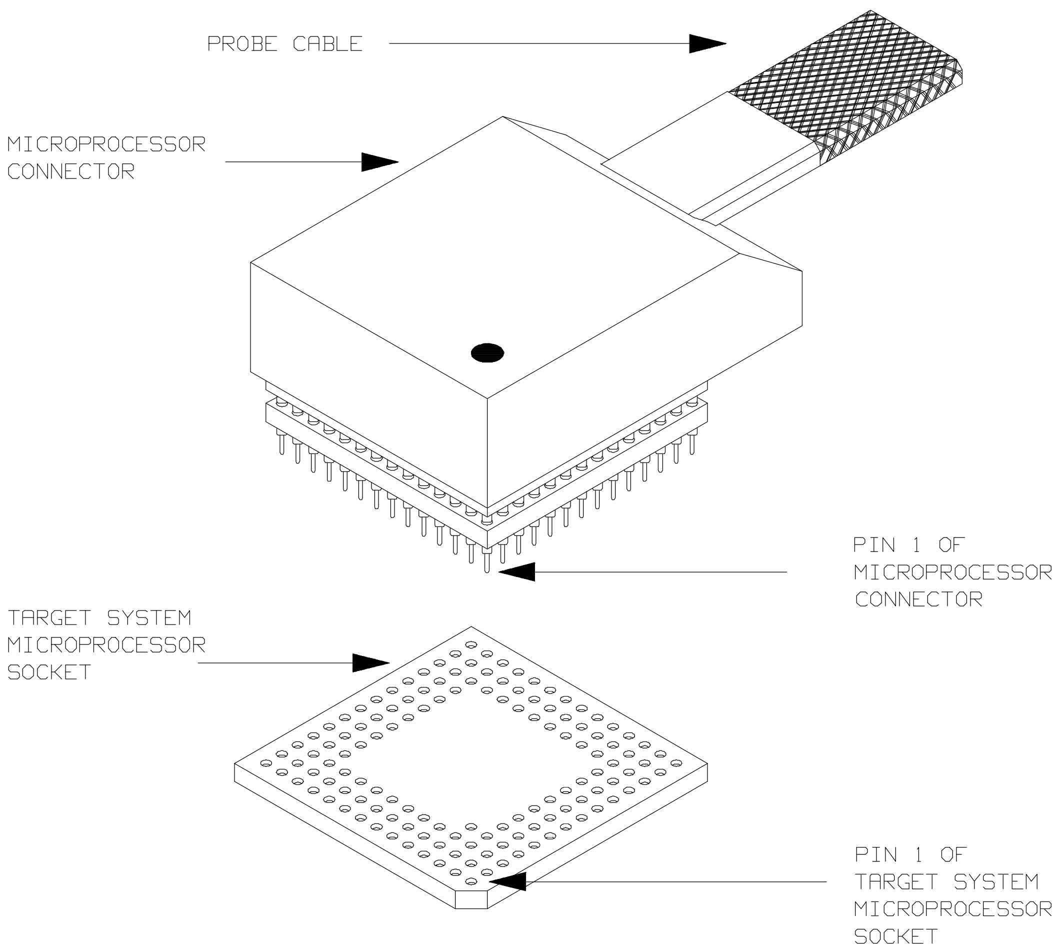


Figure 6-1. Installing Emulation Probe Into PGA Socket

Installing the Target System Probe

1. Remove the 70632 microprocessor from the target system socket. Note the location of pin 1 on the processor and on the target system socket.

2. Store the microprocessor in a protected environment (such as antistatic foam).

3. Install the target system probe into the target system microprocessor socket. Remember to use the pin protector!

In-Circuit Configuration Options

The 70632 emulator provides configuration options for the following in-circuit emulation issues. Refer to the "Configuring the Emulator" chapter for the configuration.

Selecting the Emulator Clock Source

The default emulator configuration selects the internal 20 MHz clock as the emulator clock source. You can configure the emulator to select an external target system clock source in the range of 8-20 MHz.

Driving Background Cycles to the Target System

You can choose whether emulator bus cycles are driven to your target system bus when the emulator is in background cycle. If your target system requires bus cycle activities constantly, such as /BCYST, will need to drive the emulation bus cycles to your target system bus. By default, no bus cycles are driven to the target system in background operation.

Selecting Memory Block during Background Cycles

You can select the value of the 70632 address bus which should be driven to your target system. Pin A31 through A8 of the address bus is configurable. This configuration is meaningful when the "Driving Background Cycles to Target System" configuration mentioned above is activated.

Allowing /HLDRQ Signal from Target System

You can specify whether the emulator accepts or ignores the /HLDRQ signal from your target system. By default, the emulator accepts the /HLDRQ signal from the target system.

Allowing BFREZ Signal from Target System

You can specify whether the emulator accepts or ignores the BFREZ signal from your target system. By default, the emulator accepts the BFREZ signal from the target system.

Allowing INT Signal from Target System

You can specify whether the emulator accepts or ignores the INT signal from your target system. By default, the emulator accepts the INT signal from the target system.

Allowing /NMI Signal from Target System

You can specify whether the emulator accepts or ignores the /NMI signal from your target system. By default, the emulator accepts the /NMI signal from the target system.

Allowing the Target System to Insert Wait States

High-speed emulation memory provides no-wait-state operation. However, the emulator may optionally respond to the target system /READY, /BERR, RT/EP lines while emulation memory is being accessed.

You can specify whether the emulation memory accesses are honored by these target system signals or not, in a memory mapping term. When you map emulation memory, if you would like to cause the emulation memory to honor these target system signals, use the "**eram_lock**" or "**erom_lock**" attribute for emulation memory type.

When the ready relationship is locked to the target system by using "**eram_lock**" or "**erom_lock**" attribute, the emulation memory accesses honor /READY, /BERR, RT/EP signals from the target system (wait states or retry cycles are inserted if requested).

If you do not specify the "**lock**" attribute, the ready relationship is not locked to the target system, and the emulation memory accesses ignore these signals from the target system (no wait states are inserted).

The Usage of I/O Command

The emulator has "**P**rocessor, **I**/O" command, you can manipulate an I/O address by using this command. You can specify an I/O address in either virtual or real address space as well as the "**M**emory" command.

There are two I/O spaces according to methods for accessing to I/O in the 70632 microprocessor.

The first I/O space can be accessed by using an IN/OUT instruction. In this section, this I/O space is referred as "Isolated I/O space" distinguish from Memory Mapped I/O described below.

The second I/O space can be accessed by simply reading from or writing to the memory. The I/O space can be mapped to the virtual address space and known as Memory Mapped I/O.

How to Access an Isolated I/O space

If you would like to manipulate an Isolated I/O space which is accessed by using an IN/OUT instruction of the microprocessor, designate the I/O address in real address.

How to Access a Memory Mapped I/O space

If you would like to manipulate a Memory Mapped I/O space which is accessed by reading from or writing to a memory. designate the I/O address in virtual address. The I/O mapped bit of the page table entry which includes the I/O address must be set to 1, in other word, the address is mapped as I/O.

Notes

A

File Format Readers

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, it may be necessary to execute it as a separate process if there is not enough memory on your personal 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 Accomplishes

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 which 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 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  012345678
global_symbol2  056789ABC
...
global_symbolN  09ABCDEF0
|module_name1|# 1234         76543210
|module_name1|local_symbol1  0CBA98765
|module_name1|local_symbol2  087654321
...
|module_name1|local_symbolN  0FEDCBA98
```

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

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

Note

If your emulator can store symbols internally, symbols will appear in disassembly. When the line number symbol is displayed in the emulator, it appears in brackets. Therefore, the symbol "MODNAME: line 345" will be displayed as "MODNAME:[345]" in mnemonic memory and trace list displays.

The space preceding module names is required. Although formatted for readability here, a single tab separates symbol and address.

The local symbols are scoped. This means that to access a variable named "count" in a source file module named "main.c," you would enter "MAIN.C:count" as shown below.

Table A-1. How to Access Variables (HP64000 Format)

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

You access line number symbols by entering the following on one line in the order shown:

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

For example:

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

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. The PATH 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:

```
C:\HP64700\BIN\RHP64000 [-q] <filename>
```

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

<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 up the PC Interface.

2. Select "**M**emory **L**oad." The memory load menu will appear.

3. Specify the file format as "HP64000." This will appear as the default file format.

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

Using the HP 64000 file that you specify (TESTFILE.L, for example), the PC Interface performs 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 only done 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.

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 occurs, you will need to exit the PC Interface and execute the program at the MS-DOS command prompt to create the files that are downloaded to the emulator.

Including RHP64000 in a Make File

You may wish to incorporate the “RHP64000” process as the last step in your “make file,” as a step in your construction process, 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, loading an HP 64000 file will automatically create them.

Using the NEC COFF Reader

The 70632 PC Interface provides with the NEC COFF Reader.

The Reader converts an absolute file into two files that are usable with the HP 64758 emulator. This means that you can use those available language tools to create absolute files, then load those files into the emulator using the 70632 PC Interface. The Reader can operate from within the PC Interface or as a separate process. When operating the Reader, it may be necessary to execute it as a separate process if there is not enough memory on your personal computer to operate the PC Interface and Reader simultaneously. You can also operate the reader as part of a "make file".

What the NEC COFF Reader Accomplishes

Using absolute file in the form "<file>.<ext>", the NEC COFF Reader will produce two new files, an "absolute" file and an ASCII symbol file, that will be used by the 70632 PC Interface. These new files are named: "<file>.hpa" and "<file>.hps".

The Absolute File

During execution of the NEC COFF 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 NEC COFF Reader contains global symbols, module names, local symbols, and, when using applicable development tools such as a "C" Compiler, program line number. 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 absolute file. The NEC COFF Reader will only convert symbol information that is present in the input absolute file.

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

```
 module_name1
 module_name2
 ...
module_nameN
global_symbol1  01001234
global_symbol2 01005678
...
global_symbolN  0100ABCD
|module_name|# 1234        02000872
|module_name|local_symbol1 02000653
|module_name|local_symbol2  02000872
...
|module name|local_symbolN  02000986
```

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

The space preceding module names is required. Although formatted for readability here, a single tab separates symbol and address.

The local symbols are scoped. This means that to access a variable named “count” in a source file module named “main.c,” you would enter “MAIN:count.” See table A-2.

Table A-2. How to Access Variables (NEC COFF Format)

Module Name	Variable Name	You Enter:
main	count	main:count
main	line number 23	main: line 23

Line numbers will appear similar to a local symbol except that "local_symbolX" will be replaced by "#NNNNN" where NNNNN is a five digit decimal number.

Note

When the line number symbol is displayed in the emulator, it appears in brackets. Therefore, the symbol "MODNAME:# 345" will be displayed as "MODNAME:[345]" in mnemonic memory and trace list displays.

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 NEC COFF Reader Program

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

Using the NEC COFF Reader from MS-DOS

The command names for the NEC COFF Reader are shown below.

```
RDNEC70.EXE
```

To execute the NEC COFF Reader from the command line, for example...

ENTER: **`RDNEC70 [-q] [-u] [-r] [-a]`** `<filename>`

-q specifies the "quiet" mode. This option suppress the display of messages.

-u defeats removal of a leading underscore in the symbol name (for example, "_symbol"). When used, a symbol name containing a leading underscore will be left alone.

-r generates load address information in real address. If this option is not specified, the load address is generated in virtual. The HP 64758 emulator can load a program in real address or virtual address. It is determined by configuration option "Object file address attribute". If you want to load a program in real address, use this option. In case of real mode application, this option is senseless because the address is the same between real address and virtual address.

-a adds address attributes to the symbol file (.HPS). You can add address attributes to symbols. The address attributes are determined by the load address attribute specified by the above option. If load address attribute is real, "@r" suffix is added to address. If load address attribute is virtual, "@v" suffix is added to address. In case of virtual mode application, this option should be specified for accurate manipulation of symbols.

<filename> is the same of the file containing the absolute program. You can include an extension in the file name.

Using the NEC COFF Reader from the PC Interface

The 70632 PC Interface has a file format option under the "**M**emory **L**oad" command.

For example, after you select NEC COFF as the file format, the NEC COFF Reader will operate on the file you specify. After this completes successfully, the 70632 PC Interface will accept the absolute file and symbol files produced by the NEC COFF Reader.

To use the NEC COFF Reader from the PC Interface:

1. Start up the 70632 PC Interface.

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

3. Specify the file format as "NEC COFF". This will appear as the default file format.

4. Specify a file in NEC COFF format ("TESTFILE.X", for example). The file extension can be something other than ".X", but ".HPA" or ".HPS" cannot be used.

The PC Interface performs the following:

- It checks to see if 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 NEC COFF 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 NEC COFF file creation date/time, the NEC COFF Reader recreates them. The new absolute file, TESTFILE.HPA, is then loaded into emulator.

- If TESTFILE.HPS and TESTFILE.HPA already exist but the dates and times are earlier than the creation date and time for the NEC COFF file, the NEC COFF Reader will not recreate

TESTFILE.HPA. The current absolute file, TESTFILE.HPA, is then loaded into emulator.

Note

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

When the Reader operates on a file, a status message will be displayed indicating that it is reading an absolute file. When the 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. The other options (-u , -r and -a) are not in effect by default. If you wish to use these options, you must execute the Reader from the MS-DOS prompt.

If the NEC COFF 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 used. If this condition occurs, you will need to exit the PC Interface and execute that are downloaded to the emulator.

Note

If you use the HP 64879 assembler/linker, specify module name in your source file by using ".file" directive for the local symbols.

Including RDNEC70 in a Make File

You may want to incorporate the “RDNEC70” 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 “-.HPA” and “-.HPS” files are not current, loading an NEC COFF file will automatically create them.

Notes

Index

Notes