HP 64758

70632 Emulator Softkey Interface

User's Guide

HP Part No. 64758-97006
Printed in U.S.A.
March, 1993

Edition 2

Notice

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

Printing History

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

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

Edition 1	64758-97002, August 1990
Edition 2	**64758-97006, April 1993**

Using This manual

This manual introduces you to the HP 64758G/H 70632 Emulator as used with the Softkey 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 Softkey Interface command and option. See the *Softkey Interface 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 **Using the Foreground Monitor.** This appendix describes the advantages and disadvantages of foreground and background monitors and how to use foreground monitors.

Appendix B **Using the Format Convertor.** This appendix describes the usage of the file format converter.

Conventions

Example commands throughout the manual use the following conventions:

bold	Commands, options, and parts of command syntax.
bold italic	Commands, options, and parts of command syntax which may be entered by pressing softkeys.
normal	User specified parts of a command.
$	Represents the HP-UX prompt. Commands which follow the "$" are entered at the HP-UX prompt.
<RETURN>	The carriage return key.

Notes

Contents

5 Using The Emulator

6 In-Circuit Emulation Topics

Illustrations

1

Introduction to the 70632 Emulator

Introduction

The topics in this 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.

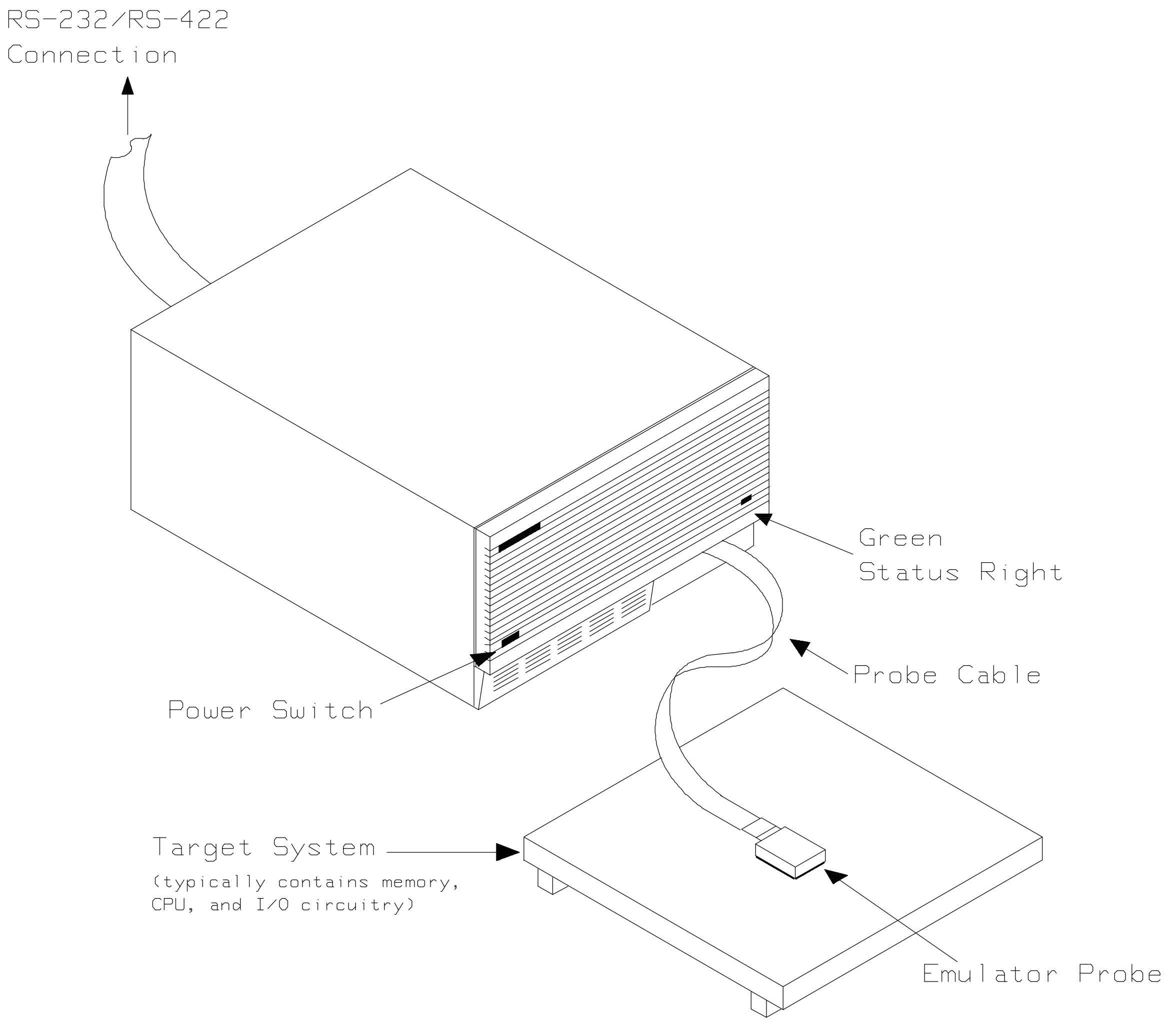

Figure 1-1. HP 64758 Emulator for the 70632

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 designed to familiarize you with the use of the HP 64758G/H 70632 emulator with the Softkey 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 example.

This chapter will show you how to:

- Start up the Softkey Interface.
- 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 show you how to do this.

2. Installed the Softkey Interface software on your computer. Refer to the *HP 64700 Series Installation/Servicel* manual for instructions on installing software.

3. In addition,you should read and understand the concepts of emulation presented in the *Concepts of Emulation and Analysis* manual. The *Installation/Service* manualalso covers HP 64700 system architecture. A brief understanding of these concepts may help avoid questions later.

   You should read the *Softkey Interface Reference* manual to learn how to use the Softkey 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 "C" and assembly in Figures 2-1 through 2-4. The sample program is skdemo consisting of source programs *skdemo.c* and *init.s* The program emulates a primitive command interpreter. The sample program is shipped with the Softkey Interface and may be copied from the following location.

/usr/hp64000/demo/emul/hp64758/skdemo.c

/usr/hp64000/demo/emul/hp64758/init.s

The sample program is checking for a new command continually. If a new command, other than 20 (hex), is entered, the command interpreter routine (**_cmd_process**) is called. The command interpreter interprets the command entered and outputs the resulting message and status.

cmd_code and cmd_result

The switch statement evaluates the value of cmd_code with cases within it. You will change the **_cmd_code** (to 'A', 'B' or 'C') to match each of the cases as you progress through the steps in this manual. As you enter into each branch of the switch statement:

If case CMD_A is satisfied, the **_cmd_result** (command 'A' entered), is displayed.

If case CMD_B is satisfied, the **_cmd_result** (command 'B' entered), is displayed.

If case CMD_C is satisfied, the **_cmd_result** (command 'C' entered), is displayed.

If any case statement is not satisfied, the **_cmd_result** (invalid command entered), is displayed.

When the case statement is completed, the **_cmd_code** will be assigned to the value of NO_CMD.

status

Status contains the message "Awaiting command" when the program is started. Once you enter a command, "Command received" will be displayed.

init.s

The init.s file defines start-up routine for C program skdemo.c and 70632 breakpoint instruction vector. The start-up routine performs preparing the stack, setting up stack pointer, and calling to **_main** function defined in *skdemo.c.* The breakpoint instruction vector is required for the emulator’s software breakpoint feature. The vector has to point to a memory location where permitted to fetch an instruction.

```
#define TRUE            1
#define FALSE           0

#define CMD_A           'A'
#define CMD_B           'B'
#define CMD_C           'C'
#define NO_CMD          ' '
#define MSG_SIZ         0x20

static char status[MSG_SIZ];
static char cmd_result [MSG_SIZ];
static char cmd_code;

main ()
{
    msgcpy (status, "Awaiting command", MSG_SIZ);

    cmd_code = NO_CMD;
    msgcpy (cmd_result, "No command entered", MSG_SIZ);

    while(TRUE) {
        if(cmd_code != NO_CMD) {
            cmd_process ( cmd_code, cmd_result);
            cmd_code = NO_CMD;
        }
    }
}
```

Figure 2-1. C Source skdemo.c

```
int cmd_process (cmd_code, cmd_result)
char cmd_code;
char *cmd_result;
{

    msgcpy (status, "Command received", MSG_SIZ);

    switch (cmd_code) {
    case CMD_A :
        msgcpy (cmd_result, "Command 'A' entered", MSG_SIZ);
        break;

    case CMD_B :
        msgcpy (cmd_result, "Command 'B' entered", MSG_SIZ);
        break;

    case CMD_C :
        msgcpy (cmd_result, "Command 'C' entered", MSG_SIZ);
        break;

    default :
        msgcpy (cmd_result, "Invalid command entered", MSG_SIZ);
    }

}


msgcpy( msg_dst, msg_src, msg_siz)
char    *msg_dst;
char    *msg_src;
int     msg_siz;
{
    for (; *msg_src != '\0' && msg_siz > 0; --msg_siz)
        *msg_dst++ = *msg_src++;

    for (; msg_siz > 0; --msg_siz)
        *msg_dst++ = ' ';
}
```

Figure 2-1. C Source skdemo.c (Cont'd)

```
                .file   "init.s"

                .equ    Stack_Size, 0x100

                .globl  _main, Init

                .bss    "sbt" (RW)
                .lcomm  Sbt, 0x34, 0x100
                .lcomm  brkvect, 4, 4

                .text   (RX)
Init:
                movea.w Dummy_Text, brkvect
                mov.w   #Stack+Stack_Size, sp
                call    _main,[sp]
                jr      .

Dummy_Text:     halt

                .bss    (RW)
                .lcomm  Stack, Stack_Size,4
```

Figure 2-2. init.s Source Program

```
SECTIONS
{
        sbt  0x00000:
        {
        }

        .text 0x10000:
        {
        }

        .data 0x80000:
        {
        }

        .bss  0xf0000:
        {
        }

}
```

Figure 2-3. Linker Command File

Compiling, Assembling and Linking the Program

NEC Corporation CC70616 C Compiler Package is used to compile, assemble, and link the demo program. The package are available for use in the HP 9000 300 Series work stations from NEC.

The *v70cnvhp* utility is used to generate the required HP format files. The file *skdemo.X* contains the absolute code of the program. The file *skdemo.L* contains the list of global symbols. The files *skdemo.A* and *init.A* each contain a list of local symbols for the respective files.

The user interface provides source line referencing if line information is present in the local symbol file. Line number information is included if the **-g** option is used with either the "C" compiler or the assembler.

The following commands are used to compile, assemble, and link the demo program.

```
$as70616 -a init.s >init.lis
$cc70616 -cg skdemo.c
$ld70616 skdemo.lnk -m -o skdemo init.o skdemo.o >skdemo.map
$v70cnvhp skdemo
```

The linker command file *skdemo.lnk* is listed in figure 2-3.

Entering the Softkey Interface

If you have installed your emulator and Softkey Interface software as directed in the *HP 64700 Series Emulators Softkey Interface Installation Notice*, you are ready to enter the interface. The Softkey Interface can be entered through the **pmon** User Interface Software or from the HP-UX shell.

- If you used previous HP 64000-UX emulators (for example, the HP 64200 Series), you may be more familiar with the **pmon**, **msinit**, and **msconfig** command method of entering the emulation interface.

- If you wish to run the Softkey Interface in multiple windows, you must enter from the HP-UX shell using the **emul700** command. Refer to the *Softkey Interface Reference* manual for more information on running in multiple windows.

From the "pmon" User Interface

If **/usr/hp64000/bin** is specified in your PATH environment variable, you can enter the pmon User Interface with the following command.

```
$ pmon <RETURN>
```

If you have not already created a measurement system for the 70632 emulator, you can do so with the following commands. First you must initialize the measurement system with the following command.

```
MEAS_SYS msinit <RETURN>
```

After the measurement system has been initialized, enter the configuration interface with the following command.

```
msconfig <RETURN>
```

To define a measurement system for the 70632 emulator, enter:

```
make_sys emv70 <RETURN>
```

Now, to add the emulator to the measurement system, enter:

```
add <module_number> naming_it v70
<RETURN>
```

Enter the following command to exit the measurement system configuration interface.

```
end <RETURN>
```

If the measurement system and emulation module are named "emv70" and "v70" as shown above, you can enter the emulation system with the following command:

`emv70 default v70 <RETURN>`

If this command is successful, you will see a display similar to figure 2-4. The status message shows that the default configuration file has been loaded. If the command is not successful, you will be given an error message and returned to the **pmon** User Interface. Error messages are described in the *Softkey Interface Reference* manual.

For more information on creating measurements systems, refer to the *Softkey Interface Reference* manual.

Note

The measurement system name emv70 and the emulation module name v70 are of the user’s choice.

From the HP-UX Shell

If **/usr/hp64000/bin** is specified in your PATH environment variable, you can also enter the Softkey Interface with the following command.

`$ emul700 <emul_name> <RETURN>`

The "emul_name" in the command above is the logical emulator name given in the HP 64700 emulator device table (/usr/hp64000/etc/64700tab).

If this command is successful, you will see a display similar to figure 2-4. The status message shows that the default configuration file has been loaded. If the command is not successful, you will be given an error message and returned to the HP-UX prompt. Error messages are described in the *Softkey Interface Reference* manual.

```
               HP64758-19001 A.02.00 15Jun90 Unreleased
                     70632 EMULATION SERIES 64700

                   A Hewlett-Packard Software Product
                   Copyright Hewlett-Packard Co. 1990

  All Rights Reserved. Reproduction, adaptation, or translation without prior
  written  permission  is prohibited, except as allowed under copyright laws.

                        RESTRICTED RIGHTS LEGEND

     Use , duplication , or disclosure  by the  Government is  subject to
     restrictions as set forth in subparagraph (c) (1) (II) of the Rights
     in Technical Data and Computer Software clause at  DFARS 52.227-7013.
     HEWLETT-PACKARD Company , 3000 Hanover St. , Palo Alto, CA 94304-1181


STATUS:   Loaded configuration file_____________________________________...R....



  run     trace     step   display          modify   break     end    ---ETC--
```

Figure 2-4. Softkey Interface Display

On-Line Help

There are two ways to access on-line help in the Softkey Interface. The first is by using the Softkey Interface help facility. The second method allows you to access the firmware resident Terminal Interface on-line help information.

Softkey Driven Help

To access the Softkey Interface on-line help information, type either "**help**" or "**?**" on the command line; you will notice a new set of softkeys. By pressing one of these softkeys and <RETURN>, you can cause information on that topic to be displayed on your screen. For example, you can enter the following command to access "system command" help information.

? **system_commands** <RETURN>

The help information is scrolled on to the screen. If there is more than a screenful of information, you will have to press the space bar to see the next screenful, or the <RETURN> key to see the next line, just as you do with the HP-UX **more** command. After all the information on the particular topic has been displayed (or after you press "q" to quit scrolling through information), you are prompted to press <RETURN> to return to the Softkey Interface.

```
---SYSTEM COMMANDS---

?                         displays the possible help files
help                      displays the possible help files
!                         fork a shell (specified by  shell variable SH)
!<shell cmd>              fork a shell and execute a shell command
cd <directory>            change the working directory
pwd                       print the working directory
cws <SYMB>                change the working symbol - the working symbol also
                             gets updated when displaying local symbols and
                             displaying memory mnemonic
pws                       print the working symbol
<FILE> p1 p2 p3 ...       execute a command file passing parameters p1, p2, p3

log_commands to <FILE>    logs the next sequence of commands to file <FILE>
log_commands off          discontinue logging commands
name_of_module            get the "logical" name of this module (see 64700tab)
set <ENVVAR> = <VALUE>    set and export a shell environment variable
set HP64KPATH = <MYPATH>  set and export the shell environment variable that
                             specifies the search path for command files
wait                      pause until <cntrl-c> (SIGINT)
--More--(42%)
```

Pod Command Help

To access the emulator's firmware resident Terminal Interface help information, you can use the following commands.

display pod_command <RETURN>
pod_command 'help bp' <RETURN>

```
Pod Commands
  Time                Command

23:09:36 help bp

  bp - set, enable, disable, remove or display software breakpoints
    bp                 - display current breakpoints
    bp <addr>          - set breakpoint at <addr>
    bp -e *            - enable all breakpoints
    bp -e <addr>       - enable breakpoint at <addr>
    bp -d *            - disable all breakpoints
    bp -d <addr>       - disable breakpoint at <addr>
    bp -r *            - remove all breakpoints
    bp -r <addr>       - remove breakpoint at <addr>
    bp <addr> <addr>   - more than one <addr> may be given
  --- NOTES ---
    Enable/disable breaking on software breakpoints via the bc command.
    Maximum number of breakpoints available is 32.

STATUS:   N70632--Running in monitor____________________________________...R....
 pod_command 'help bp'


pod_cmd    set    perfinit perfrun          perfend                    ---ETC--
```

The command enclosed in string delimiters (", ’, or ^) is any Terminal Interface command, and the output of that command is seen in the **pod_command** display. The Terminal Interface **help** (or ?) command may be used to provide information on any Terminal Interface command or any of the emulator configuration options (as the example command above shows).

Configuring the Emulator

You need to configure the emulator for this tutorial. To configure the emulator, type the following command to get into the configuration session.

modify configuration <RETURN>

Trace the following answer to configure the emulator. Details of each question will be described later.

```
Micro-processor clock source? internal
Enter monitor after configuration? yes
Restrict to real-time runs? no
Modify memory configuration? yes
Monitor type? background
```

Now you should be facing memory mapping screen. The address range 0 through 0ffffH is mapped as emulation RAM by default. More three mapper terms must be specified for the sample program. Enter the following lines to map the program code and constant data areas as emulation ROM, the variable data area as emulation RAM.

10000h ***thru*** 10fffh ***emulation rom***

80000h ***thru*** 80fffh ***emulation rom***

0f0000h ***thru*** 0f0fffh ***emulation ram***

The unmapped area should be defined as "guarded" to detect illegal accesses to the area.

default guarded

end

```
Modify emulator pod configuration? no
Modify debug/trace options? no
Modify simulated I/O configuration? no
Modify interactive measurement specification? no
Configuration file name? skdemo
```

Loading Absolute Files

The "**load**" command allows you to load absolute files into emulation or target system memory. If you wish to load only that portion of the absolute file that resides in memory mapped as emulation RAM or ROM, use the "**load emul_mem**" syntax. If you wish to load only the portion of the absolute file that resides in memory mapped as target RAM, use the "**load user_mem**" syntax. If you want both emulation and target memory to be loaded, do not specify "emul_mem" or "user_mem". For example:

load skdemo <RETURN>

Displaying Symbols

When you load an absolute file into memory (unless you use the "nosymbols" option), symbol information is loaded. Both global symbols and symbols that are local to a source file can be displayed.

Global

To display global symbols, enter the following command.

display global_symbols <RETURN>

```
Global symbols in skdemo
Procedure symbols
Procedure name _________________ Address range __ Segment _____________ Offset
_cmd_process                  000100A8 - 0001016B                          0088
_main                         00010020 - 000100A6                          0000
_msgcpy                       0001016C - 000101C6                          014C

Static symbols
Symbol name ____________________ Address range __ Segment _____________ Offset
Init                          00010000                                     0000
_edata                        0008008C                                     0000
_end                          000F0144                                     0000
_esbt                         00000104                                     0000
_etext                        000101C8                                     0000

Filename symbols
Filename ______________________________________________________________________
init.s

STATUS:   N70632--Running in monitor____________________________________...R....
 display  global_symbols


  run     trace     step   display          modify   break     end    ---ETC--
```

Listed are: address ranges associated with a symbol and the offset of that symbol.

Local When displaying local symbols, you must include the name of the source file in which the symbols are defined. For example,

display local_symbols_in skdemo.c:

```
Symbols in skdemo.c:
Procedure symbols
Procedure name __________________ Address range __ Segment _____________ Offset
_cmd_process                      000100A8 - 0001016B                      0088
_main                             00010020 - 000100A6                      0000
_msgcpy                           0001016C - 000101C6                      014C

Static symbols
Symbol name _____________________ Address range __ Segment _____________ Offset
_cmd_code                         000F0140                                 0040
_cmd_result                       000F0120                                 0020
_status                           000F0100                                 0000

Source reference symbols
Line range ______________________ Address range __ Segment _____________ Offset
#1-#16                            0001002C - 00010048                      000C
#17-#18                           00010049 - 00010051                      0029
#19-#19                           00010052 - 0001006E                      0032

STATUS:   cws: skdemo.c:________________________________________________...R....
 display  local_symbols_in skdemo.c:


  run     trace     step   display          modify   break     end    ---ETC--
```

<RETURN>

Loading a program will by default load the absolute code, global symbols, and local symbols. Displaying the local symbols will make the specified set of symbols active.

If source line number information is contained in the local symbol file, the memory locations may be referenced by source line numbers. Line number 1 is the first line in a source file, line number 2 is the second line, . . . etc.

Displaying Memory in Mnemonic Format

You can display, in mnemonic format, the absolute code in memory. To display memory in mnemonic format from the address of label **_main,** enter the following command:

display memory _main ***mnemonic***
<RETURN>

```
 Memory  :mnemonic :file = skdemo.c:
   address   data
  00010020  ECF4000000  PUSHM      #0m<>
  00010026  DEF4000000  PREPARE    #00000000H
  0001002C  EEF4200000  PUSH       #00000020H
  00010032  EEF4000008  PUSH       #00080000H
  00010038  EEF400010F  PUSH       #000F0100H
  0001003E  4980F22E01  CALL       0001016CH,[SP]
  00010046  843FEC      ADD.W      #CH,SP
  00010049  0980F420F2  MOV.B      #20H,000F0140H
  00010052  EEF4200000  PUSH       #00000020H
  00010058  EEF4110008  PUSH       #00080011H
  0001005E  EEF420010F  PUSH       #000F0120H
  00010064  4980F20801  CALL       0001016CH,[SP]
  0001006C  843FEC      ADD.W      #CH,SP
  0001006F  B880F420F2  CMP.B      #20H,000F0140H
  00010078  6424        BE/Z       0001009CH
  0001007A  EEF420010F  PUSH       #000F0120H

STATUS:   Warning: no ENTRY/EXIT symbol; using TEXTRANGE________________...R....
 display  memory _main mnemonic


  run     trace     step   display          modify   break     end    ---ETC--
```

Displaying Memory with Symbols

You can include symbol information in memory display.

set symbols on <RETURN>

```
 Memory  :mnemonic :file = skdemo.c:
   address  label          data
  00010020     :_main     ECF4000000  PUSHM      #0m<>
  00010026                DEF4000000  PREPARE    #00000000H
  0001002C                EEF4200000  PUSH       #00000020H
  00010032                EEF4000008  PUSH       #00080000H
  00010038                EEF400010F  PUSH       #000F0100H
  0001003E                4980F22E01  CALL       :_msgcpy,[SP]
  00010046                843FEC      ADD.W      #CH,SP
  00010049                0980F420F2  MOV.B      #20H,skdemo:_cmd_code
  00010052                EEF4200000  PUSH       #00000020H
  00010058                EEF4110008  PUSH       #00080011H
  0001005E                EEF420010F  PUSH       #000F0120H
  00010064                4980F20801  CALL       :_msgcpy,[SP]
  0001006C                843FEC      ADD.W      #CH,SP
  0001006F                B880F420F2  CMP.B      #20H,skdemo:_cmd_code
  00010078                6424        BE/Z       :_main+0000007CH
  0001007A                EEF420010F  PUSH       #000F0120H

STATUS:   N70632--Running in monitor____________________________________...R....
 set  symbols  on


pod_cmd    set    perfinit perfrun         perfend                    ---ETC--
```

Displaying Memory with Source Lines

You can include source program lines in memory display.

set source on <RETURN>

```
 Memory  :mnemonic :file = skdemo.c:
   address  label          data
  00010020     :_main     ECF4000000  PUSHM      #0m<>
  00010026                DEF4000000  PREPARE    #00000000H
      12    static char cmd_code;
      13
      14    main ()
      15    {
      16        msgcpy (status, "Awaiting command", MSG_SIZ);
  0001002C                EEF4200000  PUSH       #00000020H
  00010032                EEF4000008  PUSH       #00080000H
  00010038                EEF400010F  PUSH       #000F0100H
  0001003E                4980F22E01  CALL       :_msgcpy,[SP]
  00010046                843FEC      ADD.W      #CH,SP
      17
      18        cmd_code = NO_CMD;
  00010049                0980F420F2  MOV.B      #20H,skdemo:_cmd_code
      19        msgcpy (cmd_result, "No command entered", MSG_SIZ);

STATUS:   N70632--Running in monitor____________________________________...R....
 set  source  on


pod_cmd    set    perfinit perfrun          perfend                    ---ETC--
```

Note

The "**set**" command is effective only to the window in which the command is invoked. You need to use this command at each window.

Running the Program

The "**run**" command lets you execute a program in memory. Entering the "**run**" command by itself causes the emulator to begin executing at the current program counter address. The "**run from**" command allows you to specify an address at which execution is to start. For example to run the sample program from the address of Init label,

run from `Init <RETURN>`

You will see that the status line on your screen is changed to "Running user program".

From Reset

The "**run from reset**" command specifies that the emulator begin executing from target system reset.

Displaying Memory Repetitively

You can display memory locations repetitively so that the information on the screen updates constantly. For example, to display the **_cmd_result** and the **_status** locations of the sample program repetitively (in blocked byte format), enter the following command.

```
display memory skdemo.c:_cmd_result
thru +1fh, skdemo.c:_status thru
+1fh blocked bytes <RETURN>
```

Notice that when using local symbols in expression, the source file in which the local symbol is defined must be included.

When you display/modify the memory location, you can specify the data size or type to be displayed/modified. The following data size/type are allowed.

bytes	(one byte integer)
words	(two bytes integer)
long	(four bytes integer)
real (short)	(four bytes floating-point)
real long	(eight bytes floating-point)

```
 Memory  :bytes :blocked :repetitively
   address       data       :hex                                           :ascii
  000F0120-27    4E   6F   20   63   6F   6D   6D   61      N o   c  o m m a
  000F0128-2F    6E   64   20   65   6E   74   65   72      n d   e  n t e r
  000F0130-37    65   64   20   20   20   20   20   20      e d
  000F0138-3F    20   20   20   20   20   20   20   20
  000F0100-07    41   77   61   69   74   69   6E   67      A w a i  t i n g
  000F0108-0F    20   63   6F   6D   6D   61   6E   64        c o m  m a n d
  000F0110-17    20   20   20   20   20   20   20   20
  000F0118-1F    20   20   20   20   20   20   20   20









STATUS:   N70632--Running user program__________________________________...R....
 display  memory skdemo.c:_cmd_result thru +1fh, skdemo.c:_status thru +1fh repe
titively  blocked  bytes

  run     trace     step   display          modify   break     end    ---ETC--
```

Modifying Memory

The sample program simulates a primitive command interpreter. Commands are sent to the sample program through a byte sized memory location labeled **_cmd_code**. You can use the modify memory feature to send a command to the sample program. For example, to enter the command "A" (41 hex), use the following command.

modify memory `_cmd_code` ***bytes to*** `41h`
`<RETURN>`

Or:

modify memory `_cmd_code` ***bytes to*** `'A'`
`<RETURN>`

(Single character strings are allowed in expressions.)

```
 Memory  :bytes :blocked :repetitively
   address       data       :hex                                      :ascii
  000F0120-27    43   6F   6D   6D   61   6E   64   20      C o m m  a n d
  000F0128-2F    27   41   27   20   65   6E   74   65      ' A '    e n t e
  000F0130-37    72   65   64   20   20   20   20   20      r e d
  000F0138-3F    20   20   20   20   20   20   20   20
  000F0100-07    43   6F   6D   6D   61   6E   64   20      C o m m  a n d
  000F0108-0F    72   65   63   65   69   76   65   64      r e c e  i v e d
  000F0110-17    20   20   20   20   20   20   20   20
  000F0118-1F    20   20   20   20   20   20   20   20







STATUS:   N70632--Running user program__________________________________...R....
 modify  memory _cmd_code bytes  to 41h


  run     trace     step   display          modify   break     end    ---ETC--
```

As you can see, the memory display is automatically updated, and shows that the "Command 'A' entered" message is written to the destination locations.

Breaking into the Monitor

The "**break**" command allows you to divert emulator execution from the user program to the monitor. You can continue user program execution with the "**run**" command. To break emulator execution from the sample program to the monitor, enter the following command.

break <RETURN>

You will find that the status line on the screen is changed to "Running in monitor".

Using Software Breakpoints

Software breakpoints are handled by the 70632 BRK instruction. When you define or enable a software breakpoint (with the bp command), the emulator will replace the opcode at the software breakpoint address with a breakpoint interrupt instruction (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 up the BRK instruction vector pointing to an address allowed instruction fetch; 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 using the software breakpoints features. Refer to the "Software Breakpoints" section of the "Using the Emulator" chapter.

Up to 32 software breakpoints may be defined.

Display the software breakpoints status screen, by entering:

display software_breakpoints

The display shows that no software breakpoint is defined.

Enabling/Disabling Software Breakpoints

When you initially enter the Softkey Interface, software breakpoints are disabled. To enable the software breakpoints feature, enter the following command.

modify software_breakpoints enable <RETURN>

The top of the screen indicates that software breakpoint feature is enabled.

When software breakpoints are enabled and you set a software breakpoint, the 70632 BRK instruction will be placed at the address specified. When the BRK instruction is executed, program execution will break into the monitor.

Setting a Software Breakpoint

To set a software breakpoint at the address of the **_cmd_process** label, enter the following command.

modify software_breakpoints set `_cmd_process <RETURN>`

Notice that when using local symbols in expressions, the source file in which the local symbol is defined must be included.

```
Software breakpoints  :enabled
   address         label         status
  000100A8 @r      :_cmd_proces  pending

STATUS:   Warning: no ENTRY/EXIT symbol; using TEXTRANGE________________...R....
 modify  software_breakpoints  set _cmd_process


  run     trace     step   display          modify   break     end    ---ETC--
```

After the software breakpoint has been set, enter the following commands to display memory and see if the software breakpoint was correctly inserted.

display memory `_cmd_process` ***mnemonic*** `<RETURN>`

```
 Memory  :mnemonic :file = skdemo.c:
   address  label          data
* 000100A8  :_cmd_proces  C8          BRK
  000100A9                F40000      TEST.W     00H[R0]
  000100AC                00          HALT
  000100AD                00          HALT
  000100AE                DEF4000000  PREPARE    #00000000H
      29    int cmd_process (cmd_code, cmd_result)
      30    char cmd_code;
      31    char *cmd_result;
      32    {
      33        msgcpy (status, "Command received", MSG_SIZ);
  000100B4                EEF4200000  PUSH       #00000020H
  000100BA                EEF4240008  PUSH       #00080024H
  000100C0                EEF400010F  PUSH       #000F0100H
  000100C6                4980F2A600  CALL       :_msgcpy,[SP]
  000100CE                843FEC      ADD.W      #CH,SP
      34

STATUS:   Warning: no ENTRY/EXIT symbol; using TEXTRANGE________________...R....
 display  memory _cmd_process mnemonic


  run     trace     step   display           modify   break     end    ---ETC--
```

As you can see, the software breakpoint is shown in the memory display with an asterisk, and the instruction at the address is replaced with a BRK instruction.

Enter the following command to run the sample program again.

run from `Init <RETURN>`

Now, modify the command input byte to an invalid command for the sample program.

modify memory `_cmd_code` ***bytes to*** `75h <RETURN>`

You will see the line of the software breakpoint is displayed in inverse-video. The inverse-video shows that the Program Counter is now at the address.

A message on the status line shows that the software breakpoint has been hit. The status line also shows that the emulator is now executing in the monitor.

Display the software breakpoint status, by entering:

display software_breakpoints `<RETURN>`

```
Software breakpoints  :enabled
   address         label         status
  000100A8 @r      :_cmd_proces  inactivated




















STATUS:   N70632--Running in monitor      Software break: 0000100a8@r___...R....
 display  software_breakpoints


  run     trace     step   display           modify   break     end    ---ETC--
```

When software breakpoints are hit, they become inactivated. To reactive the breakpoint so that is "pending", you must reenter the "modify software_breakpoints set" command.

modify software_breakpoints set <RETURN>

If you display the memory contents in mnemonic format, the contents of the address you specify the breakpoint is replaced with the BRK instruction.

display memory <RETURN>

Clearing a Software Breakpoint

To remove software breakpoint defined above, enter the following command.

modify software_breakpoints clear _cmd_process <RETURN>

The breakpoint is removed from the list, and the original opcode is restored if the breakpoint was pending. To clear all software breakpoints, you can enter the following command.

modify software_breakpoints clear <RETURN>

```
 Memory  :mnemonic :file = skdemo.c:
   address  label          data
  000100A8  :_cmd_proces  ECF4000000  PUSHM      #0m<>
  000100AE                DEF4000000  PREPARE    #00000000H
      29    int cmd_process (cmd_code, cmd_result)
      30    char cmd_code;
      31    char *cmd_result;
      32    {
      33        msgcpy (status, "Command received", MSG_SIZ);
  000100B4                EEF4200000  PUSH       #00000020H
  000100BA                EEF4240008  PUSH       #00080024H
  000100C0                EEF400010F  PUSH       #000F0100H
  000100C6                4980F2A600  CALL       :_msgcpy,[SP]
  000100CE                843FEC      ADD.W      #CH,SP
      34
      35        switch (cmd_code) {
  000100D1                0C207D      MOVS.BW    [AP],R0
  000100D4                6A72        BR         :_cmd_process+0000009EH

STATUS:   Warning: no ENTRY/EXIT symbol; using TEXTRANGE________________...R....
 modify  software_breakpoints  clear _cmd_process


  run     trace     step   display          modify   break     end    ---ETC--
```

Stepping Through the Program

The step command allows you to step through program execution an instruction or a number of instructions at a time. Also, you can step from the current program counter or from a specific address. To step through the example program from the address of the software breakpoint set earlier, enter the following command.

`step` `<RETURN>, <RETURN>, <RETURN>, ...`

You will see the inverse-video moves according to the step execution. You can continue to step through the program just by pressing the <RETURN> key; when a command appears on the command line, it may be entered by pressing <RETURN>.

You can step program execution by source lines, enter:

`step source`

Source line stepping is implemented by single stepping assembly instructions until the next PC is outside of the address range of the current source line. When source line stepping is attempted on assembly code, stepping will complete when a source line is found. To terminate stepping type <Ctrl>-C.

```
Registers

Next PC 00010178@r
 PC 00010178  SP 000F00C8  FP 000F00C8  AP 000F00D4  PSW 10000000
 R0-7   000F013F 00080022 00000000 00000000 00000000 00000000 00000000 00000000
 R8-15  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 R16-23 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 R24-31 00000000 00000000 00000000 00000000 00000000 000F00D4 000F00C8 000F00C8









STATUS:   N70632--Stepping complete_____________________________________...R....
 display  registers


  run     trace     step   display          modify   break     end    ---ETC--
```

Displaying Registers

Enter the following command to display registers. You can display the basic registers class, or an individual register.

display registers <RETURN>

When you enter the "**step**" command with registers displayed, the register display is updated every time you enter the "step" command.

step <RETURN>, <RETURN>, <RETURN>

```
Registers

Step_PC 0001017B@r  BE/Z       :_msgcpy+0000003BH
Next PC 0001017D@r
 PC 0001017D  SP 000F00C8  FP 000F00C8  AP 000F00D4  PSW 10000000
 R0-7   000F013F 00080022 00000000 00000000 00000000 00000000 00000000 00000000
 R8-15  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 R16-23 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 R24-31 00000000 00000000 00000000 00000000 00000000 000F00D4 000F00C8 000F00C8

Step_PC 0001017D@r  TEST.W     08H[AP]
Next PC 00010180@r
 PC 00010180  SP 000F00C8  FP 000F00C8  AP 000F00D4  PSW 10000000
 R0-7   000F013F 00080022 00000000 00000000 00000000 00000000 00000000 00000000
 R8-15  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 R16-23 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 R24-31 00000000 00000000 00000000 00000000 00000000 000F00D4 000F00C8 000F00C8

STATUS:   N70632--Stepping complete_____________________________________...R....
 step

  run     trace     step   display          modify   break     end    ---ETC--
```

Enter the following command to cause sample program execution to continue from the current program counter.

run <RETURN>

Using the Analyzer

HP 64700 emulators contain an emulation analyzer. The emulation analyzer monitors the internal emulation lines (address, data, and status). 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.

Specifying a Simple Trigger

Suppose you want to trace program execution around the point at which the sample program read the byte value 42H (CMD_B) from the address **_cmd_code**. The following command makes this trace specification.

trace about skdemo.c:_cmd_code ***data*** 0xxxxxx42h ***status read***<RETURN>

Note that the analyzer is to search for a lowest byte read of 42H because the address is a multiple of four.

The message "Emulation trace started" will appear on the status line. Now, modify the command input byte to "B" with the following command.

modify memory skdemo.c:_cmd_code
bytes to 42h <RETURN>

The status line now shows "Emulation trace complete".

Displaying the Trace

The trace listings which follow are of program execution on the 70632 emulator. To display the trace, enter:

display trace <RETURN>

```
Trace List                  Offset=0
Label:      Address       Data    Opcode or Status w/ Source Lines
Base:       symbols       hex            mnemonic w/symbols
-003    :_main+00000060 C0F2A00C              fetch
-002    :_main+00000064 BF000E00              fetch
      ##########skdemo.c - line    20 thru    22 ##################

          while(TRUE) {
              if(cmd_code != NO_CMD) {
-001    :_main+0000004F FFFFFF42   CMP.B     #20H,skdemo:_cmd_code
about  skdemo:_cmd_code FFFFFF42     ......42H data  read
+001    :_main+00000058 FFFFFF42   BE/Z      :_main+0000007CH
+002    :_main+00000068 20F28049              fetch
+003    :_main+0000006C 7F000000              fetch
      ##########skdemo.c - line    23 ##############################
                  cmd_process ( cmd_code, cmd_result);
+004    :_main+0000005A 7F000000   PUSH      #000F0120H
+005    :_main+00000070 09E83F84              fetch

STATUS:   N70632--Running user program    Emulation trace complete______...R....
 display  trace


  run     trace     step   display           modify   break     end    ---ETC--
```

The line labeled "about" in the trace list shows the state which triggered the analyzer. To list the next lines of the trace, press the <**PGDN**> or <**NEXT**> key.

```
Trace List                  Offset=0
Label:      Address       Data    Opcode or Status w/ Source Lines
Base:       symbols       hex            mnemonic w/symbols
+006   skdemo:_cmd_code FFFFFF42     ......42H data  read
+007   :init.:+000000F0 000F0120     000F0120H data  write
+008    :_main+00000074 F220F480               fetch
+009    :_main+00000060 F220F480   MOVS.BW   skdemo:_cmd_code,[-SP]
+010    :_main+00000078 000E00AD               fetch
+011    :_main+0000007C E4CCD36A               fetch
+012   :init.:+000000EC 00000042     00000042H data  write
+013    :_cmd_process   0000F4EC               fetch aft br
+014   :_cmd_p+00000004 F4DE0000               fetch
+015   :_cmd_p+00000008 00000000               fetch
      ##########skdemo.c - line    28 thru    33 ##################
      int cmd_process (cmd_code, cmd_result)
      char cmd_code;
      char *cmd_result;
      {

STATUS:   N70632--Running user program    Emulation trace complete______...R....
 display  trace


  run     trace     step   display           modify   break     end    ---ETC--
```

Displaying the Trace with Compress Mode

At this time you may want to see more executed instructions on a display. To see flow of executed instructions, the 70632 emulator Softkey Interface provides compress mode for analysis display. To see trace display with compress mode, enter the following command.

display trace compress on <RETURN>

Your analysis trace display should look similar as below. You can see executions without prefetch cycles.

If you want to see all of cycles including prefetch cycles, enter "**display trace compress off**" command.

```
Trace List                  Offset=0
Label:      Address       Data    Opcode or Status w/ Source Lines
Base:       symbols       hex            mnemonic w/symbols
+006   skdemo:_cmd_code FFFFFF42     ......42H data  read
+007   :init.:+000000F0 000F0120     000F0120H data  write
+009    :_main+00000060 F220F480   MOVS.BW   skdemo:_cmd_code,[-SP]
+012   :init.:+000000EC 00000042     00000042H data  write
+017    :_main+00000068 0020F4EE   CALL      :_cmd_process,[SP]
+018   :init.:+000000E4 00010090     00010090H data  write
+020   :init.:+000000E8 000F0100     000F0100H data  write
+022    :_cmd_process   0100F4EE   PUSHM     #0m<>
+024   :_cmd_p+00000006 0100F4EE   PREPARE   #00000000H
+025   :init.:+000000E0 000F00F4     000F00F4H data  write
      ##########skdemo.c - line    28 thru    33 ##################
      int cmd_process (cmd_code, cmd_result)
      char cmd_code;
      char *cmd_result;
      {

STATUS:   N70632--Running user program    Emulation trace complete______...R....
 display  trace  compress  on


  run     trace     step   display          modify   break     end    ---ETC--
```

The trace listing includes source lines and symbols because you issued "**set symbols on**" and "**set source on**" command in the previous section. You can cause these source lines highlight by entering the following command.

set source on inverse_video on <RETURN>

To list the previous lines of the trace, press the <**PGUP**> or <**PREV**> key.

Changing the Trace Depth

The default states displayed in the trace list is 256 states. To change the number of states, use the "display trace depth" command.

display trace depth 1024 <RETURN>

You can see the states more than 256 by using the above command.

Using the Storage Qualifier

You can use storage qualifier to trace only states with specific conditions. Suppose that you would like to trace only states which write the messages to the cmd_result area. To accomplish this, you can use the "**trace only**" command like following.

trace after _cmd_result ***only range*** _cmd_result ***thru*** +1fh ***status write*** <RETURN>

Only write accesses to address **_cmd_result** through **_cmd_result+1fh** will be stored in the trace buffer.

```
Trace List                  Offset=0
Label:      Address       Data    Opcode or Status w/ Source Lines
Base:       symbols       hex            mnemonic w/symbols
after  skde:_cmd_result FFFFFF43     ......43H data  write
+001   :skdem:+00000021 FFFF6FFF     ....6F..H data  write
+002   :skdem:+00000022 FF6DFFFF     ..6D....H data  write
+003   :skdem:+00000023 6DFFFFFF     6D......H data  write
+004   :skdem:+00000024 FFFFFF61     ......61H data  write
+005   :skdem:+00000025 FFFF6EFF     ....6E..H data  write
+006   :skdem:+00000026 FF64FFFF     ..64....H data  write
+007   :skdem:+00000027 20FFFFFF     20......H data  write
+008   :skdem:+00000028 FFFFFF27     ......27H data  write
+009   :skdem:+00000029 FFFF41FF     ....41..H data  write
+010   :skdem:+0000002A FF27FFFF     ..27....H data  write
+011   :skdem:+0000002B 20FFFFFF     20......H data  write
+012   :skdem:+0000002C FFFFFF65     ......65H data  write
+013   :skdem:+0000002D FFFF6EFF     ....6E..H data  write
+014   :skdem:+0000002E FF74FFFF     ..74....H data  write

STATUS:   N70632--Running user program    Emulation trace started_______...R....
 modify  memory _cmd_code bytes  to 41h


  run     trace     step   display          modify   break     end    ---ETC--
```

Modify the command input byte with the following command.

modify memory _cmd_code ***bytes to*** 41h <RETURN>

The display shows that the message bytes are written to the location **_cmd_result**. You will find the status line still shows "Emulation trace started" because the analyzer trace buffer is not filled up. As the length of resulting message consists of 32 bytes, only 32 states are stored in the trace buffer. If you want to stop the trace, enter the following command.

stop_trace <RETURN>

The status line will show "Emulation trace complete".

Triggering the Analyzer at an Instruction Execution State

The emulation analyzer can capture states of instruction executions. If you want to trigger the analyzer when an instruction at a desired address is executed, you should not set up the analyzer trigger condition to detect only the address. If you do so, the analyzer will be also triggered in case that the address is accessed to prefetch the instruction, or read the data from the address. You should use the "**exec**" status qualifier.

Suppose that you want to trace the states of the execution after the instruction at the *line 43* of the *skdemo.c* file, issue the following command. The *line 43* of the file *skdemo.c* is executed when the command "C" is entered.

```
Trace List                  Offset=0
Label:      Address       Data    Opcode or Status w/ Source Lines
Base:       symbols       hex            mnemonic w/symbols
      ##########skdemo.c - line    43 thru    45 ##################

          case CMD_C :
              msgcpy (cmd_result, "Command 'C' entered", MSG_SIZ);
after  :_cmd_p+00000066 0000004F   No fetch cycle found
+002   :init.:+000000DC 00000020     00000020H data  write
+004   :_cmd_p+0000006C F4EE3B6A   No fetch cycle found
+005   :init.:+000000F0 000F0120     000F0120H data  read
+006   :init.:+000000D8 0008005D     0008005DH data  write
+008   :_cmd_p+00000072 0071F4EE   No fetch cycle found
+010   :init.:+000000D4 000F0120     000F0120H data  write
+015   :_cmd_p+00000075 64049DF0   No fetch cycle found
+016   :init.:+000000CC 00010125     00010125H data  write
+018   :init.:+000000D0 000F00EC     000F00ECH data  write
+020       :_msgcpy     202D236A   PUSHM     #0m<>

STATUS:   N70632--Running user program    Emulation trace complete______...R....
 modify  memory skdemo.c:_cmd_code bytes  to 43h


  run     trace     step   display          modify   break     end    ---ETC--
```

trace after `skdemo.c: line 43` ***status exec*** `<RETURN>`

The message "Emulation trace started" will appear on the status line. To trigger the analyzer, send the command "C" by entering:

modify memory `skdemo.c:_cmd_code` ***bytes to*** `43h <RETURN>`

The status line now shows "Emulation trace complete".

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.

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.

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

70632 Analysis Status Qualifiers

The status qualifier "**write**" was used in the example trace command used above. The following analysis status qualifiers may also be used with the 70632 emulator.

```
fetch          0x1xxxxxxxxxx011x     code fetch
brfetch        0x1xxxxxxxxxx0111     code fetch after branch
read           0x1xxxxxxxxxxxxxx     read
write          0x0xxxxxxxxxxxxxx     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
tbl            0xxxxxxxxxxxx0101     translation table access (read/write)
coproc         0xxxxxxxxxxxx1000     co-processor access(read/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
block          0xxxxxxxx0xxxxxxx     bus lock
retry          00xxxxxxxxxxxxxxx     retry
holdtag        0xxxxxxxxxxxx0001     bus hold
```

For a Complete Description

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

Exiting the Softkey Interface

There are several options available when exiting the Softkey Interface: exiting and releasing the emulation system, exiting with the intent of reentering (continuing), exiting locked from multiple emulation windows, and exiting (locked) and selecting the measurement system display or another module.

End Release System

To exit the Softkey Interface, releasing the emulator so that other users may use the emulator, enter the following command.

end release_system <RETURN>

Ending to Continue Later

You may also exit the Softkey Interface without specifying any options; this causes the emulator to be locked. When the emulator is locked, other users are prevented from using it and the emulator configuration is saved so that it can be restored the next time you enter (continue) the Softkey Interface.

end <RETURN>

Ending Locked from All Windows

When using the Softkey Interface from within window systems, the "**end**" command with no options causes an exit only in that window. To end locked from all windows, enter the following command.

end locked <RETURN>

This option only appears when you enter the Softkey Interface via the **emul700** command. When you enter the Softkey Interface via **pmon** and **MEAS_SYS**, only one window is permitted. Refer to the *Softkey Interface Reference* manual for more information on using the Softkey Interface with window systems.

Selecting the Measurement System Display or Another Module

When you enter the Softkey Interface via **pmon** and **MEAS_SYS**, you have the option to select the measurement system display or another module in the measurement system when exiting the Softkey Interface. This type of exit is also "**locked**"; that is, you can continue the emulation session later. For example, to exit and select the measurement system display, enter the following command.

end select measurement_system <RETURN>

This option is not available if you have entered the Softkey Interface via the **emul700** command.

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.

Sample Program for Virtual Mode Emulation

The sample program is *skdemo2* consisting of source programs *command.c*, *process.c* and *os.s*. The program emulates a primitive command interpreter.

The file *os.s* is a simple operating system which performs task-switching, the file is listed in figure 3-1. The file *command.c* is a command generator, which is listed in figure 3-2. The file *process.c* is a command interpreter, which is listed in figure 3-3.

```
                .file   "os.s"

                .globl  Sys_SBT, Current_Task, Num_Of_Task, TCB_Entry
                .globl  Sys_Init, Setup_Task, Sys_Trap, Start_Ini_Task
                .globl  Switch_Task

                .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,0x1000-16
                .equ    Dest_Size,0x20

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


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

TCB_Entry:      .word   TCB_A
                .word   0x7fffffff
                .word   0x00000000
                .word   0x40000000
                .word   0x00006000
                .word   0
```

Figure 3-1. Sample Program Source os.s

```
                .word   TCB_B
                .word   0x7fffffff
                .word   0x00000000
                .word   0x40000000
                .word   0x00007000
                .word   0

TCB_A:          .word   0x0000e000
                .space  32*4
                .word   0x00009009,0x00000000

TCB_B:          .word   0x0000e000
                .space  32*4
                .word   0x00009011,0x00000000


                .bss    "sys_stk" (RW)
                .lcomm  tmp_area,16,4
                .lcomm  Sys_Stack,Stack_Size,4


                .text   "sys_text" (RX)
                .align  4

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

                ldpr    #0x9001,#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:   mov.w   r0,tmp_area
                mov.w   r1,tmp_area+4
                ldtask  4[r1],[r1]
                mov.w   tmp_area+4,r1
                mov.w   tmp_area,r0
                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
```

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

```
                .align  4
Sys_Trap:       mov.w   Current_Task,tmp_area
                mul.w   #24,tmp_area
                add.w   #TCB_Entry,tmp_area
                sttask  4[tmp_area]
                mov.w   Current_Task,r0
                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
                mov.w   #TCB_Entry,r1
                ldtask  4[r1](r0),[r1](r0)
Switch_Task:    retis   #4

Dummy_Text:     halt

                .data   "sharemem"      (RW)
_cmd_sem:       .word   0
_command:       .byte   0
                .align  4
_msg_sem:       .word   0
_message:       .space  0x20

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

                .bss    "stack_b"       (RW)
                .lcomm  Stack_B,Stack_Size,4
```

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

```
#define TRUE    1
#define FALSE   0
#define MSG_SIZ 0x20

#define trap(x) asm ("  trap #0xa0+(x)")
static char cmd;
static char msg_dest [MSG_SIZ];

main()
{
    clear_dest();
    while ( TRUE ) {
        for( cmd = ’A’; cmd <= ’C’; cmd++) {
           write_command ( cmd );
           read_message ( msg_dest );
        }
    }
}

clear_dest()
{
    int i;

    for ( i = 0; i < MSG_SIZ ; i++ )
        msg_dest[i] = ’ ’;
}

write_command ( cmd )
char cmd;
{
    extern char command;
    extern int  cmd_sem;

    while ( cmd_sem )
        trap(0);
    command = cmd;
    cmd_sem++;
}

read_message ( buf )
char *buf;
{
    extern char *message;
    extern int  msg_sem;
    int i;

    while ( !msg_sem )
        trap(0);
    for( i = 0; i < MSG_SIZ; i++ )
        buf [i] = message [i];
    msg_sem--;
}
```

Figure 3-2. Sample Program Source command.c

```
#define TRUE            1
#define FALSE           0

#define CMD_A           ’A’
#define CMD_B           ’B’
#define CMD_C           ’C’
#define NO_CMD          ’ ’
#define MSG_SIZ         0x20

#define trap(x) asm ("  trap #0xa0+(x)")

static char status[MSG_SIZ];
static char cmd_result[MSG_SIZ];

main ()
{
    char cmd_code;

    msgcpy (status, "Awaiting command", MSG_SIZ);
    msgcpy (cmd_result, "No command entered", MSG_SIZ);

    while(TRUE) {
        read_command ( &cmd_code );
        cmd_process ( cmd_code, cmd_result);
        write_message ( cmd_result );
    }
}

int cmd_process (cmd_code, cmd_result)
char cmd_code;
char *cmd_result;
{
    msgcpy (status, "Command received", MSG_SIZ);

    switch (cmd_code) {
    case CMD_A :
        msgcpy (cmd_result, "Command ’A’ entered", MSG_SIZ);
        break;

    case CMD_B :
        msgcpy (cmd_result, "Command ’B’ entered", MSG_SIZ);
        break;

    case CMD_C :
        msgcpy (cmd_result, "Command ’C’ entered", MSG_SIZ);
        break;

    default :
        msgcpy (cmd_result, "Invalid command entered", MSG_SIZ);
    }
}
```

Figure 3-3. Sample Program Source process.c

```
msgcpy( msg_dst, msg_src, msg_siz)
char    *msg_dst;
char    *msg_src;
int     msg_siz;
{
    for (; *msg_src != '\0' && msg_siz > 0; --msg_siz)
        *msg_dst++ = *msg_src++;

    for (; msg_siz > 0; --msg_siz)
        *msg_dst++ = ' ';
}

read_command ( cmd )
char *cmd;
{
    extern char command;
    extern int  cmd_sem;

    while ( !cmd_sem )
        trap(0);
    *cmd = command;
    cmd_sem--;
}

write_message ( buf )
char *buf;
{
    extern char *message;
    extern int  msg_sem;
    int i;

    while ( msg_sem )
        trap(0);
    for( i = 0; i < MSG_SIZ; i++ )
        message [i] = buf [i];
    msg_sem++;
}
```

Figure 3-3. Sample Program Source process.c (Cond'd)

```
SECTIONS
{
        sys_sbt         0x00000000: {}
        sys_tcb         0x00001000: {}
        sys_stk         0x00002000: {}
        sys_text        0x00003000: {}
        sharemem        0x00004000: {}
        stack_a         0x00005000: {}
        stack_b         0x00006000: {}
}
```

Figure 3-4. Linker command file os.lnk

```
SECTIONS
{
        command 0x40000000: {
                        command.o (.text)
                        command.o (.data)
                        command.o (.bss)
                        _cmd_sem = 0x00004000;
                        _command = 0x00004004;
                        _msg_sem = 0x00004008;
                        _message = 0x0000400c;
        } >(RWX)


}
```

Figure 3-5. Linker Command File command.lnk

```
SECTIONS
{
        process 0x40000000: {
                        process.o (.text)
                        process.o (.data)
                        process.o (.bss)
                        _cmd_sem = 0x00004000;
                        _command = 0x00004004;
                        _msg_sem = 0x00004008;
                        _message = 0x0000400c;
        } >(RWX)


}
```

Figure 3-6. Linker Command File process.lnk

```
SPACE(OS) 0x0  < {os}
SPACE(COMMAND) < {command}
SPACE(PROCESS) < {process}
```

Figure 3-7. Configurator Command File skdemo2.cfc

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 "*command* " 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 "*process*". The task number is "1".

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 (*command* task).

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 Area The "sharemem" section defines common area for both *command* task and *process* task. The common area is private buffer between these tasks.

command.c

The file *command.c* defines a command generator. *Command* task generates commands to *process* task. A command is an ASCII byte, and ’A’ through ’C’ are generated sequentially. Commands are delivered to *process* task via the common area.

When a command byte is interpreted by *process* task, resulting message is written in the common area. After the message is written, *command* task reads the message from the common area. The message is transferred to **msg_dest** location.

process.c

The file *process.c* defines a command interpreter. *Process* task checks whether a command is send from *command* task. When a command is generated by *command* task, *process* task interprets the command and output a message into the common area. If the command is one of the correct command (’A’ through ’C’), the corresponded message is written.

Compiling, Assembling and Linking the Sample Program

NEC Corporation CC70616 C Compiler Package is used to compile, assemble, and link the demo program. The package are available for use in the HP 9000 300 Series work stations from NEC.

The v70cnvhp utility is used to generate the required HP format files. Each file which has ".X" suffix contains the absolute code of the program. Each file which has ".L" suffix contains the list of global symbols. Each file which has ".A" suffix contains the list of local symbols. The symbol files for *os.s* contain real addresses of the symbols. The symbol files for *command.c* and *process.c* contain virtual addresses of the symbols. All the absolute files are generated for real address location.

The following commands are used to compile, assemble, and link the demo program.

```
$as70616 -a os.s >os.lis
$cc70616 -cg command.c
$cc70616 -cg process.c
$ld70616 -m -o os os.o os.lnk >os.map
$ld70616 -m -o command command.o command.lnk >command.map
$ld70616 -m -o process process.o process.lnk >process.map
$cf70616 -m -o skdemo2.cfo skdemo2.cfc >skdemo2.cfm
$ar70616 -x skdemo2.cfo
$v70cnvhp -r os.cf
$v70cnvhp -rx command.cf
$v70cnvhp -vla command.cf
$v70cnvhp -rx process.cf
$v70cnvhp -vla process.cf
```

The linker command files *os.lnk*, *command.lnk* and *process.lnk* used in the above command are shown in figure 3-4 through 3-6.

The configurator command file *skdemo2.cfc* is listed in figure 3-7.

The sample programs used in this chapter can be found in the following path:

/usr/hp64000/demo/emul/hp64758/*

Setting Up the Emulator

Before debugging, you have to set up the emulator by typing some commands. The details of these commands used below are mentioned in chapter 2.

Entering the Softkey Interface

From the "pmon" User Interface

If **/usr/hp64000/bin** is specified in your PATH environment variable, you can enter the **pmon** User Interface with the following command.

$ ***pmon*** <RETURN>

If the measurement system and emulation module are named "emv70" and "v70", you can enter the emulation system with the following command:

If you have not set up the measurement system or emulation module, set up the system or module. Refer to the "Entering the Softkey Interface" section of chapter 2.

emv70 default v70 <RETURN>

From the HP-UX Shell

If **/usr/hp64000/bin** is specified in your PATH environment variable, you can also enter the Softkey Interface with the following command.

$ ***emul700*** <emul_name> <RETURN>

The "emul_name" in the command above is the logical emulator name given in the HP 64700 emulator device table (/usr/hp64000/etc/64700tab).

Configuring the Emulator

To entering the emulator configuration session, enter the following command.

modify configuration <RETURN>

Trace the following answer to configure the emulator. Details of each question will be described later.

```
Micro-processor clock source? internal
Enter monitor after configuration? yes
Restrict to real-time runs? no
Modify memory configuration? yes
Monitor type? background
```

Now you should be facing memory mapping screen. The sample program occupies address range 0 through 9fffh of actual memory.

Delete the default mapping, and map the address range as emulation ram.

delete all
0 ***thru*** 9fffh ***emulation ram***
default guarded
end

```
Modify emulator pod configuration? no
Modify debug/trace options? no
Modify simulated I/O configuration? no
Modify interactive measurement specification? no
Configuration file name? skdemo2
```

Loading Absolute Files

Enter the following command to load the absolute files.

load os <RETURN>
load command <RETURN>
load process <RETURN>

The *v70cnvhp* converter also generated an absolute file which contains address translation tables for the sample program. The absolute file name is *aptable.X*. To load the file, specify **nosymbols** option because symbol files for *aptable.X* are not generated.

load aptable ***nosymbols*** <RETURN>

Loading the Symbols for os

The sample program is executed from the address **Sys_Init**. Load the symbols for *os* because the file *os.s* includes this label.

load symbols os <RETURN>

After loading symbol file, display the global symbols.

display global_symbols <RETURN>

```
Global symbols in os
Static symbols
Symbol name ____________________ Address range __ Segment _____________ Offset
Current_Task                   00001000                                  0000
Num_Of_Task                    00001004                                  0000
Setup_Task                     00003043                                  0000
Start_Ini_Task                 000030A0                                  0000
Switch_Task                    000030FF                                  0000
Sys_Init                       00003000                                  0000
Sys_SBT                        00000000                                  0000
Sys_Trap                       000030A4                                  0000
TCB_Entry                      00001008                                  0000
_esharemem_                    0000402C                                  0000
_estack_a                      00005FF0                                  0000
_estack_b                      00006FF0                                  0000
_esys_sbt                      000000C4                                  0000
_esys_stk                      00003000                                  0000
_esys_tcb                      00001150                                  0000

STATUS:   Build successful; no warnings were issued_____________________...R....
 display  global_symbols


  run     trace     step   display          modify   break     end    ---ETC--
```

Display the local symbols, include the source file name in which the symbols are defined.

display local_symbols_in
os.s:<RETURN>

```
Symbols in os.s:
Static symbols
Symbol name ____________________ Address range __ Segment _____________ Offset
Dummy_Text                     00003101                                  1101
Setup_Task_0                   00003051                                  1051
Stack_A                        00005000                                  0000
Stack_B                        00006000                                  0000
Sys_Stack                      00002010                                  0010
Sys_Trap_0                     000030E7                                  10E7
TCB_A                          00001038                                  0038
TCB_B                          000010C4                                  00C4
_cmd_sem                       00004000                                  0000
_command                       00004004                                  0004
_message                       0000400C                                  000C
_msg_sem                       00004008                                  0008
tmp_area                       00002000                                  0000



STATUS:   cws: os.s:____________________________________________________...R....
 display  local_symbols_in os.s:


  run     trace     step   display          modify   break     end    ---ETC--
```

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.

modify software_breakpoints enable <RETURN>

modify software_breakpoints set Start_Ini_Task <RETURN>

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

run from Sys_Init <RETURN>

You will see the following in the status line.

```
Software break: 00000030a0@v
```

Display memory in mnemonic format from the current PC.

display memory mnemonic <RETURN>

The next instruction to be executed is high-lighted. You can include symbols in the display.

```
 Memory  :mnemonic :file = os.s:
   address  label          data
  000030A0  :Start_Ini_T  FAE4        RETIS      #4H
  000030A2                00          HALT
  000030A3                00          HALT
  000030A4   :Sys_Trap    2D80F25CDF  MOV.W      :Current_Task,sk/os.s:tmp_area
  000030B0                8580F41800  MUL.W      #00000018H,sk/os.s:tmp_area
  000030BC                8480F40810  ADD.W      #00001008H,sk/os.s:tmp_area
  000030C8                FCFE38EFFF  STTASK     00000004H[sk/os.s:tmp_area]
  000030D2                2D20F22EDF  MOV.W      :Current_Task,R0
  000030D9                DD60        INC.W      R0
  000030DB                BC00F229DF  CMP.W      R0,:Num_Of_Task
  000030E2                6505        BNE/NZ     /os.s:Sys_Trap_0
  000030E4                B44060      XOR.W      R0,R0
  000030E7  o:Sys_Trap_0  2D00F219DF  MOV.W      R0,:Current_Task
  000030EE                8520E6      MUL.W      #6H,R0
  000030F1                2D21F40810  MOV.W      #00001008H,R1
  000030F8                01E0C00104  LDTASK     04H[R1](R0),[R1](R0)

STATUS:   N70632--Running in monitor      Software break: 0000030a0@v___...R....
 set  symbols  on


pod_cmd    set    perfinit perfrun         perfend                    ---ETC--
```

set symbols on <RETURN>

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 *command* task and *process* task.
- Switching to *command* task.

The emulator broke just before the transition from operating system to *command* task. Step one instruction to enter the *command* task.

step <RETURN>

The display is updated to disassemble from the current PC. The symbols for these addresses are included in the symbol file for *command* task. Load the symbols for *command* task.

load symbols command <RETURN>

The display will come to include the symbols.

Enter the following command to include source file in the display.

set source on <RETURN>

```
 Memory  :mnemonic :file = command.c:
   address  label          data
  40000000     :_main     ECF4000000  PUSHM      #0m<>
  40000006                DEF4000000  PREPARE    #00000000H
      7    static char msg_dest [MSG_SIZ];
      8
      9    main()
     10    {
     11        clear_dest();
  4000000C                4980F25C00  CALL       :_clear_dest,[SP]
     12        while ( TRUE ) {
     13         for( cmd = 'A'; cmd <= 'C'; cmd++) {
  40000014                0980F441F2  MOV.B      #41H,s/command.c:_cmd
  4000001D                B880F443F2  CMP.B      #43H,s/command.c:_cmd
  40000026                6F37        BGT        :_main+0000005DH
     14            write_command ( cmd );
  40000028                0CA0F20C01  MOVS.BW    s/command.c:_cmd,[-SP]
  40000030                4980F27C00  CALL       :_write_command,[SP]

STATUS:   N70632--Stepping complete_____________________________________...R....
 set  source  on


pod_cmd    set    perfinit perfrun          perfend                   ---ETC--
```

Define software breakpoint at the address **Switch_Task**. This address is the exit of the task dispatcher. The symbol **Switch_Task** in included in *os*.

Since you have loaded the symbols for *command*, you must reload the symbols for *os*.

load symbols os <RETURN>

Define software breakpoint at **Switch_Task**, and continue the execution.

modify software_breakpoints set Switch_Task <RETURN>

run <RETURN>

You will see the following in the status line.

```
Software break: 00000030ff@v
```

The processor executed the following tasks.

- Generating the command 'A'.
- Sending the command into the common area.
- Aborting the execution of *command*
- Storing the Task Context for *command*
- Loading the Task Context for *process*
- Switching to *process*

The emulator broke just before the transition from task dispatcher to *process*. Step one instruction to enter the *process* task.

step <RETURN>

The display is updated to disassemble from the current PC. The symbols for these addresses are included in the symbol file for *process* task. Load the symbols for *process* task.

load symbols process <RETURN>

```
 Memory  :mnemonic :file = process.c:
   address  label          data
  40000000     :_main     ECF4000000  PUSHM      #0m<>
  40000006                DEF4040000  PREPARE    #00000004H
      15    main ()
      16    {
      17        char cmd_code;
      18
      19        msgcpy (status, "Awaiting command", MSG_SIZ);
  4000000C                EEF4200000  PUSH       #00000020H
  40000012                EEF4380200  PUSH       #40000238H
  40000018                EEF4C40200  PUSH       #400002C4H
  4000001E                4980F22E01  CALL       :_msgcpy,[SP]
  40000026                843FEC      ADD.W      #CH,SP
      20        msgcpy (cmd_result, "No command entered", MSG_SIZ);
  40000029                EEF4200000  PUSH       #00000020H
  4000002F                EEF4490200  PUSH       #40000249H
  40000035                EEF4E40200  PUSH       #400002E4H

STATUS:   cws: process.c:_______________________________________________...R....
 load  symbols process


  load    store   stop_trc   copy           reset   specify  cmb_exec ---ETC--
```

Displaying Registers

Display basic registers by entering:

display registers <RETURN>

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

display registers PRIV <RETURN>

display registers MMU <RETURN>

```
Registers

Next PC 40000000@v
 PC 40000000  SP 00007000  FP 00000000  AP 00000000  PSW 00000000
 R0-7   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 R8-15  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 R16-23 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 R24-31 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00007000

  SBR 00000000    TR 000010C4  SYCW 00002171  TKCW 0000E000   PIR 00007006
  ISP 00003000  L0SP 00007000  L1SP 00000000  L2SP 00000000  L3SP 00000000
 PSW2 0000F002

 ATBR0 00009001  ATBR1 00009011  ATBR2 00000000  ATBR3 00000000
 ATLR0 00000000  ATLR1 00000000  ATLR2 00000000  ATLR3 00000000



STATUS:   N70632--Stepping complete_____________________________________...R....
 display  registers  MMU


  run     trace     step   display          modify   break     end    ---ETC--
```

Tracing the Program Execution

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

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 specify any trace specifications. Start the trace and continue the program.

trace <RETURN>
run <RETURN>

The status line shows that the emulation trace is completed.

To display the trace listing without fetch cycles, enter:

display trace compress on <RETURN>

The resulting trace is similar to the following display.

```
Trace List                   Offset=0
Label:      Address       Data    Opcode or Status w/ Source Lines
Base:       symbols       hex            mnemonic w/symbols
after       00009010    0000903B     0000903BH trans table  read
+001        00009014    000000FC     000000FCH trans table  read
+002        00009038    00008E05     00008E05H trans table  read
+003        00009038    00008E85     00008E85H trans table  write
+011        00009000    0000901B     0000901BH trans table  read
+012        00009004    000006FC     000006FCH trans table  read
+013        00009030    00006F85     00006F85H trans table  read
+014        00008000    00006F85   PUSHM     #0m<>
+015        00008006    00006F85   PREPARE   #00000004H
+016        00006FFC    00000000     00000000H data  write
+019        0000800C    3F847F00   PUSH      #00000020H
+021        00006FF4    00000020     00000020H data  write
+022        00008012    00000020   PUSH      #40000238H
+024        00006FF0    40000238     40000238H data  write
+026        00008018    000249F4   PUSH      #400002C4H

STATUS:   N70632--Running user program    Emulation trace complete______...R....
 display  trace  compress  on


  run     trace     step   display           modify   break     end    ---ETC--
```

The trace listing shows the beginning of the execution of *process* task, and now you can find that the address fields of the trace are displayed in real address. Regardless of address mode, addresses which the analyzer captures are real addresses by default.

Note

Since the symbols for *process* are generated in virtual address, you can not include the symbols in the trace listing even if you load the symbols for *process*. To include the symbols, you must trace virtual address or generate the symbol file in real address.

Specifying Virtual Space

The program executes *command* and *process* alternately. Suppose that you wish to note to *process* task. In this case, you should load the symbols for *process* and use the XMMU function.

Since you have loaded the symbols for *process* in previous section, you do not have to reload the symbols. Display the global symbols, enter:

display global_symbols<RETURN>

The global symbols for *process* are displayed.

```
Global symbols in process
Procedure symbols
Procedure name _________________ Address range __ Segment _____________ Offset
_cmd_process                     40000088 - 4000014B                      0088
_main                            40000000 - 40000087                      0000
_msgcpy                          4000014C - 400001A6                      014C
_read_command                    400001A8 - 400001D8                      01A8
_write_message                   400001DC - 40000234                      01DC

Static symbols
Symbol name ____________________ Address range __ Segment _____________ Offset
_cmd_sem                         00004000                                 0000
_command                         00004004                                 0000
_eprocess                        40000304                                 0000
_message                         0000400C                                 0000
_msg_sem                         00004008                                 0000

Filename symbols

STATUS:   N70632--Running user program    Emulation trace complete______...R....
 display  global_symbols


  run     trace     step   display          modify   break     end    ---ETC--
```

To display local symbols, select:

display local_symbols_in `process.c:`
<RETURN>

The resulting display follows.

```
Symbols in process.c:
Procedure symbols
Procedure name _________________ Address range __ Segment _____________ Offset
_cmd_process                  40000088 - 4000014B                          0088
_main                         40000000 - 40000087                          0000
_msgcpy                       4000014C - 400001A6                          014C
_read_command                 400001A8 - 400001D8                          01A8
_write_message                400001DC - 40000234                          01DC

Static symbols
Symbol name ____________________ Address range __ Segment _____________ Offset
_cmd_result                   400002E4                                     02E4
_status                       400002C4                                     02C4

Source reference symbols
Line range ______________________ Address range __ Segment _____________ Offset
#1-#19                        4000000C - 40000028                          000C
#20-#20                       40000029 - 40000045                          0029

STATUS:   cws: process.c:_______________________________________________...R....
 display  local_symbols_in process.c:


  run     trace     step   display          modify   break     end    ---ETC--
```

Using the XMMU Function.

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, enter:

display registers XMMU <RETURN>

The resulting display shows the contents of XMMU class registers. The display also includes the contents of on-chip MMU registers, you

```
Registers

 PC 40000000  SP 00007000  FP 00000000  AP 00000000  PSW 00000000
 R0-7   00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 R8-15  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 R16-23 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 R24-31 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00007000

  SBR 00000000    TR 000010C4  SYCW 00002171  TKCW 0000E000   PIR 00007006
  ISP 00003000  L0SP 00007000  L1SP 00000000  L2SP 00000000  L3SP 00000000
 PSW2 0000F002

 ATBR0 00009001  ATBR1 00009011  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______...R....
 display  registers  XMMU


  run     trace     step   display          modify   break     end    ---ETC--
```

displayed in previous section, and these values define virtual space for *process*.

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

> ***modify register XMMU XATBR0 to*** 9001h <RETURN>
>
> ***modify register XMMU XATBR1 to*** 9011h <RETURN>

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

> ***modify register XMMU MMUMOD to*** 1 <RETURN>

To display the contents of memory at address from **_main**.

Enter:

```
 Memory  :mnemonic :file = process.c:
   address  label          data
  40000000     :_main     ECF4000000  PUSHM      #0m<>
  40000006                DEF4040000  PREPARE    #00000004H
      15    main ()
      16    {
      17        char cmd_code;
      18
      19        msgcpy (status, "Awaiting command", MSG_SIZ);
  4000000C                EEF4200000  PUSH       #00000020H
  40000012                EEF4380200  PUSH       #40000238H
  40000018                EEF4C40200  PUSH       #400002C4H
  4000001E                4980F22E01  CALL       :_msgcpy,[SP]
  40000026                843FEC      ADD.W      #CH,SP
      20        msgcpy (cmd_result, "No command entered", MSG_SIZ);
  40000029                EEF4200000  PUSH       #00000020H
  4000002F                EEF4490200  PUSH       #40000249H
  40000035                EEF4E40200  PUSH       #400002E4H

STATUS:   Warning: no ENTRY/EXIT symbol; using TEXTRANGE________________...R....
 display  memory _main mnemonic


  run     trace     step   display          modify   break     end    ---ETC--
```

> ***display memory*** _main ***mnemonic*** <RETURN>

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. Use the **pod_command** to issue the Terminal Interface command.

To display the ATE corresponding with address **_main** (address 40000000H), use the **ate** command of the Terminal Interface.

Note that the Terminal Interface cannot accept any symbols.

display pod_command <RETURN>

pod_command 'ate 40000000h' <RETURN>

To display the PTE corresponding with address **_main** (address 40000000H), use the **pte** command of the Terminal Interface.

pod_command 'pte 40000000H' <RETURN>

```
Pod Commands
  Time                Command
     wait         - do not use, will tie up the pod, blocking access
     init, pv     - will reset pod and force end release_system
     t            - do not use, will confuse trace status polling and unload
 _____________________________________________________________________________

10:24:39 ate 40000000h

1:000 at 000009010  Present
      PTB=000009038  Limit=000  Growth=positive
      Execute level=3  Write level=3  Read level=3

10:24:44 pte 40000000h

1:000:000 at 000009038  Present
          Page base=000008000  Executable  Writable  Readable
          Modified  Accessed  User=0  Not locked

STATUS:   N70632--Running user program    Emulation trace complete______...R....
 pod_command 'pte 40000000h'
```

Breakpoints

Before defining the breakpoint, break the emulator by entering:

break <RETURN>

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

modify software_breakpoints set _cmd_process <RETURN>

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

run <RETURN>

The status line shows as follows.

```
Software break: 040000088@v
```

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 (*process*) is 7fffffffH, enter:

pod_command 'tcb -l 7fffffffh' <RETURN>

```
Pod Commands
  Time                Command
1:000:000 at 000009038  Present
          Page base=000008000  Executable  Writable  Readable
          Modified  Accessed  User=0  Not locked

10:26:12 tcb -l 7fffffffh

  tkcw ATT=7 OTM=0 FIT=0 FZT=0 FVT=0 FUT=0 FPT=0 RDI=0 RD=0
  l0sp=00006fdc
   r0=00000000   r1=0000001f   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=00006ff4
  r30=00006fe8
  atrp1 ATB=000009010  Limit=000  Growth=positive  Valid

STATUS:   N70632--Running in monitor      Software break: 040000088@v___...R....
 pod_command 'tcb -l 7fffffffh'


pod_cmd    set    perfinit perfrun          perfend                   ---ETC--
```

Tracing Virtual Address

The analyzer can capture virtual address by modifying configuration.

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

modify configuration <RETURN>

Press **Return** key until the "Modify debug/trace option?" question is displayed. Answer **yes** to entering the debug/trace configuration session. Press **Return** key until the "Trace virtual or real address?" question is displayed. Answer **virtual** to trace virtual address.

Press **Return** several times to exit the configuration session.

Specifying Trigger

To trace the program states after the execution of the address **_read_command**.

trace after _read_command ***status exec*** <RETURN>

The status line shows that the trace is started.

To continue the execution, enter:

run <RETURN>

The trace status changes to "Emulation trace complete".

To display the trace, enter:

display trace compress on <RETURN>

The resulting display shows the execution of the function **_read_command**.

```
Trace List                  Offset=0
Label:      Address       Data    Opcode or Status w/ Source Lines
Base:       symbols       hex            mnemonic w/symbols
after   :_read_command  F28009F7   No fetch cycle found
+002   :_read_+00000006 F28009F7   No fetch cycle found
+003        00006FE8    00006FFC     00006FFCH data  write
      ##########process.c - line    65 thru    68 #################

      read_command ( cmd )
      char *cmd;
      {
+005   :_read_+0000000C 64C0003E   No fetch cycle found
+010      :_cmd_sem     00000000     00000000H data  read
+011   :_read_+00000011 00000000   TEST.W    :_cmd_sem
+012   :_read_+00000017 00000000   BE/Z      :_read_command+0000000
      ##########process.c - line    69 thru    73 #################
          extern char     command;
          extern int cmd_sem;

STATUS:   N70632--Running user program    Emulation trace complete______...R....
 display  trace  compress  on


  run     trace     step   display          modify   break     end    ---ETC--
```

Press the **<PGDN>** or **<NEXT>** key to see more lines. Then you will see the transition from *process* task to the task dispatcher. Press the **<PGDN>** or **<NEXT>** key several times until the *command* task execution is displayed (The states of *command* task will be stored from line 175).

As you can see, some addresses are replaced with the symbols for *process* task (in this case, **_cmd_process**). You may confuse the states with the states of *process* task. The reason is because *command* and *process* occupy the same virtual address (not the same virtual space) each other.

```
Trace List                  Offset=0
Label:      Address       Data    Opcode or Status w/ Source Lines
Base:       symbols       hex            mnemonic w/symbols
+166        00009008    00009037     00009037H trans table  read
+167        0000900C    000000FC     000000FCH trans table  read
+168        00009034    00007F85     00007F85H trans table  read
+174      :_msg_sem     00000001     00000001H data  read
+175   :_cmd_p+00000065 00000001   TEST.W    :_msg_sem
+177   :_cmd_p+0000006B 1E202D22   BE/Z      :_cmd_process+00000062
+180   :_cmd_p+0000006D C0003F03   MOV.W     #0H,-04H[FP]
+182        00005FE8    00000000     00000000H data  write
+184   :_cmd_p+00000072 00000000   CMP.W     #00000020H,-04H[FP]
+185        00005FE8    00000000     00000000H data  read
+186   :_cmd_p+0000007B 00000000   BGE       :_cmd_process+0000009D
+189   :_cmd_p+0000007D 00000000   MOV.W     -04H[FP],R0
+190        00005FE8    00000000     00000000H data  read
+191      :_message     00000000     00000000H data  read
+193   :_cmd_p+00000081 000020F4   MOV.W     :_message,R1

STATUS:   N70632--Running user program    Emulation trace complete______...R....
 display  trace  compress  on


  run     trace     step   display          modify   break     end    ---ETC--
```

Load the suitable symbols for displaying the accurate symbols in the display.

load symbols command <RETURN>

```
Trace List                   Offset=0
Label:      Address       Data    Opcode or Status w/ Source Lines
Base:       symbols       hex            mnemonic w/symbols
+166        00009008    00009037     00009037H trans table  read
+167        0000900C    000000FC     000000FCH trans table  read
+168        00009034    00007F85     00007F85H trans table  read
+174      :_msg_sem     00000001     00000001H data  read
+175   :_read_+00000011 00000001   TEST.W    :_msg_sem
+177   :_read_+00000017 1E202D22   BE/Z      :_read_message+0000000
      ##########command.c - line    49 ############################
          for( i = 0; i < MSG_SIZ; i++ )
+180   :_read_+00000019 C0003F03   MOV.W     #0H,-04H[FP]
+182        00005FE8    00000000     00000000H data  write
+184   :_read_+0000001E 00000000   CMP.W     #00000020H,-04H[FP]
+185        00005FE8    00000000     00000000H data  read
+186   :_read_+00000027 00000000   BGE       :_read_message+0000004
      ##########command.c - line    50 ############################
           buf [i] = message [i];

STATUS:   Loaded symbol data base_______________________________________...R....
 load  symbols command


  load    store   stop_trc   copy           reset   specify  cmb_exec ---ETC--
```

Address Mode Option

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 select either real or virtual address by using the "**fcode**" option. To specify an address mode, add this option just before an address expression. The following options are allowed.

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

The following is an example usage of the fcode option.

display memory fcode v 4000000h
mnemonic <RETURN>

You can also designate addresses with no suffix. In this case, the address mode which is lastly specified by the **fcode** option is used to evaluate the addresses.

Until you specify an address mode by using the fcode option, the emulator use default address mode. The default address mode 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.

After you use the fcode option, if you wish to make the emulator to evaluate addresses in the default address mode, use the "**fcode none**" option.

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.

4

Configuring the 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 your target system software, or you can use the emulator in-circuit when integrating software with target system hardware. You can use the emulator's internal clock or the target system clock. Emulation memory can be used in place of, or along with, target system memory. 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 may be configured to suit your needs at any stage of the development process. This chapter describes the options available when configuring the HP 64758 emulator.

The configuration options are accessed with the following command.

modify configuration <RETURN>

After entering the command above, you will be asked questions regarding the emulator configuration. The configuration questions are listed below and grouped into the following classes.

General Emulator Configuration:

- Specifying the emulator clock source (internal/external).
- Selecting monitor entry after configuration.
- Restricting to real-time execution.

Memory Configuration:

- Selecting the background or foreground emulation monitor.
- Mapping memory.

Emulator Pod Configuration:

- Responding to /HLDRQ signal from target system.
- Responding to /NMI signal from target system.
- Responding to INT signal from target system.
- Responding to BFREZ signal from target system.
- Selecting target memory access data size.
- Driving background cycles to target system.
- Selecting value for address bus during background cycles.
- Selecting object file address attribute.

Debug/Trace Configuration:

- Enabling breaks on writes to ROM.
- Selecting tracing of foreground/background cycles.
- Enabling tracing bus hold cycles.
- Selecting tracing of real/virtual address.
- Enabling tracing execution cycles.

Simulated I/O Configuration: Simulated I/O is described in the Simulated I/O reference manual.

Interactive Measurement Configuration: See the chapter on coordinated measurements in the Softkey Interface Reference manual.

General Emulator Configuration

The configuration questions described in this section involve general emulator operation.

Micro-processor clock source?

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

internal

Selects the emulator's internal 20 MHz oscillator as the emulator clock source.

external

Selects the clock input to the emulator probe from the target system. You must use a clock input conforming to the specifications for the 70632 microprocessor. The maximum clock speed is 20 MHz.

Note

Changing the clock source drives the emulator into the reset state. The emulator may later break into the monitor depending on how the following "Enter monitor after configuration?" question is answered.

Enter monitor after configuration?

This question allows you to select whether the emulator will be running in the monitor or held in the reset state upon completion of the emulator configuration.

How you answer this configuration question is important in some situations. For example, when the external clock has been selected and the target system is turned off, reset to monitor should not be selected; otherwise, configuration will fail. When an external clock source is specified, this question becomes
"Enter monitor after configuration (using external clock)?" and the default answer becomes "no".

yes

When reset to monitor is selected, the emulator will be running in the monitor after configuration is complete. If the reset to monitor fails, the previous configuration will be restored.

no

After the configuration is complete, the emulator will be held in the reset state.

Restrict to real-time runs?

The "restrict to real-time" question lets you configure the emulator so that commands which cause the emulator to break to monitor and return to the user program are refused.

no

All commands, regardless of whether or not they require a break to the emulation monitor, are accepted by the emulator.

yes

When runs are restricted to real-time and the emulator is running the user program, all commands that cause a break (except "reset", "break", "run", and "step") are refused. For example, the following commands are not allowed when runs are restricted to real-time:

- Display/modify registers.
- Display/modify target system memory.
- Load/store target system memory

Refer to the "Target Memory Access" section of chapter 4, for more information.

Caution

If your target system circuitry is dependent on constant execution of program code, you should restrict the emulator to real-time runs. This will help insure that target system damage does not occur. However, remember that you can still execute the "**reset**", "**break**", and "**step**" commands; you should use caution in executing these commands.

Memory Configuration

The memory configuration questions allow you to select the monitor type and to map memory. To access the memory configuration questions, you must answer “yes” to the following question.

Modify memory configuration?

Monitor type?

The monitor type configuration question allows you to choose between a foreground monitor (supplied with the emulation software but must be assembled, linked, and loaded into emulation memory) or the background monitor (which resides in the emulator).

The *emulation monitor* is a program executed by the emulation processor. It allows the emulation system controller to access target system resources. For example, you may enter a command to display target system memory. This requires access to target system resources. The system controller writes a command code to the monitor communications area, breaking execution of the emulation processor from the user program into the monitor program. The monitor program then reads the command from the communications area and executes the 70632 instructions that read the contents of the target system memory locations. After the monitor has completed its task, execution returns to the user program.

The *background monitor*, resident in the emulator, offers the greatest degree of transparency to your target system (that is, your target system shouldn’t be affected by monitor execution). In some cases, you may require an emulation monitor tailored to the requirements of your system. Here, you will need to use a foreground monitor linked into your program modules. See the “Using the Foreground Monitor” appendix for more information on foreground monitors.

background

Selects the use of the built-in background monitor. A memory overlay is created and the background monitor is loaded into that area. You can use the emulator pod configuration questions listed below to specify how the emulator will drive the target system during background monitor execution.

- “Drive background cycles to target system?”
- “Value for address bits A31-A8 during background cycles?”

When you select the background monitor and the current monitor type is “foreground”, you are asked the following question.

Reset map (change of monitor type requires map reset)?

This question must be answered “yes” to change the monitor type.

foreground

Specifies that a foreground monitor will be used. Foreground monitor programs are shipped with the Softkey Interface (refer to the “Using the Foreground Monitor” appendix). When you select a foreground monitor, you are asked additional questions.

Reset map (change of monitor type requires map reset)?

This question must be answered “yes” or else the foreground monitor is not selected. This question is asked any time the foreground monitor is selected.

Monitor location for real address?

The default configuration specifies a monitor address of 00000000H. The monitor base address must be located on a 4 Kbyte boundary; otherwise, configuration will fail. Specify the real memory location of foreground monitor.

When using the foreground monitor in virtual mode, you must also answer the next question ("Monitor location for the virtual address").

Monitor location for virtual address?

Specify the virtual memory location of the foreground monitor. The default configuration specifies a monitor virtual address of 00000000H. The monitor base address must be located on a 4 Kbyte boundary; otherwise, configuration will fail.

When using the foreground monitor only in real mode, you may not answer this question.

Refer to the "Using the Foreground Monitor" appendix for more information.

Mapping Memory

The default emulator configuration maps locations 0-0FFFFH as emulation RAM. If your programs occupy locations outside this address range or in target system memory, you must modify the memory map.

The memory map specifies the location and type of various memory regions used by your programs and your target system (whether or not it exists). The memory map is necessary for several reasons:

- The emulator must know whether a given memory location resides in emulation memory or in target system memory. The emulator then orients the buffers for the data transfer.
- The emulator needs to know the size of any emulation memory blocks so it can properly reserve emulation memory space for those blocks.
- The emulator must know if a given space is RAM (read/write), ROM (read only), or does not exist. This allows the emulator to decide if certain actions taken by the

emulation processor are proper for the memory type accessed. For example, if the processor tries to write to an emulation memory location mapped as ROM, the emulator will not permit the write (though the memory at the given location is RAM). You can optionally configure the emulator to break to the monitor upon such occurrence. See the “Break processor on write to ROM?” debug/trace configuration question. Target memory locations will be overwritten if they are actually RAM but mapped as ROM. Also, if the emulation processor attempts to access a non–existent location (known as “guarded”), the emulator will break to the monitor.

The HP 64758G emulator contains 510 kilobytes of emulation memory, which can be mapped at a resolution of 4 Kbytes.

The HP 64758H emulator contains 1020 kilobytes of emulation memory, which can be mapped at a resolution of 4 Kbytes.

The memory mapper allows you to define up to 8 different map terms. You can specify one of five different memory types (target rom, target ram, emulation rom, emulation ram, or guarded).

For example, to map memory location 10000H through 1FFFFH as emulation ram, enter the following command.

`10000h` ***thru*** `1ffffh` ***emulation ram***
`<RETURN>`

If you wish to remove a mapper term, use the "delete" command. You can delete the mapper term numbered "1", enter the following command.

delete `1 <RETURN>`

If you want to remove all memory mappings, enter the following command.

delete all `<RETURN>`

By default, the emulation memory access operated with no-wait-state. If you are using the emulator in in-circuit mode, you can configure emulation memory location to honor target system ready signals. To respond to the target system ready signals while emulation memory is being accessed, add "**lock**" attribute as follows.

```
10000h thru 1ffffh emulation ram
lock <RETURN>
```

When accessing the emulation memory located at address 10000h thru 1ffffh, the target system ready signals will be referred in order to insert the wait states.

Note

You should map all memory ranges used by your programs **before** loading programs into memory. This helps safeguard against loads that accidentally overwrite earlier loads if you follow a **map/load** procedure for each memory range.

Emulator Pod Configuration

To access the emulator pod configuration questions, you must answer “yes” to the following question.

Modify emulator pod configuration?

Enable responding to HLDRQ signal?

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

yes

Accept Hold Request from target system. The /HLDRQ signals are driven from the target system to the emulator. The emulator will respond in the same manner as they would respond if the CPU were present.

no

Ignore Hold Request from target system. The /HLDRQ signals are not driven from the target system to the emulator. The emulator will not drive an active level on the address, data and control signals will not be placed in a tristate condition.

Enable /NMI input from target system?

This configuration allows you to specify whether or not the emulator responds to NMI signals from the target system during foreground operation.

yes

The emulator will respond to NMI signals from the target system.

no

The emulator will not respond to NMI signals from the target system.

Respond to target system interrupts?

This configuration allows you to specify whether or not the emulator responds to interrupt signals from the target system during foreground operation.

yes

The emulator will respond to interrupt signals from the target system.

no

The emulator will not respond to interrupt signals from the target system.

Respond to target bus freeze signal?

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

yes

Accept Bus Freeze Signals from target system. The BFREZ signals are driven from the target system to the emulator. The emulator will respond in the same manner as they would respond if the CPU were present.

no

Ignore Bus Freeze Signals from target system. The BFREZ signals are not driven from the target system to the emulator. The emulator will not drive an active level on the address, data and control signals will not be placed in a tristate condition.

Target memory access size?

This question allows you to specify the types of cycles that the emulation monitor use when accessing target system memory. When an emulation command requests the monitor to read or write target system memory locations, the monitor will either use byte or word instructions to accomplish the read/write.

bytes

Specifies that the emulator will access target system memory by byte accesses.

half_words

Specifies that the emulator will access target system memory by half word (2 bytes) accesses.

words

Specifies that the emulator will access target system memory by word (4 bytes) accesses.

Drive background cycles to target system?

This question allows you to specify whether the emulator will drive the target system bus on all background monitor cycles.

If you have chosen to use a foreground monitor, emulator foreground monitor cycles will appear at the target interface exactly as if they were bus cycles caused by any target system program.

yes

Specifies that background cycles are driven to the target system. The emulation processor's address, data and control strobes are driven during background cycles.

The value driven on the upper 24 bits (A31-A8) of the address bus is selected by the "Value for address bits A31-A8 during background cycles?" question.

When background cycles are driven to the target system, background write cycles appear as read cycles to the target system.

Use the "drive background cycles" option to avoid target system interaction problems. For example, your target system memory refresh scheme may depend on the constant repetition of bus cycles. Or, you may be using a watchdog timer in your target system, which resets the system when no bus cycles occur in a specified period. Driving background cycles to the target system will help avoid problems in either case.

no

Background monitor cycles are not driven to the target system. The emulator will appear to the target system as if it is between bus cycles while it is operating in the background monitor.

Value for address bits A31-A8 during background cycles?

This configuration question allows you to specify what memory address will be driven to the target system on address lines A31-A8 during emulation background monitor accesses. These lines will only be driven if you have specified that the emulator drive background cycles to the target system. See the previous “Drive background cycles to target system” question.

If you choose to use a foreground monitor, this configuration option is still valid. The emulation processor executes a few bus cycles in the background monitor before the transition to the foreground monitor.

Object file address attribute?

This configuration item allows you to specify whether the emulator should load absolute files into virtual address or real address when you use the load command. In other words, you can specify that in which address space the address location information are recorded in the absolute files. The default virtual address are used to translate the location address to actual memory address.

real

The emulator interprets the location address information in the absolute files as real address.

vir

The emulator interprets the location address information in the absolute files as virtual address.

Debug/Trace Configuration

The debug/trace configuration questions allow you to specify breaks on writes to ROM and whether the analyzer should trace foreground or background execution. To access the trace/debug configuration questions, you must answer “yes” to the following question.

Modify debug/trace options?

Break processor on write to ROM?

This question allows you to specify that the emulator break to the monitor upon attempts to write to memory space mapped as ROM. The emulator will prevent the processor from actually writing to memory mapped as emulation ROM. It cannot prevent writes to target system RAM locations mapped as ROM, though the write to ROM break is enabled.

yes

Causes the emulator to break into the emulation monitor whenever the user program attempts to write to a memory region mapped as ROM.

no

The emulator will not break to the monitor upon a write to ROM. The emulator will not modify the memory location if it is in emulation ROM.

Note

The **wrrom** trace command status option allows you to use "write to ROM" cycles as trigger and storage qualifiers. For example, you could use the following command to trace about a write to ROM:

trace about status wrrom <RETURN>

Trace background or foreground operation?

This question allows you to specify whether the analyzer trace only foreground emulation processor cycles, only background cycles, or both foreground or background cycles. When background cycles are stored in the trace, all but mnemonic lines are tagged as background cycles.

foreground

Specifies that the analyzer trace only foreground cycles. This option is specified by the default emulator configuration.

background

Specifies that the analyzer trace only background cycles. This is rarely a useful setting for user program debugging.

both

Specifies that the analyzer trace both foreground and background cycles. You may wish to specify this option so that all emulation processor cycles may be viewed in the trace display.

Trace HOLD tag?

You can direct the emulator to send HOLD cycle data to emulation analyzer or not to send it.

yes

When you enable tracing HOLD cycles, these cycles will appear as one analysis trace line.

no

HOLD cycles will not appear on analysis trace list.

Trace virtual or real address?

This configuration item allows you to specify whether analyzer should trace virtual address or real address.

real

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

vir

The analyzer captures virtual address. The trace listing shows the logical addresses executed by the processor.

Enable the execution cycles trace?

The emulation analyzer can capture states of instruction executions in addition to processor bus activity. By default, the emulation analyzer captures execution states. In this case, the analyzer can count neither time between states nor occurrence of bus states.

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. The maximum trace depth is 512 because of counting time or states.

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

Simulated I/O Configuration

The simulated I/O feature and configuration options are described in the *Simulated I/O* reference manual.

Interactive Measurement Configuration

The interactive measurement configuration questions are described in the chapter on coordinated measurements in the *Softkey Interface Reference* manual.

Saving a Configuration

The last configuration question allows you to save the previous configuration specifications in a file, which can be loaded into the emulator later.

Configuration file name? <FILE>

The name of the last configuration file is shown. No filename is shown if you are modifying the default emulator configuration.

If you press <RETURN> without specifying a filename, the configuration is saved to a temporary file. This file is deleted when you exit the Softkey Interface with the **end release_system** command.

When you specify a filename, the configuration is saved in two files. The file with the ".EA" extension is the "source" copy of the file, and the file with the ".EB" extension is the "binary" or loadable copy of the file.

Exiting emulation (with the **end** command) saves the current configuration, including the name of the most recently loaded configuration file, into a "continue" file. The continue file is not normally accessed.

Loading a Configuration

Previously saved configuration files may be loaded with the following Softkey Interface command.

load configuration <FILE> <RETURN>

This feature is especially useful after you have exited the Softkey Interface with the **end release_system** command. You won't have to modify the default configuration and answer all the questions again.

To reload the current configuration, you can enter the following command.

load configuration <RETURN>

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 70632 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
- Features available via "pod_command"
- Register names and classes
- 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 **r** (run) 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 as follows.

- 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 depends 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 **freg** command. The IEEE-754 standard data type is supported. To access to the general purpose registers in floating point format, use the folowing register names with the **FLOAT** attribute.

- **FR0** thru **FR31** for short real (32 bits floating point)
- **FRP0** thru **FRP30** for long real (64bits floating point)

To display all general purpose registers in short real format, enter:

display registers FLOAT <RETURN>

You can specify register to be displayed (for example, display R0 in short float format).

display registers FLOAT FR0 <RETURN>

To display two consecutive registers R0 and R1 in long real format, enter:

display registers FLOAT FRP0 <RETURN>

Modify register R0 to the value 12345.678, by typing:

modify register FLOAT FR0 to 12345.678 <RETURN>

Analyzer Topics

Analyzer Status Qualifiers

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

```
fetch          0x1xxxxxxxxxx011x     code fetch
brfetch        0x1xxxxxxxxxx0111     code fetch after branch
read           0x1xxxxxxxxxxxxxx     read
write          0x0xxxxxxxxxxxxxx     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
tbl            0xxxxxxxxxxxx0101     translation table access (read/write)
coproc         0xxxxxxxxxxxx1000     co-processor access(read/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
block          0xxxxxxxx0xxxxxxx     bus lock
retry          00xxxxxxxxxxxxxxx     retry
holdtag        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 corresponded to line 43 of the program skdemo.c was executed. Then the following command was issued to specify trigger condition.

trace after skdemo.c: line 43 ***status exec*** <RETURN>

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

If you had issued the following command instead, unexpected trigger would have occurred at the prefetch state of the instruction.

trace after skdemo.c: line 43<RETURN>

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 trigger condition as follows.

trace about `20018h` ***status exec*** `<RETURN>`

If you would type the following, the trigger will always occur at the prefetch of the address 20018H whether or not the branch condition at address 20016H is satisfied.

trace about `20018h <RETURN>`

Execution States Location in Trace Listing

The emulation analyzer stores execution states of the program in addition to actual bus cycles, if configuration "Enable the execution cycles trace?" question is answered "yes" (default).

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 examples that the execution states, numbered 64, fall behind its bus activity.

```
+061   00003004  00001e05     00001e05H trans table  read
+062   00003004  00001e85     00001e85H trans table  write
+063   00001004  00000002     00000002H data  read
+064   00005043  00000002   MOV.W     00001004H,R0
+065   0000504a  00000002   MOV.W     #00001008H,R1
+066   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 the address 1000H. You should not specify the trigger condition like this.

trace after `1000h` ***data*** `41h<RETURN>`

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

trace after `1000h` ***data*** `0xxxxxx41h`**<RETURN>**

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 enter:

trace after `1001h` ***data*** `0xxxx41xxh<RETURN>`

In case of halfword or word access to the data bus, it will be more complex, if two bus states are required to access the data because the data is across 4 byte boundary.

In this case, you need to use the analyzer sequential trigger capabilities. We do not describe the detail about the sequential trigger feature. Only how to trigger the analyzer at some example cases is described in this section.

To trigger the analyzer when the processor accesses the word data 12345678H at the address 1003H. The data bus activity of this cycles will be as follows.

```
Sequencer level    Address bus        Data bus
           1        00001003           78xxxxxx
           2        00001004           xx123456
```

To specify the trigger condition, enter:

trace find_sequence `1003h` ***data*** `78xxxxxxh` ***restart status exec trigger after*** `1004h` ***data*** `0xx123456h<RETURN>`

The "restart" condition is specified to restart sequencer when any states except for exec state are generated between sequencer level 1 and 2.

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 "Enable the execution cycles trace?" question of the Debug/Trace configuration. Refer to the "" 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.

Specify the trigger condition that the analyzer is never triggered.

trace before not range `0` ***thru*** `0ffffffffh<RETURN>`

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

stop_trace`<RETURN>`

display trace`<RETURN>`

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 break when the analyzer trigger condition is satisfied, use the "break_on_trigger" trace option.

Additionally, you can see the program states before the breakpoint in trace listing. Specify the trigger position at the end of trace listing by using "before" option.

When the trigger condition is found, emulator execution will break into the emulation monitor. Then you can also see the trace listing mentioned above, enter the following commands.

trace before <QUALIFIER>
break_on_trigger<RETURN>

Without the trigger condition, the trigger will never occur and will never break.

Example Configuration for Hardware Breakpoints Features.

The following are example configurations for typical break conditions you will use.

Breaks on Executing an Instruction

If you wish to break the execution when an instruction is executed. To specify the breakpoint when the instruction at address 12345678H is executed.

trace before 12345678h ***status exec***
break_on_trigger<RETURN>

Breaks on Accessing an Address

If you wish to break the execution when a certain data is written to a certain memory location. To specify the breakpoint when the halfword data 0abcdH is written to the address 87654321H.

trace before 87654321h ***data*** 0xxabcdxxh ***status write break_on_trigger***<RETURN>

The detail of analyzer data specification in the trigger condition is described in "Specifying Data for Trigger Condition or Store Condition" part of this section.

Breaks on 70632 Exceptions

In case that you test a simple program which does not have exception handler, you want to break the emulator on a 70632 exception. It is useful to specify the breakpoint when a 70632 exception is occurred.

There are two way to detect the 70632 exceptions as follows.

- Detect the states of the System Base Table Access at Events.

  To specify the breakpoint when the system base table access occurs by an event (exception or interrupt), enter:

trace before status sysbase break_on_trigger<RETURN>

- Detect the states of the Address Range of System Base Table.

  To specify the breakpoint when the address range of the system base table access occurs (except for Software Trap and Maskable Interrupt), enter:

trace before range 0 ***thru*** 0bfh ***break_on_trigger***<RETURN>

If the program to be tested uses the 70632 Software Trap or Maskable Interrupt or any other trap or exceptions on purpose, use the method of "Detect the System Base Table Access".

If the program to be tested accesses the 70632 system base tables which pointed at the SBR register on purpose, use the method of "Detect the Address Range of System Base Table".

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.

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.

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.

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).

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.

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.

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:

reset, ***run***, ***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

Displaying Area Table Entry and Page Table Entry is supported via Terminal Interface **ate** and **pte** commands. These commands are useful to examine in which address space the program are executed, and detect the address translation error of the program. Refer to the "Features Available via Pod Commands" section in this chapter for using Terminal Interface commands. Refer to the "70632 Emulator Terminal Interface User’s Guide" for these commands.

Making Coordinated Measurements

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

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

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

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

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.
- 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.

To display memory contents in 70108/70116 mnemonic, add the "**options v20_30**" option as follows.

display memory `1000h` ***mnemonic***
options v20_30 `<RETURN>`

To display memory contents in 70632 mnemonic, add the "**options default**" option.

When you specify the disassembler by using one of these options, the specified disassembler becomes the current disassembler.

If you do not specify neither option, the current disassembler is used to disassemble the memory.

Note

When you single-step an instruction, the current disassembler is used to display the mnemonic of the instruction which has been single-stepped in the register window.

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 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 "**Restrict to real-time runs?**" configuration, the caches will keep the values while you do not break the emulator into the monitor intentionally. Only when you issue **break**, **step** or **reset** 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 **register** command or the Terminal Interface **cpmmu** (copy current virtual address space to XMMU registers) command. See also the "Using the XMMU function" section of chapter 3. For the "**cpmmu**" command, refer to the "Features Available via Pod Commands" section in this chapter and **cpmmu** syntax in the *70632 Emulator Terminal Interface User's Guide" manual.*

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.

modify register MMUMOD to `1<RETURN>`

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

modify register MMUMOD to `0<RETURN>`

Features Available via Pod Commands

Several emulation features available in the Terminal Interface, but not in the Softkey Interface, may be accessed via the following emulation commands.

display pod_command `<RETURN>`

pod_command `'<Terminal Interface command>' <RETURN>`

Some notable Terminal Interface features not available in the softkey Interface are:

- Copying memory.
- Searching memory for strings or numeric expressions.
- Sequencing in the analyzer.
- Performing coverage analysis.
- Displaying Address Translation Tables (**ate** and **pte**).
- Displaying TCB (**tcb**).
- Fixing Virtual Space (**cpmmu**).

Refer to your Terminal Interface documentation for information on how to perform these tasks.

Note

Be careful when using **pod_command**. The Softkey Interface, and the configuration files in particular, assume that the configuration of the HP 64700 pod is NOT changed except by the Softkey Interface. What you see when using **modify configuration** will *not* reflect the HP 64700 pod's configuration if you change the pod's configuration with **pod_command**. Also, commands that affect the communications channel should *not* be used at all. Other commands may confuse the protocol depending upon how they are used. The following commands are *not recommended* for use with **pod_command**:

stty, **po**, **xp -** Do not use, will change channel operation and hang.
echo, **mac -** Usage may confuse the protocol in use on the channel.
wait - Do not use, will tie up the pod, blocking access.
init, **pv -** Will reset pod and force end release_system.
t - Do not use, will confuse trace status polling and unload.

Register Names and Classes

The following register names and classes may be used with the "**display/modify registers**" commands.

BASIC

Register Name	Description
R0 thru R31	All basic registers.
AP FP SP PC PSW SYCW	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	Area Table Register Pairs
ATBR1 ATLR1	
ATBR2 ATLR2	
ATBR3 ATLR3	

DEBUG (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.

OTHER

FR0 thru FR31 **FRP0 thru FRP30**	These register names are for display/modification of the registers in floating-point format. Each register name FRPn is corresponded to the two consecutive register (FRn and FRn+1). You can specify the "**FLOAT**" attribute to display/modify the registers in floating-point format. If you do not specify the "**FLOAT**" attribute, the contents of the registers are displayed or modified in hexadecimal format.

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.

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.

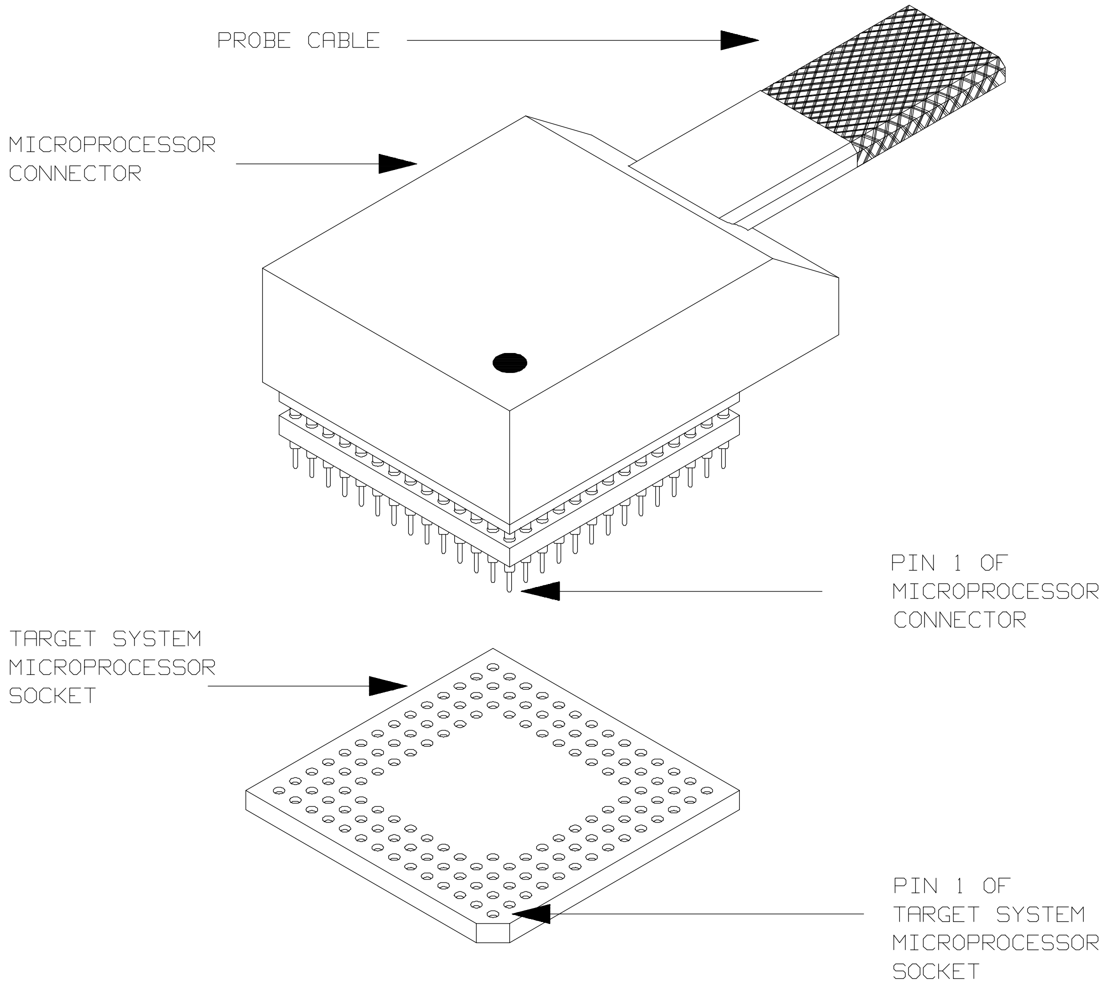

Figure 6-1. Installing Emulation Probe Into PGA Socket

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, add the "**lock**" attribute for emulation memory type.

When the ready relationship is locked to the target system by using the "**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 "**display/modify io_port**" 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 "**display/modify memory**" 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

Using the Foreground Monitor

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

Comparison of Foreground and Background Monitors

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

Background Monitors

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

However, all background monitors sacrifice some level of support for the target system. For example, when the emulation processor enters the monitor code to display registers, it will not respond to target system interrupt requests. This may pose serious problems for complex applications that rely on the microprocessor for real-time, non-intrusive support. Also, the background monitor code resides in emulator firmware and can't be modified to handle special conditions.

Foreground Monitors

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

You can tailor the foreground monitor to meet your needs, such as servicing target system interrupts. However, the foreground monitor does use part of the processor's address space, which may cause problems in some target systems. You must also properly configure the emulator to use a foreground monitor.

Foreground Monitor Selection

The HP 64758 emulator provides two kinds of foreground monitor. One is included in the emulator, the other is provided with assembler source file.

The foreground monitor included in the emulator allows you to use the foreground monitor quickly. When you use this built-in foreground monitor, you do not have to assemble, link and load the monitor program.

The foreground monitor provided with assembler source file allows you to customize the foreground monitor as you desire. When you use this custom foreground monitor, you need to assemble, link and load the monitor program.

Using Built-in Foreground monitor

The 70632 emulator includes foreground monitor. The built-in foreground monitor saves your tasks for assembling, linking and loading the monitor. To use the built-in foreground monitor, all you have to do is to specify the location of the monitor. The location is specified by the "Monitor Location for real address?" configuration.

Specify the monitor location (real address) as follows.

modify configuration <RETURN>

```
Modify memory configuration ? yes
Monitor type? foreground
Reset map (change monitor type requires map reset)? yes
Monitor location for real address? <real_address>
```

When your application is executed in virtual mode, you should also specify the virtual memory location for the monitor. The address translation tables for the monitor must be set up.

```
Monitor location for virtual address? <virtual_address>
```

If you do not use the emulator in virtual mode, you do not have to answer the "Monitor location for virtual address?" configuration question.

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

Interrupt/Exception Handler

The foreground monitor supports interrupt/exception handler. The interrupt/exception handler allows you to break the emulator into monitor when a certain interrupt or exception is generated.

After you exit the configuration session, six equation label pairs are defined in the Terminal Interface. These equation label pairs contain the entry addresses of the handlers, which are included in the foreground monitor. One of each equation label pair contains real address of the entry, the other (which has "V" prefix) contains virtual address of the entry. The description of these equation label pairs are as follows.

real address entry,	virtual address entry,	description
NMI_ENTRY	VNMI_ENTRY	NMI handler entry
INT_ENTRY	VINT_ENTRY	INT handler entry
EXC1_ENTRY	VEXC1_ENTRY	3 words stacking Exception handler entry
EXC2_ENTRY	VEXC2_ENTRY	4 words stacking Exception handler entry
STEP_ENTRY	VSTEP_ENTRY	Single-Step Trap handler entry
BRK_ENTRY	VBRK_ENTRY	Breakpoint Instruction Trap handler entry

Either of each equation label pair can be used so that vectors in system base table point to the corresponded handlers, if desired. The system base table must be defined in your program. For using single-step and software breakpoint features the single-step trap and breakpoint instruction trap handler entries must be set up.

For example, if you wish to use the emulator’s single-step feature, you must define the single-step trap handler entry in the corresponded vector table.

```
pod_command ’m -dd 30=STEP_ENTRY’
<RETURN>
```

If you use the single-step feature in virtual mode, you should have entered the following command instead.

```
pod_command ’m -dd 30=VSTEP_ENTRY’
<RETURN>
```

Refer to the *HP64758 70632 Emulator Terminal Interface User’s Guide*.

According to the system base table location, you may have to change the address (in this case, 30H) to be modified.

Using Custom Foreground monitor

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

The foreground monitor is supplied with the emulation software and can be found in the following path:

/usr/hp64700/monitor/*

The monitor program is named **Nfmon70632.s**

The monitor program is provided with HP 64758 emulator. You should modify the following statement of the monitor program to specify the monitor location.

```
.text   "FG_MON" > 0x000000000
```

The default monitor location is defined at address 000000000 (hex).

To tell the monitor location to the emulator, you should specify the monitor location (real address) by entering the configuration session.

modify configuration <RETURN>

In the configuration session, answer as follows.

```
Modify memory configuration ? yes
Monitor type? foreground
Reset map (change monitor type requires map reset)? yes
Monitor location for real address? <real_address>
```

When your application is executed in virtual mode, you should also specify the virtual memory location for the monitor. The address translation tables for the monitor must be set up.

```
Monitor location for virtual address? <virtual_address>
```

If you do not use the emulator in virtual mode, you do not have to answer the "Monitor location for virtual address?" configuration question.

After you exit the configuration session, you must load the monitor program into the emulator. The memory for the foreground monitor is already mapped when configuring the monitor location.

Interrupt/Exception Handler

The foreground monitor supports interrupt/exception handler. The interrupt/exception handler allows you to break the emulator into monitor when a certain interrupt or exception is generated.

In the foreground monitor program, some entry labels of the handlers are defined. See the monitor program for these entry labels. Write these labels in your program's system base table description. When you link the foreground monitor with your program, these labels will be referred by your program. The system base table must be defined in your program.

To use the single-stepping and/or software breakpoints feature(s), you must define the single step trap vector and/or the breakpoint instruction trap vector into the system base table. When you use these features in virtual mode, you must set up these vectors to point to their handler's entry in the foreground monitor in virtual address.

Even if you link the monitor with your program, you should also prepare the absolute file separated from user program to load the monitor program.

Loading Foreground Monitor

To load the monitor program, enter the following command; whether or not the monitor program is linked with your program.

load fg_mon <foreground_monitor> <RETURN>

The "**fg_mon**" option was used to load the foreground monitor program. You should specify the file name of the foreground monitor absolute separated from your program. After loading the monitor, map the memory for your program and load your program into the emulator.

Loading User Program

To load your program into target memory and emulation memory, do the following.

Loading into Target Memory

To load the program into target memory, enter the following commands.

break <RETURN>
load user_mem <user_program> <RETURN>

The **break** command causes the emulator to break into the monitor. For loading into target memory, the emulator must be running in monitor.

The "**user_mem**" option specify to load only target memory portion of the program.

Loading into Emulation Memory

To load the program into emulation memory, enter the following commands.

reset <RETURN>
load emul_mem <user_program> <RETURN>

The **reset** command causes the emulator to reset. For loading into emulation memory (which includes monitor program portion), the emulator must be reset.

The "**emul_mem**" option specifies to load only emulation memory portion of the program.

Restrictions and Considerations

When using the foreground monitor, there are some restrictions and considerations.

Cannot Single-step the Instruction RETIS and RETIU

The foreground monitor cannot step the RETIS and RETIU instruction. If you step either the RETIS or RETIU instruction, the emulator cannot break into monitor. As a result, the emulator runs your program without stepping.

Two Pages for the Monitor Program Must be Set Up

When you use the foreground monitor in virtual mode, the address translation tables for the foreground monitor must be set up. The monitor occupies one page (4 Kbytes memory), and further, one more page is required for accessing to target memory. In virtual mode, when accessing to target memory, the monitor modifies the page table to point to the target memory to be accessed to. The page must follows the foreground monitor page. For this reason, you must set up the address translation tables of two pages for the foreground monitor.

Monitor Must be Located at the Same Virtual Address Always.

The foreground monitor must be located at the same virtual address whenever virtual space is changed. This allows the emulator to break into monitor in any virtual space.

An Example Configuration of the Foreground Monitor

In the following example, we will illustrate how to set up the emulator to use the custom foreground monitor in virtual mode.

For this example, we will locate the monitor at 400000000h (virtual) and 1000h (real).

Modify Monitor Source Program

To use the monitor, you must modify the following statement near the top of the monitor program. In this example, the monitor will be located at 40000000h in virtual.

```
.text   "FG_MON" > 0x40000000
```

Defining System Base Table in Your Program

To use the single-step and software breakpoint feature of the emulator, you must define the single-step trap and breakpoint instruction trap vector into the system base table. Assuming that the system table description in **your program** as follows.

```
.data "sys_base"
.word  .....       -- + 00
.word  .....       -- + 04
.word  NMI_ENTRY   -- + 08
.word  .....       -- + 0C
         :
         :
.word  STEP_ENTRY  -- + 30
.word  BRK_ENTRY   -- + 34
         :
         :
```

The NMI_ENTRY label is also defined to break the emulator into monitor when NMI signal is generated.

Defining Address Translation Tables for Monitor Program

The following statements define two page tables for monitor program. The real address location of label PTE_FGMON must be pointed by the Area Table Entry of Section 1, Area 0 because the monitor location is 40000000h (virtual).

```
PTE_FGMON:    .word   0x00001e05  -- for foreground monitor location
              .word   0x00001e05  -- for accessing to target memory by monitor
```

The PTE in the second line must be defined to access to target memory by monitor program. The monitor modifies the PTE to point to target memory location to be accessed. Initially, the PTE had better point to the foreground monitor location.

Note that the foreground monitor must be reside in the fixed virtual address, even if virtual space is changed. This allows the emulator to break into monitor in any virtual space.

Assembling and Linking the Foreground Monitor

To refer to these labels (in this example, NMI_ENTRY, STEP_ENTRY and BRK_ENTRY), the foreground monitor program and your program should be linked together. Suppose that the generated absolute file name is "usr_prog.X".

You must prepare another absolute file which contains only foreground monitor program. The absolute file will be used to load the monitor program into the emulator. Suppose that the generated absolute file name is "Nfmon70632.X".

Setting Up the Monitor Configuration Item

The following configuration should be required to tell the use of foreground monitor and the location of the monitor to the emulator.

modify configuration `<RETURN>`

In the configuration session, answer as follows.

```
Modify memory configuration ? yes
Monitor type? foreground
Reset map (change monitor type requires map reset)? yes
Monitor location for real address? 1000h
Monitor location for virtual address? 40000000h
```

Mapping Memory for Your Program

Map memory for your program in the mapping memory configuration session. The monitor location is already mapped as emulation RAM ("MONITOR" is displayed in the "type" field).

Loading Foreground Monitor

Load the foreground monitor program.

load fg_mon `Nfmon70632.X <RETURN>`

The linked monitor program (Nfmon70632.X) is separated from user program. In this example, the Intel hexadecimal format and transparent configuration are assumed.

Loading User Program

Load the target memory portion of your program. To load the program into target memory, the emulator must be running in monitor.

break <RETURN>

load user_mem usr_prog.X <RETURN>

Next, load the emulation portion of your program. Since the portion includes the foreground monitor program, which is linked to refer to the symbols (in this example, STEP_ENTRY, BRK_ENTRY and NMI_ENTRY), the monitor program should not be running. Therefore, reset the emulator.

reset <RETURN>

load emul_mem usr_prog.X <RETURN>

Notes

B

Using the Format Converter

The 70632 Softkey Interface provides with the NEC COFF converter.

How to Use the Converter

The format converter is a program that generates HP format files from COFF format files for the 70632 (or the 70616). This means you can use available language tools to create the COFF format file, then load the file into the emulator using the format converter.

To execute the converter program, use the following command:

```
$v70cnvhp [options] <file_name>
```

<file_name> is the name of COFF format file (for the 70632 or the 70616) which is created by the 70632 linker (*ld70616*) or retrieved from the archive file created by the 70632 configurator (*cf70616*). The converter program will read the COFF format file. It will generate the following HP format files:

- HP Absolute file (with .X suffix)
- HP Linker symbol file (with .L suffix)
- HP Assembler symbol files (with .A suffix)

The converter accepts the following options.

Load address location options

You can select address mode (real or virtual) for the load address location of the HP absolute file, and for address symbols in the HP symbol files.

-v generates load address location and symbols in **virtual** address.

-r generates load address location and symbols in **real** address.

The HP 64758 emulator can load a program in real address or virtual address. It is determined by configuration question "Object file address attribute ?" in the "Pod configuration".

If neither option is specified, **-v** option is assumed.

In case of real mode application, this option is senseless because the address is the same between real address and virtual address.

File output control

The converter generates HP format files; Absolute file (with .X suffix), Linker symbol file (with .L suffix) and Assembler symbol files (with .A suffix).

You can specify which HP format file(s) should be generated by the converter.

-x generates absolute file (with .X suffix).

-l generates linker symbol file (with .L suffix).

-a generates assembler symbol files (with .A suffix).

If no option is specified, the converter generates all HP format files.

Note

For generating local symbols:

- specify "**-g**" option when you invoke the 70632 C Compiler (*cc70616*) from NEC.
- specify file name by using "**.file**" directive in the assembly source file.

Note

If you want to refer to global symbls in the assembly source file, you must specify file name by using "**.file**" directive. Othewise, global symbol can not be displayed by "display memory" commands.

Address Translation Table File

When the converter reads an address translation table file (aptable) generated by the configurator (*cf70616*), the converter generates the following files.

- Absolute file for address translation tables (aptable.X)
- Command files for specifying virtual space (files with .regs suffix)

Absolute file for address translation tables

The configurator can generate the file for the address translation tables (aptable). The converter converts this file to HP format absolute file (aptable.X). You can load the file *aptable.X* into emulator.

Command files for specifying virtual space

The converter generates command files to specify a virtual space. The command files contain emulator commands for modifying the XMMU class registers to specify a virtual space. The command files are generated for each virtual space which you specify to the configurator. The file name of each command file is its virtual space name for the base name and ".regs" for the suffix.

For example, to specify the virtual space for *process* task described in the "Virtual Mode Emulation Topics" chapter, enter the following commands.

```
load symbols process <RETURN>
PROCESS.regs <RETURN>
```

The "PROCESS", which is the base name of the command file, is the virtual space name specified in the configurator command file *skdemo2.cfc*.

Notes

Index

Notes