HP 64756/7

70136/70236 Emulator Softkey Interface

User's Guide

HEWLETT PACKARD

HP Part No. 64756-97013
Printed in U.S.A.
August 1994

Edition 5

Notice

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

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

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

Printing History

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

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

Edition 1	64756-97002, April 1990
Edition 2	64756-97005, August 1990
Edition 3	64756-97007, February 1991
Edition 4	64756-97010, August 1993
Edition 5	**64756-97013, July 1994**

Using this Manual

This manual covers the following emulators as used with the Softkey Interface.

- HP 64756F 70136 emulator
- HP 64757F 70236 emulator
- HP 64757G 70236A emulator

For the most part, the 70136, 70236 and 70236A emulators all operate the same way. Differences between the emulators are described where they exist. All of the 70136, 70236 and 70236A emulators will be referred to as the "70136 emulator" in this manual where they are alike. In the specific instances where 70236 or 70236A emulator differs from the 70136 emulator, it will be referred as the "70236 emulator" or "70236A emulator".

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; the Softkey Interface is described in the *Softkey Interface Reference* manual.

Organization

Chapter 1 Introduction to the 70136 Emulator. This chapter briefly introduces you to the concept of emulation and lists the basic features of the 70136 emulator.

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

Chapter 3 "In-Circuit" Emulation. This chapter shows you how to install the emulator probe into a target system and how to use "in-circuit" emulation features.

Chapter 4 Configuring the Emulator. This chapter shows you how to: restrict the emulator to real-time execution, select a target system clock source, allow the target system to insert wait states, and select foreground or background monitor.

Chapter 5 Using the Emulator. This chapter describes emulation topics which are not covered in the "Getting Started" chapter.

Appendix A Using the Foreground Monitor. This appendix describes the advantages and disadvantages of foreground and background monitors and how to use foreground monitor.

Appendix B Using the Extended Mode. This appendix shows you how to use the emulator in extended mode. This appendix describes a sample program and how to: load programs into the emulator, display memory, set software breakpoints, and use the emulation analyzer in extended mode.

Contents

3 In-Circuit Emulation

4 Configuring the Emulator

A Using the Foreground Monitor

B Using the Extended Mode

Illustrations

1

Introduction to the 70136 Emulator

Introduction

The topics in this chapter include:

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

Purpose of the Emulator

The 70136 emulator is designed to replace the 70136 microprocessor in your target system to help you debug/integrate target system software and hardware. The emulator performs just like the processor which it replaces, but at the same time, it gives you information about the bus cycle operation of the processor. The emulator gives you control over target system execution and allows you to view or modify the contents of processor registers, target system memory, and I/O resources.

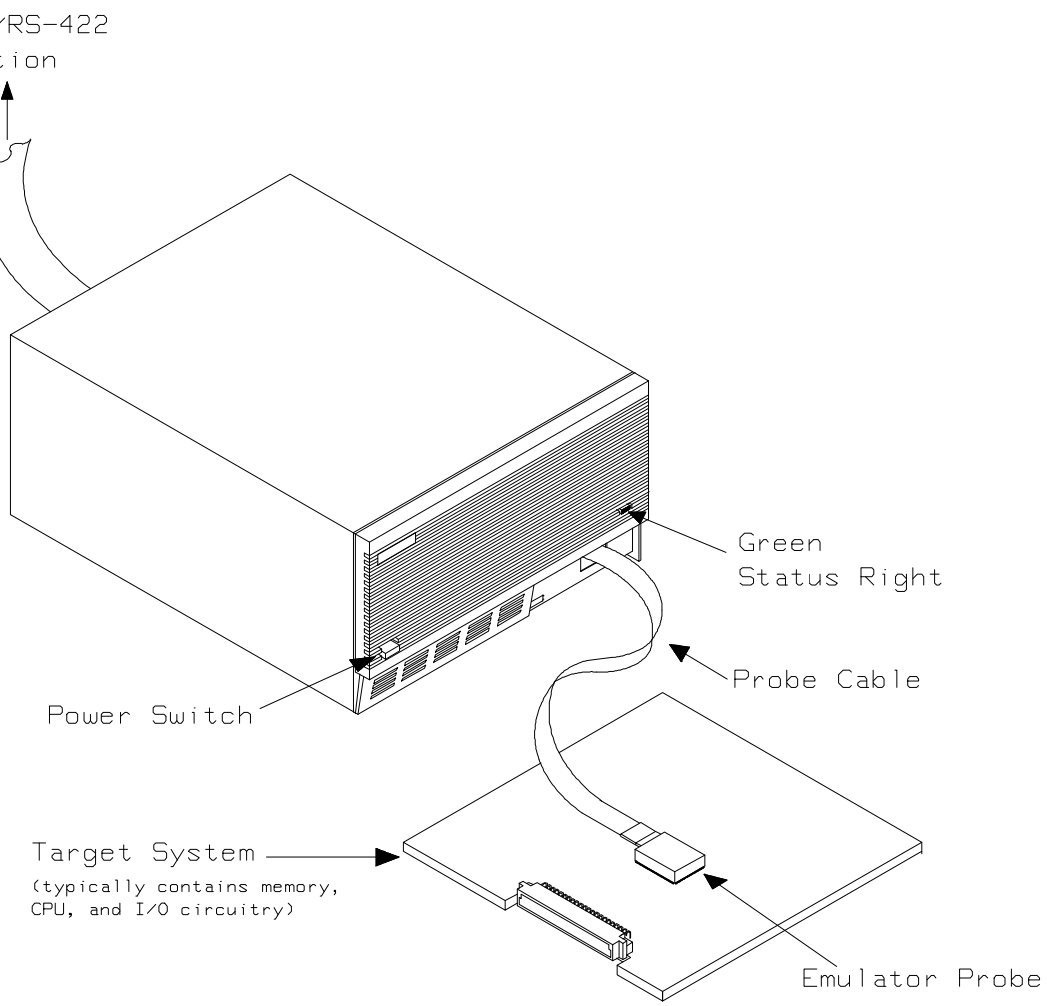


Figure 1-1. HP 64756/7 Emulator for uPD70136/70236

Features of the 70136 Emulator

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

Supported Microprocessors

The 70136 emulator probe has a 68-pin PLCC connector. Also provided is the adapter, HP PART No. 64756-61612, that will allow the PLCC probe to connect to the NEC EV-9200G-74 socket which replaces the 74-pin QFP package of 70136 microprocessor.

The HP 64756 emulator supports the following packages of 70136 microprocessor.

- 68-pin PLCC
- 68-pin PGA
  (With using PLCC to PGA adapter; refer to the "In-Circuit Emulation Topics" chapter in this manual)
- 74-pin QFP
  (With using PLCC to QFP adapter (HP PART No. 64756-61612) and NEC EV-9200G-74 socket)

The 70236 and 70236A emulator probe has an 132-pin PGA connector. Also provided is the NEC EV-9500GD-120 adapter that will allow the PGA probe to connect to the NEC EV-9200GD-120 socket which replaces the 120-pin QFP package of 70236 microprocessor.

The HP 64757 emulator supports the following packages of 70236 or 70236A microprocessor.

- 132-pin PGA
- 120-pin QFP
  (With using NEC EV-9500GD-120 adapter and NEC EV-9200GD-120 socket)

Clock Speeds

The 70136 emulator runs with an internal clock speed of 16 MHz (system clock), or with target system clocks from 2-16 MHz.

The 70236 emulator runs with an internal clock speed of 16 MHz (system clock), or with target system clocks from 4-32 MHz.

The 70236A emulator runs with an internal clock speed of 16 MHz (system clock), or with target system clocks from 4-40 MHz.

Emulation memory

The HP 70136 emulator is used with one of the following Emulation Memory Cards.

- HP 64726 128K byte Emulation Memory Card
- HP 64727 512K byte Emulation Memory Card
- HP 64728 1M byte Emulation Memory Card
- HP 64729 2M byte Emulation Memory Card

You can define up to 16 memory ranges (at 256 byte boundaries and at least 256 byte in length). The monitor occupies 4K bytes leaving 124K, 508K, 1020K or 2044K 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 guarded memory. The emulator generates an error message when accesses are made to guarded memory locations. You can also configure the emulator so that writes to memory defined as ROM cause emulator execution to break out of target program execution.

Analysis

The HP 70136 emulator is used with one of the following analyzers which allows you to trace code execution and processor activity.

- HP 64704 80-channel Emulation Bus Analyzer
- HP 64703 64-channel Emulation Bus Analyzer and 16-channel State Timing Analyzer

The HP 70236/70236A emulator is used with one of the following analyzers which allows you to trace code execution and processor activity.

- HP 64704 80-channel Emulation Bus Analyzer
- HP 64703 64-channel Emulation Bus Analyzer and 16-channel State Timing Analyzer
- HP 64794A/C/D Deep Emulation Bus Analyzer

When you use the HP 70236A emulator over 16MHz, you have to use the HP 64794 Deep Emulation Bus Analyzer.

The Emulation Bus Analyzer monitors the emulation processor using an internal analysis bus. The HP 64703 64-channel Emulation Bus Analyzer and 16-channel State/Timing Analyzer allows you to probe up to 16 different lines in your target system.

Registers

You can display or modify the 70136 internal register contents.

Single-Step

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

Breakpoints

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

You can also define software breakpoints in your program. The emulator uses one of 70136 undefined opcode (F1 hex) as software breakpoint interrupt instruction. When you define a software breakpoint, the emulator places the breakpoint interrupt instruction (F1 hex) at the specified address; after the breakpoint interrupt instruction causes emulator execution to break out of your program, the emulator replaces the original opcode.

Reset Support

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

Configurable Target System Interface

You can configure the emulator so that it honors target system wait requests when accessing emulation memory. You can configure the emulator so that it presents cycles to, or hides cycles from, the target system when executing in background.

Foreground or Background Emulation Monitor

The emulation monitor is a program that is executed by the emulation processor. It allows the emulation controller to access target system resources. For example, when you display target system memory, it is the monitor program that executes 70136 instructions which read the target memory locations and send their contents to the emulation controller.

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

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

Real-Time Operation

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

You can restrict the emulator to real-time execution. When the emulator is executing your program under the real-time restriction, commands which display/modify registers, display/modify target system memory or I/O, or single-step are not allowed.

Easy Products Upgrades

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

Limitations, Restrictions

DMA Support

In the 70136 Emulator, Direct memory access to the emulation memory by DMA controller is not permitted.

In the 70236 and the 70236A Emulator, Direct memory access to the emulator by external DMA controller is not permitted.

User Interrupts

If you use the background monitor in the 70136 emulator, interrupts are suspended or ignored during background operation. NMI is suspended until the emulator goes into foreground operation. INT interrupt is ignored.

If you use the background monitor in the 70236 and the 70236A emulator, interrupts from target system are suspended during background operation. NMI, and INTP0-INTP7 are suspended until the emulator goes into foreground operation.

Interrupts While Executing Step Command

While executing user program code in stepping in the foreground monitor, interrupts are accepted if they are enabled in the foreground monitor program. When using the foreground monitor you will see the following error message, if the interrupts are acknowledged before stepping user program code.

```
ERROR: Stepping failed
```

Although the error message above appears, the code is executed as you expected to do.

Accessing Internal I/O Registers

When you access internal I/O registers of the emulator, you should use the "display/modify register" command with their register name instead of the "display/modify io_port" command.

PC relative addressing in trace list

When you use the following setting in your program, the branch address forming in PC relative addressing may change to a wrong value only in disassemble list.

- The program is running in the extended address mode.
- The effective address for the PC relative addressing is in the other page.
- The order of the pages is not in sequence in extended address.

"BRKXA" and "RETXA" Instructions in Stepping

When the "BRKXA" and "RETXA" instructions are executed in stepping, the emulator reads memory for disassembly after stepping.
When you execute "BRKXA" instruction in stepping, the normal address where the "BRKXA" instruction is located is extended to read memory for disassemble after stepping.
When you execute "RETXA" instruction in stepping, the normal address which is extended to point the "RETXA" instruction is not extended to read memory for disassemble after stepping.

Stepping at Software Breakpoint

When you execute step commands in the foreground monitor, you should not step at the address which the "Software Breakpoint" was set; the stepping will be failed.

```
ERROR: Stepping failed
```

Evaluation Chip

Hewlett-Packard makes no warranty of the problem caused by the 70136/70236/70236A Evaluation chip in the emulator.

2

Getting Started

Introduction

This chapter will lead you through a basic, step by step tutorial that shows how to use the 70136 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 examples.

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.

Note that this chapter will show you how to use the emulator mainly about the normal mode. Refer to appendix B for using the extended mode of the emulator.

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/Service* 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* manual also 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 70136 emulator.

A Look at the Sample Program

The sample program used in this chapter is shown in Figure 2-1. The program continuously reads values from **Cmd_Input**; when a value other than NULL is found, the program calls the **Write_Msg** function to copy a string to the **Msg_Dest** array.

The sample program and the associated output files, including the HP format absolute files, have been shipped with the Softkey Interface; copy these files to the current directory with the following command:

```
$ cp /usr/hp64000/demo/emul/hp64756/* .
(70136)
$ cp /usr/hp64000/demo/emul/hp64757/* .
(70236)
```

The file *cmd_rds.X* contains the absolute code of the program. The file *cmd_rds.L* contains the list of global symbols. The files *cmd_rds.A* contains the 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.

```
 1  volatile char Cmd_Input;
 2  char Msg_Dest[0x20];
 3
 4  void Write_Msg (const char *s)
 5  {
 6          char *Dest_Ptr;
 7
 8          Dest_Ptr = Msg_Dest;
 9          while (*s != '\0')
10          {
11                  *Dest_Ptr = *s;
12                  Dest_Ptr++;
13                  s++;
14          }
15  }
16
17  main ()
18  {
19          static char Msg_A[] = "Command A Entered              ";
20          static char Msg_B[] = "Entered B Command              ";
21          static char Msg_I[] = "Invalid Command                ";
22          char c;
23
24          for (;;)
25          {
26                  Cmd_Input = '\0';
27                  while ((c = Cmd_Input) == '\0');
28                  switch (c) {
29                          case 'A' :
30                                  Write_Msg (Msg_A);
31                                  break;
32                          case 'B' :
33                                  Write_Msg (Msg_B);
34                                  break;
35                          default :
36                                  Write_Msg (Msg_I);
37                                  break;
38                  }
39          }
40  }
```

Figure 2-1. The "cmd_rds.c" Sample Program

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 have used previous HP 64000-UX emulators (for example, HP 64200 Series), you may be more familiar with the **pmon**, **msinit**, and **msconfig** 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 70136 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 70136 emulator, enter:

```
make_sys emv33 <RETURN>
```

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

```
add <module_number> naming_it n70136 <RETURN>
```

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

```
end <RETURN>
```

If the measurement system and emulation module are named "emv33" and "n70136" as shown above, you can enter the emulation system with the following command:

```
emv33 default n70136 <RETURN>
```

If this command is successful, you will see a display similar to figure 2-2. 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.

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).
For example, the emulator name in the device table entry shown below are "v33" for 70136 and "v53" for 70236/70236A.

```
#-------------+---------+--------------------+----+------+------+----+----+----
#             |         |                    |xpar| baud |parity|flow|stop|char
# logical name|processor|     physical       |mode| rate |      |    |bits|size
#  (14 chars) |  type   |      device        |    |      |      |XON |    |
#             |         |                    |OFF |      | NONE |RTS | 2  | 8
#-------------+---------+--------------------+----+------+------+----+----+----
#
v33            n70136      /dev/emcom23      OFF  230400  NONE  RTS   2    8
v53            n70236      /dev/emcom23      OFF  230400  NONE  RTS   2    8
```

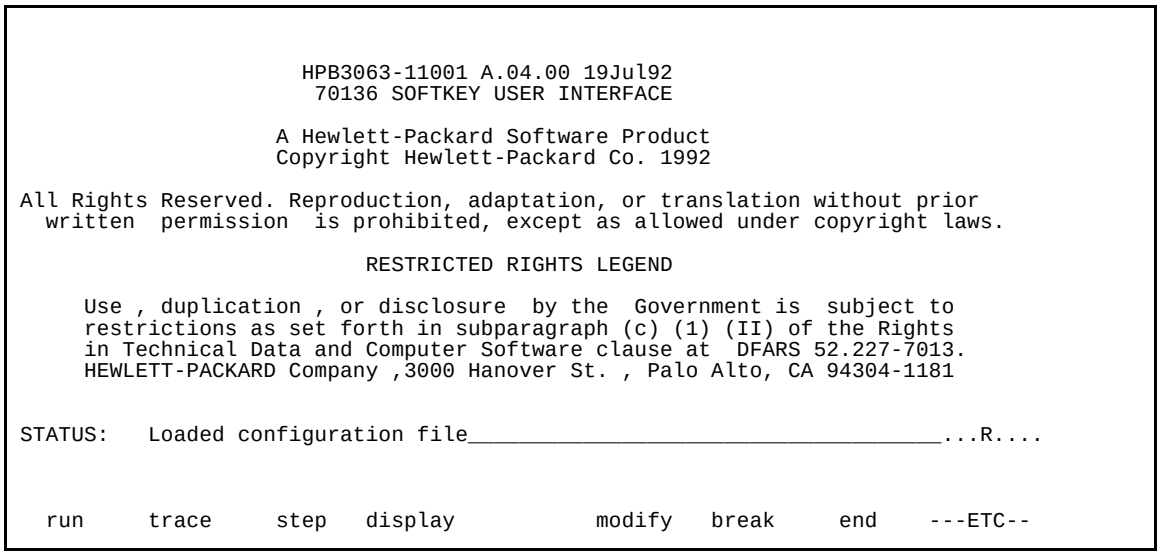


Figure 2-2. Softkey Interface Display

If this command is successful, you will see a display similar to figure 2-2. 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.

Configure the Emulator for Examples

To do operations described in this chapter (loading absolute program into emulation memory, displaying memory contents, etc), you need to configure the emulator as below. For detailed description of each configuration options (question), refer to the "*Configuring the Emulator*" chapter.

To get into the configure session of the emulator, enter the following command.

```
modify  configuration <RETURN>
```

The answer to series of questions as below.

```
Micro-processor clock source?  internal <RETURN>
Enter monitor after configuration?  yes <RETURN>
Restrict to real-time runs?  no <RETURN>
Modify memory configuration?  yes <RETURN>
Monitor type?  background <RETURN>
Background monitor location?  0FF000H <RETURN>
```

Now you should be facing memory mapping screen. Three mapper terms must be specified for the sample program.

```
    0h thru 03ffh emulation  ram <RETURN>
    10000h thru 1f3ffh emulation  ram <RETURN>
    80000h thru 8f7ffh emulation  rom <RETURN>
    end <RETURN>
Modify emulator pod configuration?  no <RETURN>
Modify debug/trace options?  no <RETURN>
Modify simulated I/O configuration?  no <RETURN>
Modify external analyzer configuration?  no <RETURN>
Modify interactive measurement specification?  no <RETURN>
Configuration file name?  cmd_rds <RETURN>
```

If you wish to save the configuration specified above, answer this question as shown.

Now you are ready to go ahead. Above configuration is used through out this chapter.

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>

```
---SYSTEM COMMANDS & COMMAND FILES---

?                          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
                              see "COMMAND FILES EXAMPLES" below for more detail
log_commands to <FILE>     logs the next sequence of commands to file
log_commands off           discontinue logging commands
name_of_module             get the "logical" name of module (see 64700tab.net)

set <ENVVAR> = <VALUE>     set and export a shell environment variable
set HP64KPATH = <MYPATH>   set and export the shell environment variable that
 --More--(23%)
```

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.

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 cf' <RETURN>
```

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).

Note

If you want to use the Terminal Interface command by entering from keyboard directly, you can do it after entering the following command.

```
pod_command keyboard
```

```
Pod Commands
  Time                Command
  --- VALID CONFIGURATION          NAMES ---
    clk     - select internal or external emulation clock
    rrt     - enable/disable restrict to real time runs
    mon     - select foreground or background monitor
    cyc     - enable/disable driving background cycle
    loc     - background monitor location
    rad     - segment:offset translation method
    trst    - enable/disable RESET signal from the target system
    nmi     - enable/disable NMI signal from the target system
    rdy     - relationship between emulator and target ready (lk or unlk)
    ebs     - select bus sizing signal for the emulation memory
    tbs     - select bus sizing signal for the target memory
    pgrd    - enable/disable read PGR on address translation
    lad     - select address mode for file loading
    aex     - select AEX signal when background
    fpu     - select FPU type for disassembly

STATUS:   N70136--Running in monitor_____________________________________...R....
pod_command 'help cf'


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

Loading Absolute Files

The "load" command allows you to load absolute files into emulation or target system memory. You can load absolute files in the following formats:

- HP absolute.
- Intel Object Module Format (OMF-86).
- HP-OMF V33 absolute. (This is the file format generated by the HP 64875 V33/53 Extended Mode Locator product.)

The "load" command has no special options for loading different absolute file formats; instead, the contents of the file are examined to determine the format being used.

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".

To load the emulator sample program absolute file, enter the following command:

```
load cmd_rds <RETURN>
```

Displaying Symbols

If symbol information is present in the absolute file, it is loaded along with the absolute file (unless you use the "nosymbols" syntax). Both global symbols and symbols that are local to a program module can be displayed.

Global

To display global symbols, enter the following command.

```
display global_symbols <RETURN>
```

Listed are: address ranges associated with a symbol, the segment that the symbol is associated with, and the offset of that symbol within the segment.

```
Global symbols in cmd_rds
Procedure symbols
Procedure name ________________ Address range __ Segment _____________ Offset
Write_Msg                    8000:0000 - 003C     PROG                     0000
_div_by_0_trap               800C:0054 - 006E     PROG                     0000
_exec_funcs                  8056:0041 - 0061     PROG                     0000
_exit_msg                    800C:015E - 018B     PROG                     0000
_fp_trap                     800C:02AE - 0415     PROG                     0000
_initGlobals                 800C:00FE - 015C     PROG                     0000
atexit                       8056:0008 - 0040     PROG                     0000
main                         8000:003D - 00C1     PROG                     003D

Static symbols
Symbol name ___________________ Address range __ Segment _____________ Offset
Cmd_Input                    1009:0008            DATA                     0000
Err_Handler                  805C:0057            PROG                     0000
MM_CHECK_L                    1000000A            PROG                     0000
MM_CHECK_X                    1000000A            PROG                     0000

STATUS:   N70136--Running in monitor_____________________________________...R....
display  global_symbols


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

Local

When displaying local symbols, you must include the name of the module in which the symbols are defined. For example:

```
display local_symbols_in cmd_rds.c: <RETURN>
```

As you can see, the procedure symbols and static symbols in "cmd_rds.c" are displayed.
If there is more than a screenful of information, you can use the up arrow, down arrow, <Next> or <Prev> keys to scroll the information up or down on the display.

```
Symbols in /usr/hp64000/demo/emul/hp64756/cmd_rds.c:
Procedure symbols
Procedure name ________________ Address range __ Segment _____________ Offset
Write_Msg                    8000:0000 - 003C     PROG                      0000
main                         8000:003D - 00C1     PROG                      003D

Static symbols
Symbol name ___________________ Address range __ Segment _____________ Offset
Cmd_Input                    1009:0008            DATA                      0000
Msg_A                        1009:0029            DATA                      0021
Msg_B                        1009:004A            DATA                      0042
Msg_Dest                     1009:0009            DATA                      0001
Msg_I                        1009:006B            DATA                      0063
_Cmd_Input                   1009:0008            DATA                      0000
_Msg_Dest                    1009:0009            DATA                      0001
_Write_Msg                   8000:0000            PROG                      0000
_main                        8000:003D            PROG                      003D


STATUS:   N70136--Running in monitor_____________________________________...R....
display local_symbols_in cmd_rds.c:


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

Source Lines

To display the address ranges associated with the program's source file, you must display the local symbols in the file. For example:

```
display local_symbols_in cmd_rds.c: <RETURN>
```

And scroll the information down on the display with up the arrow, or <Next> key.

```
Symbols in /usr/hp64000/demo/emul/hp64756/cmd_rds.c:
Symbol name _____________________ Address range __ Segment _____________ Offset

Source reference symbols
Line range _____________________ Address range __ Segment _____________ Offset
#1-#5                          8000:0000 - 0009     PROG                     0000
#6-#8                          8000:000A - 0013     PROG                     000A
#9-#9                          8000:0014 - 0017     PROG                     0014
#10-#11                        8000:0018 - 0023     PROG                     0018
#12-#12                        8000:0024 - 0027     PROG                     0024
#13-#13                        8000:0028 - 002B     PROG                     0028
#14-#14                        8000:002C - 0038     PROG                     002C
#15-#15                        8000:0039 - 003C     PROG                     0039
#16-#18                        8000:003D - 0046     PROG                     003D
#19-#24                        8000:0047            PROG                     0047
#25-#26                        8000:0048 - 004C     PROG                     0048
#27-#27                        8000:004D - 005E     PROG                     004D
#28-#28                        8000:005F - 007B     PROG                     005F

STATUS:    N70136--Running in monitor_____________________________________...R....
display local_symbols_in cmd_rds.c:


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

Displaying Memory in Mnemonic Format

You can display, in mnemonic format, the absolute code in memory. For example to display the memory of the sample program,

```
display memory main mnemonic <RETURN>
```

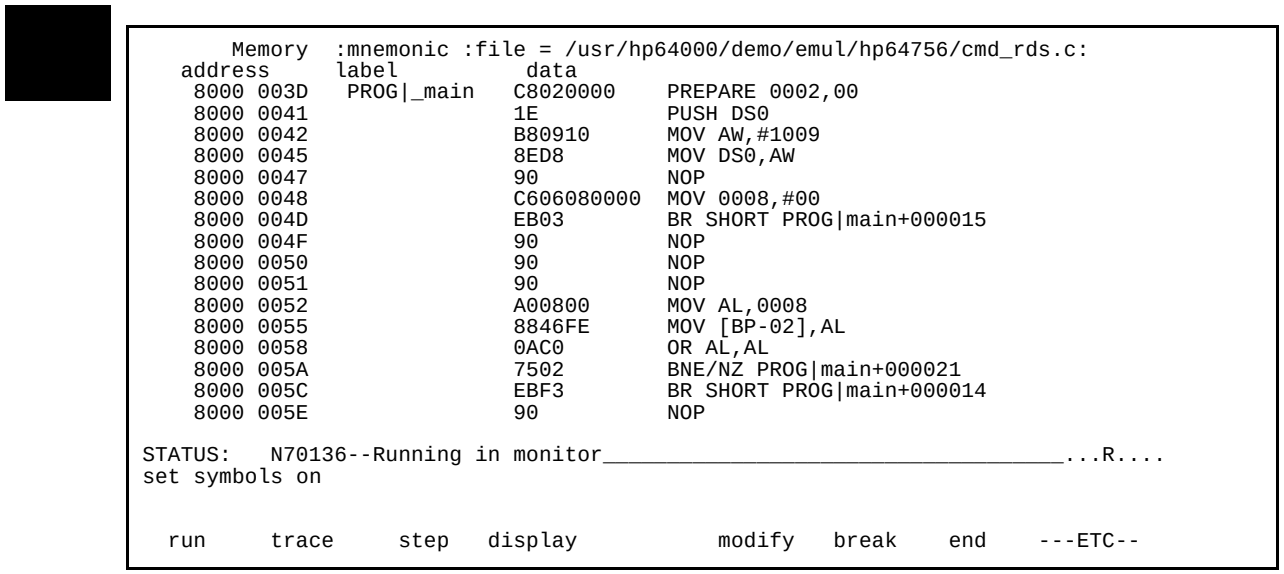
Notice that you can use symbols when specifying expressions. The global symbol **main** is used in the command above to specify the starting address of the memory to be displayed.

```
   Memory  :mnemonic :file = /usr/hp64000/demo/emul/hp64756/cmd_rds.c:
   address      data
    8000 003D  C8020000    PREPARE 0002,00
    8000 0041  1E          PUSH DS0
    8000 0042  B80910      MOV AW,#1009
    8000 0045  8ED8        MOV DS0,AW
    8000 0047  90          NOP
    8000 0048  C606080000  MOV 0008,#00
    8000 004D  EB03        BR SHORT 000052
    8000 004F  90          NOP
    8000 0050  90          NOP
    8000 0051  90          NOP
    8000 0052  A00800      MOV AL,0008
    8000 0055  8846FE      MOV [BP-02],AL
    8000 0058  0AC0        OR AL,AL
    8000 005A  7502        BNE/NZ 00005E
    8000 005C  EBF3        BR SHORT 000051
    8000 005E  90          NOP

STATUS:   N70136--Running in monitor_____________________________________...R....
display memory main mnemonic


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

Symbols in the Display

The "set" command allows you to include symbols in mnemonic memory displays and in the trace displays. For example:

```
set symbols on <RETURN>
```



```
       Memory  :mnemonic :file = /usr/hp64000/demo/emul/hp64756/cmd_rds.c:
   address     label          data
    8000 003D   PROG|_main   C8020000    PREPARE 0002,00
    8000 0041                1E          PUSH DS0
    8000 0042                B80910      MOV AW,#1009
    8000 0045                8ED8        MOV DS0,AW
    8000 0047                90          NOP
    8000 0048                C606080000  MOV 0008,#00
    8000 004D                EB03        BR SHORT PROG|main+000015
    8000 004F                90          NOP
    8000 0050                90          NOP
    8000 0051                90          NOP
    8000 0052                A00800      MOV AL,0008
    8000 0055                8846FE      MOV [BP-02],AL
    8000 0058                0AC0        OR AL,AL
    8000 005A                7502        BNE/NZ PROG|main+000021
    8000 005C                EBF3        BR SHORT PROG|main+000014
    8000 005E                90          NOP

STATUS:   N70136--Running in monitor_____________________________________...R....
set symbols on


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

Source Lines in the Display

The "set" command also allows you to include source lines in mnemonic memory displays and in the trace displays. For example:

```
set source on  <RETURN>
```

```
   Memory  :mnemonic :file = /usr/hp64000/demo/emul/hp64756/cmd_rds.c:
   address     label          data
      16
      17       main ()
      18       {
    8000 003D   PROG|_main   C8020000    PREPARE 0002,00
    8000 0041                1E          PUSH DS0
    8000 0042                B80910      MOV AW,#1009
    8000 0045                8ED8        MOV DS0,AW
      19            static char Msg_A[] = "Command A Entered              ";
      20            static char Msg_B[] = "Entered B Command              ";
      21            static char Msg_I[] = "Invalid Command                ";
      22            char c;
      23
      24            for (;;)
    8000 0047                90          NOP
      25            {
      26                 Cmd_Input = '\0';

STATUS:   N70136--Running in monitor_____________________________________...R....
set source on


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

Using Software Breakpoints

Software breakpoints are provided with one of 70136 undefined opcode (F1 hex) as breakpoint interrupt instruction.
When you define or enable a software breakpoint, the emulator will replace the opcode at the software breakpoint address with the breakpoint interrupt instruction.

Caution

When you use extended address mode, care should be taken for software breakpoints. If you change the relation between the physical address and the extended address after you set a software breakpoint (ex. change address mode or change the value of the page registers), emulation system may not recognize the software breakpoint.

Refer to the "*Using the Extended Mode*" appendix.

Caution

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

Note

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

Note

Because software breakpoints are implemented by replacing opcodes with the breakpoint interrupt instruction, you cannot define software breakpoints in target ROM.

When software breakpoints are enabled and emulator detects the breakpoint interrupt instruction (F1 hex), it generates a break to background request which as with the "processor break" command. Since the system controller knows the locations of defined software breakpoints, it can determine whether the breakpoint interrupt instruction (F1 hex) is a software breakpoint or opcode in your target program.

If it is a software breakpoint, execution breaks to the monitor,and the breakpoint interrupt instruction is replaced by the original opcode. A subsequent run or step command will execute from this address.
If it is an opcode of your target program, execution still breaks to the monitor, and an "Undefined software breakpoint" status message is displayed.
When software breakpoints are disabled, the emulator replaces the breakpoint interrupt instruction with the original opcode.

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>
```

Setting a Software Breakpoint

To set a software breakpoint at the address of global symbol "main" or (or source line 17), enter the following command.

```
modify software_breakpoints set main
<RETURN>
```

or:

```
modify software_breakpoints set line 17
<RETURN>
```

```
     Memory  :mnemonic :file = /usr/hp64000/demo/emul/hp64756/cmd_rds.c:
   address     label          data
      16
      17       main ()
      18       {
*   8000 003D   PROG|_main   F1          illegal opcode, data = F1
    8000 003E                0200        ADD AL,[BW][IX]
    8000 0040                001EB809    ADD 09B8,BL
    8000 0044                108ED890    ADDC [BP-6F28],CL
      25             {
      26                  Cmd_Input = '\0';
    8000 0048                C606080000  MOV 0008,#00
      27                  while ((c = Cmd_Input) == '\0');
    8000 004D                EB03        BR SHORT PROG|main+000015
    8000 004F                90          NOP
    8000 0050                90          NOP
    8000 0051                90          NOP
    8000 0052                A00800      MOV AL,0008

STATUS:   N70136--Running in monitor_____________________________________...R....
modify software_breakpoints set line 17


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

Notice that an asterisk (*) appears next to the breakpoint address. The asterisk shows that a software breakpoint is pending at that address.

Running the Program

The "run" command causes the emulator to execute the user program. 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.

From Transfer Address

The "run from transfer_address" command specifies that the emulator start executing at a previously defined "start address". Transfer addresses are defined in assembly language source files with the END assembler directive (i.e., pseudo instruction). Enter:

```
run from transfer_address <RETURN>
```

```
   Memory  :mnemonic :file = /usr/hp64000/demo/emul/hp64756/cmd_rds.c:
   address     label          data
      16
      17       main ()
      18       {
>   8000 003D   PROG|_main   C8020000    PREPARE 0002,00
    8000 0041                1E          PUSH DS0
    8000 0042                B80910      MOV AW,#1009
    8000 0045                8ED8        MOV DS0,AW
      19            static char Msg_A[] = "Command A Entered               ";
      20            static char Msg_B[] = "Entered B Command               ";
      21            static char Msg_I[] = "Invalid Command                 ";
      22            char c;
      23
      24            for (;;)
    8000 0047                90          NOP
      25            {
      26                 Cmd_Input = '\0';

STATUS:   N70136--Running in monitor      Software break: 08054:00049___...R....
run from transfer_address


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

Notice the highlighted bar on the screen; it shows the current program counter.

Notice also that the asterisk is no longer next to the breakpoint address; this shows that the breakpoint has been hit and is no longer active.

From Reset

The "run from reset" command specifies that the emulator begin executing from reset vector as actual microprocessor does.

(See "Running From Reset" section in the "In-Circuit Emulation" chapter).

Note

You cannot use over 100000 hex address in "run" command.

Stepping Through the Program

The step command allows you to step through program execution an instruction or a number of instructions at a time. You can step though the instructions associated with high-level program source lines. 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 source <RETURN>
```

```
   Memory  :mnemonic :file = /usr/hp64000/demo/emul/hp64756/cmd_rds.c:
   address     label          data
      17       main ()
      18       {
    8000 003D   PROG|_main   C8020000    PREPARE 0002,00
    8000 0041                1E          PUSH DS0
    8000 0042                B80910      MOV AW,#1009
    8000 0045                8ED8        MOV DS0,AW
      19            static char Msg_A[] = "Command A Entered              ";
      20            static char Msg_B[] = "Entered B Command              ";
      21            static char Msg_I[] = "Invalid Command                ";
      22            char c;
      23
      24            for (;;)
    8000 0047                90          NOP
      25            {
      26                 Cmd_Input = '\0';
>   8000 0048                C606080000  MOV 0008,#00

STATUS:   N70136--Stepping complete_____________________________________...R....
step source


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

Notice that the highlighted bar (the current program counter) moves to the instructions associated with the next source line.

Enter the "step source" command again by pressing:

```
<RETURN>,  <RETURN>
```

Notice that the emulator continues to step through the program and that the message "assembly steps taken: XXX" appears on the status line. This happens because the "while" test remains true, and the emulator never completes the execution of the assembly instructions associated with that source line. To stop the "step source" command, enter:

```
<CTRL>-c
```

Continue user program execution with the "run" command.

```
run <RETURN>
```

Modifying Memory

The sample program is a simple command interpreter. Commands are sent to the sample program through a "char" sized memory location, global variable **Cmd_Input**. You can use the modify memory feature to send a command to the sample program.
For example, to enter the command "A" (41H), use the following command:

```
modify memory Cmd_Input bytes to 41h <RETURN>
```

or:

```
modify memory Cmd_Input string to 'A'
<RETURN>
```

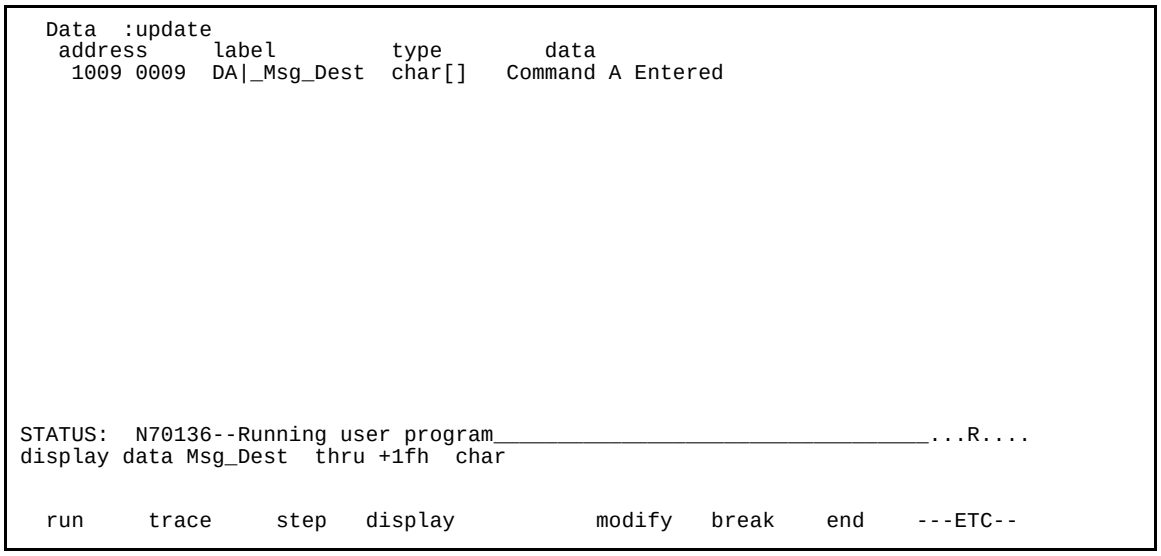
To verify that the program correctly copied the message "Command A Entered" to the **Msg_Dest** array, display the contents of the array with the following command:

```
display data Msg_Dest thru +1fh char
<RETURN>
```

Enter the following commands to verify that the program works for the other possible command inputs.

```
modify memory Cmd_Input string to 'B'
<RETURN>
modify memory Cmd_Input string to 'C'
<RETURN>
```

Notice that the display is updated when the memory contents change due (indirectly) to the "modify memory" command.



Breaking into the Monitor

The "break" command causes emulator execution to break 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>
```

Displaying Registers

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

```
display registers <RETURN>
```

Note

You should not change the value of the 70136 page registers with using "modify io_port" command. You should use the "modify registers" command to change the value of page registers.

Refer to "Register Names and Classes" section in chapter 5.

```
Registers

Next PS:PC 8000:0055H
    PC 0055    SP 7EF6    IX 1703    IY 0049    BP 7EFA
    PS 8000    SS 1111   DS0 1009   DS1 1009      [rrrrvdibszfafpic]
    AW 1000    BW 0000    CW 0000    DW 1009   PSW 1111001001000110









STATUS:  N70136--Running in monitor_____________________________________...R....
 display registers


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

Stepping Through the Program

You can step through sample program instructions while displaying registers. For example, entering several step commands will give you a display similar to the following.

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

Note

You cannot use over 100000 hex address in "step from" command.

Note

There are a few cases in which the emulator can not step.
Step command is not accepted between each of the following instructions and the next instruction.

1) Manipulation instructions for sreg:
   MOV sreg,reg16; MOV sreg,mem16; POP sreg.

2) Prefix instructions:
   PS:, SS:, DS0:, DS1:,
   REPC, REPNC, REP, REPE, REPZ, REPNE, REPNZ.

3) EI, RETI, DI, BUSLOCK.

```
Registers

Next PS:PC 8000:0055H
    PC 0055    SP 7EF6    IX 1703    IY 0049    BP 7EFA
    PS 8000    SS 1111   DS0 1009   DS1 1009      [rrrrvdibszfafpic]
    AW 1000    BW 0000    CW 0000    DW 1009   PSW 1111001001000110

Step_PC 8000:0055H  MOV [BP-02],AL
Next PS:PC 8000:0058H
    PC 0058    SP 7EF6    IX 1703    IY 0049    BP 7EFA
    PS 8000    SS 1111   DS0 1009   DS1 1009      [rrrrvdibszfafpic]
    AW 1000    BW 0000    CW 0000    DW 1009   PSW 1111001001000110

Step_PC 8000:0058H  OR AL,AL
Next PS:PC 8000:005AH
    PC 005A    SP 7EF6    IX 1703    IY 0049    BP 7EFA
    PS 8000    SS 1111   DS0 1009   DS1 1009      [rrrrvdibszfafpic]
    AW 1000    BW 0000    CW 0000    DW 1009   PSW 1111001001000110

STATUS:   N70136--Stepping complete_____________________________________...R....
step


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

Continue user program execution with the "run" command.

```
run <RETURN>
```

Using the Analyzer

HP 64700 emulators contain an emulation analyzer. The emulation analyzer monitors the internal emulation lines (address, data, and status). Optionally, you may have an additional 16 trace signals which monitor external input lines. 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 look at the execution of the sample program after the address of the first instruction in the **Write_Msg** function (cmd_rds.c : line 4). To trigger on this address, enter:

```
trace after line 4 <RETURN>
```

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

```
modify memory Cmd_Input string to 'A'
<RETURN>
```

The status line now shows "Emulation trace complete".

Displaying the Trace

To display the trace, enter:

```
display trace <RETURN>
```

```
Trace List                  Offset=0
Label:      Address       Data    Opcode or Status w/ Source Lines   time count
Base:       symbols       hex           mnemonic w/symbols           relative
after  PROG|_Write_Msg      04C8    04C8  prefetch                   -----------
+001   Write_Msg+000002     0000    0000  prefetch                     240     nS
+002   ct5CAAa2:+007EE4     8000    8000  memory write                 200     nS
+003   ct5CAAa2:+007EE2     008A    008A  memory write                 200     nS
      ##########.../demo/emul/hp64756/cmd_rds.c - line     1 thru     5 ########
      volatile char Cmd_Input;
      char Msg_Dest[0x20];

      void Write_Msg (const char *s)
      {
+004   PROG|_Write_Msg      008A  PREPARE 0004,00                      40.    nS
+005   Write_Msg+000004     B81E    B81E  prefetch                     80.    nS
+006   ct5CAAa2:+007EE0     7EFA    7EFA  memory write                160     nS
+007   Write_Msg+000006     1009    1009  prefetch                    120     nS
+008   Write_Msg+000008     D88E    D88E  prefetch                    160     nS

STATUS:   N70136--Running user program    Emulation trace complete_____...R....
display trace


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

Line 0 (labeled "after") in the trace list above shows the state which triggered the analyzer. The trigger state is always on line 0.

To display the remaining lines of the trace, press the <PGDN> or <NEXT> key.

Displaying Trace with No Symbol

The trace listing shown above has symbol information because of the "**set symbols on**" setting before in this chapter. To see the trace listing with no symbol information, enter the following command.

```
set symbols off
```

```
Trace List                  Offset=0
Label:  Address   Data        Opcode or Status w/ Source Lines       time count
Base:     hex     hex                    mnemonic                    relative
after    080000     04C8    04C8  prefetch                   N      ------------
+001     080002     0000    0000  prefetch                   N        240     nS
+002     019000     8000    8000  memory write               N        200     nS
+003     018FFE     008A    008A  memory write               N        200     nS
      ##########.../demo/emul/hp64756/cmd_rds.c - line     1 thru     5 ########
      volatile char Cmd_Input;
      char Msg_Dest[0x20];

      void Write_Msg (const char *s)
      {
+004     080000     008A  PREPARE 0004,00                            40.     nS
+005     080004     B81E    B81E  prefetch                   N        80.     nS
+006     018FFC     7EFA    7EFA  memory write               N       160      nS
+007     080006     1009    1009  prefetch                   N       120      nS
+008     080008     D88E    D88E  prefetch                   N       160      nS

STATUS:  N70136--Running user program    Emulation trace complete_____...R....
 set  symbols  off


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

As you can see, the analysis trace display shows the trace list without symbol information.

Note

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

Character	Information
N	Normal address mode
E	Extended address mode
M	Monitor cycle (background)

Note

When you use the following setting in your program, the branch address forming in PC relative addressing may change to a wrong value in disassemble list.

- The program is running in the extended address mode.
- The effective address for the PC relative addressing is in the other page.
- The order of the pages is not in sequence in extended address.

Displaying Trace with Time Count Absolute

Enter the following command to display count information relative to the trigger state.

```
display trace count absolute <RETURN>
```

```
Trace List                  Offset=0
Label:  Address   Data        Opcode or Status w/ Source Lines       time count
Base:     hex     hex                 mnemonic                       absolute
after    080000     04C8    04C8  prefetch                  N      ------------
+001     080002     0000    0000  prefetch                  N      + 240     nS
+002     019000     8000    8000  memory write              N      + 440     nS
+003     018FFE     008A    008A  memory write              N      + 640     nS
      ##########.../demo/emul/hp64756/cmd_rds.c - line     1 thru     5 ########
      volatile char Cmd_Input;
      char Msg_Dest[0x20];

      void Write_Msg (const char *s)
      {
+004     080000     008A  PREPARE 0004,00                         + 680     nS
+005     080004     B81E    B81E  prefetch                  N      + 760     nS
+006     018FFC     7EFA    7EFA  memory write              N      + 920     nS
+007     080006     1009    1009  prefetch                  N      +   1.0   uS
+008     080008     D88E    D88E  prefetch                  N      +   1.2   uS

STATUS:  N70136--Running user program    Emulation trace complete______...R....
 display   trace count absolute


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

Displaying Trace with Compress Mode

If you want to see more executed instructions on a display, the 70136 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>
```

As you can see, the analysis trace display shows the analysis trace lists without prefetch cycles. With this command you can examine program execution easily.

If you want to see all of cycles including fetch cycles, enter following command:

```
display trace compress off <RETURN>
```

The trace display shows you all of the cycles the emulation analyzer have captured.

```
Trace List                  Offset=0
Label:  Address   Data        Opcode or Status w/ Source Lines       time count
Base:     hex     hex                    mnemonic                    absolute
+002     019000     8000    8000  memory write               N      + 440     nS
+003     018FFE     008A    008A  memory write               N      + 640     nS
      ##########.../demo/emul/hp64756/cmd_rds.c - line     1 thru     5 ########
      volatile char Cmd_Input;
      char Msg_Dest[0x20];

      void Write_Msg (const char *s)
      {
+004     080000     008A  PREPARE 0004,00                          + 680     nS
+006     018FFC     7EFA    7EFA  memory write               N      + 920     nS
+009     080004     D88E  PUSH    DS0                              +   1.2   uS
+011     018FF6     1009    1009  memory write               N      +   1.5   uS
+012     080005     1009  MOV     AW,#1009                         +   1.6   uS
+014     080008     09FC  MOV     DS0,AW                           +   1.7   uS
      ##########.../demo/emul/hp64756/cmd_rds.c - line     6 thru     8 ########

STATUS:   N70136--Running user program     Emulation trace complete_____...R....
 display  trace compress on


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

Note

When the analysis trace is displayed with compress mode, the time count may not indicate correct time counts. This happens when time count is **relative**. Since the compress mode feature is implemented by eliminating prefetch cycles when displaying analysis trace, relative time count shows incorrect value. If you are interested in the time count, display with time count **absolute**. Absolute value of time count always show correct value. Keep this note in your mind when display the trace with compress mode.

Reducing the Trace Depth

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

```
display trace depth 512 <RETURN>
```

Emulator Analysis Status Qualifiers

The following analysis status qualifiers may also be used with the 70136 emulator.

Qualifier	Status Bits	Description
bs16	0xx xx1x xxxx xxxxB	bus size 16
bs8	0xx xx0x xxxx xxxxB	bus size 8
coproc	0x0 xxxx x101 x0xxB	co-processor access
exec	0x0 xxxx x0xx xxxxB	executed code
extaddr	0xx 1xxx xxxx xxxxB	extended address mode
fetch	0x0 xxxx 1100 x11xB	program fetch
grdacc	0xx xxxx 0xxx x1xxB	guarded access
haltack	0x0 xxxx x111 x00xB	halt acknowledge
holdack	0x1 xxxx xxxx xxxxB	hold acknowledge
intack	0x0 xxxx x100 x01xB	interrupt acknowledge
io	0x0 xxxx x110 x0xxB	I/O access
memory	0x0 xxxx 1110 x1xxB	memory access
memforcp	0x0 xxxx 1101 x1xxB	memory access for cp
nmladdr	0xx 0xxx xxxx xxxxB	normal address mode
read	0x0 xxxx x1xx xx1xB	read cycle
write	0x0 xxxx x1xx xx0xB	write cycle
wrrom	0xx xxx0 xxxx xx0xB	write to ROM

In using the 70236 emulator, the analysis status qualifiers are shown below.

Qualifier	Status Bits	Description
bs16	1xxx xx1x xxxx xxxxB	bus size 16
bs8	1xxx xx0x xxxx xxxxB	bus size 8
coproc	1xx0 xxxx x101 00xxB	co-processor access
dma	1xx0 xxxx x110 11xxB	DMA cycle
dmac	0xxx xxxx xxxx xxxxB	DMA cascade
eio	1xx0 xxxx x110 00xxB	external I/O access
exec	1xx0 xxxx x0xx xxxxB	executed code
extaddr	1xxx 1xxx xxxx xxxxB	extended address mode
fetch	1xx0 xxxx 1100 011xB	program fetch
grdacc	1xxx xxxx 0xxx x1xxB	guarded access
haltack	1xx0 xxxx x111 000xB	halt acknowledge
holdack	1xx1 xxxx xxxx xxxxB	hold acknowledge
intacki	1xx0 xxxx x100 101xB	interrupt acknowledge (from ICU)
intacks	1xx0 xxxx x100 001xB	interrupt acknowledge (from SLAVE)
iio	1xx0 xxxx x110 10xxB	internal I/O access
memory	1xx0 xxxx 1110 01xxB	memory access
memforcp	1xx0 xxxx 1101 01xxB	memory access for cp
nmladdr	1xxx 0xxx xxxx xxxxB	normal address mode
read	1xx0 xxxx x1xx xx1xB	read cycle
refresh	1xx0 xxxx x100 111xB	refresh cycle
write	1xx0 xxxx x1xx xx0xB	write cycle
wrrom	1xxx xxx0 xxxx xx0xB	write to ROM

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*.

Resetting the Emulator

To reset the emulator, enter the following command.

```
reset <RETURN>
```

Note

When the emulator is held in a reset state,the emulator is set to normal address mode.

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

In-Circuit Emulation

Introduction

The emulator is *in-circuit* when it is plugged into the target system. This chapter covers topics which relate to in-circuit emulation.

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 Target System Probe

The 70136 emulator probe has a 68-pin PLCC connector; the 70236 and 70236A emulator probe has a 132-pin PGA connector. The 70236 and 70236A emulator is shipped with a pin protector over the target system probe. This guard is designed to prevent impact damage to the pins and should be left in place while you are not using the emulator.

Caution

DAMAGE TO THE EMULATOR CIRCUITRY MAY RESULT IF THESE PRECAUTIONS ARE NOT OBSERVED. The following precautions should be taken while using the 70136 emulator.

Power Down Target System. Turn off power to the user target system and to the 70136 emulator before inserting the user plug to avoid circuit damage resulting from voltage transients or mis-insertion of the user plug.

Verify User Plug Orientation. Make certain that Pin 1 of the target system microprocessor socket and Pin 1 of the user plug are properly aligned before inserting the user plug in the socket. Failure to do so may result in damage to the emulator circuitry.

Protect Against Static Discharge. The 70136 emulator contains devices which are susceptible to damage by static discharge. Therefore, operators should take precautionary measures before handling the user plug to avoid emulator damage.

Protect Target System CMOS Components. If your target system includes any CMOS components, turn on the target system first, then turn on the 70136 emulator; when powering down, turn off the emulator first, then turn off power to the target system.

Auxiliary Output Lines

Two auxiliary output lines, "**TARGET BUFFER DISABLE**" and "**SYSTEM RESET**", are provided with the 70136 emulator. The "**TARGET BUFFER DISABLE**" output line is also provided with the 70236 and 70236A emulator.

Caution

DAMAGE TO THE EMULATOR PROBE WILL RESULT IF THE AUXILIARY OUTPUT LINES ARE INCORRECTLY INSTALLED. When installing the auxiliary output lines into the end of the emulator probe cable, make sure that the ground pins on the auxiliary output lines (labeled with white dots) are matched with the ground receptacles in the end of the emulator probe cable.

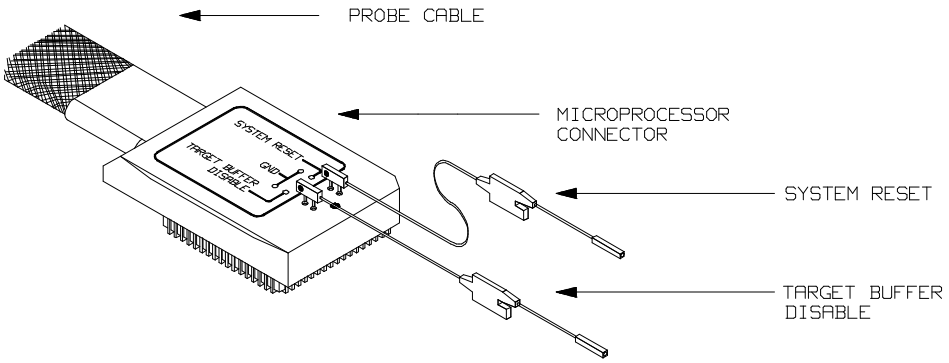
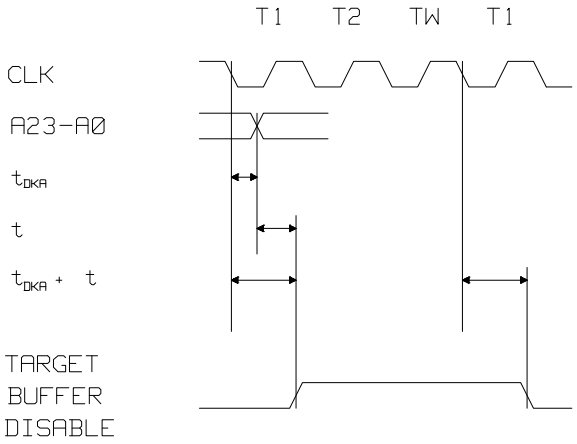


Figure 3-1. Auxiliary Output Lines (70136 Emulator)

TARGET BUFFER DISABLE ---This active-high output is used when the co-processor memory accesses to emulation memory will be operated. This output is used to tristate (in other words, select the high Z output) any target system devices on the 70136/70236/70236A data bus. Target system devices should be tristated because co-processor memory reads from emulation memory will cause data to be output on the user probe.

This "TARGET BUFFER DISABLE" output will be driven with the following timing in the co-processor memory access cycle.



The time 't' is

26 nsec MIN. (70136 Emulator)
56.8 nsec MAX.

26 nsec MIN. (70236 Emulator)
61.8 nsec MAX.

SYSTEM RESET (70136 only) ---This active-high, CMOS output should be used to synchronously reset the emulator and the target system.

Installing into a 70136 PLCC Type Socket

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

- Remove the 70136 microprocessor (PLCC type) from the target system socket. Note the location of pin 1 on the microprocessor and on the target system socket.
- Store the microprocessor in a protected environment (such as antistatic form).
- Install the microprocessor connector into the target system microprocessor socket.

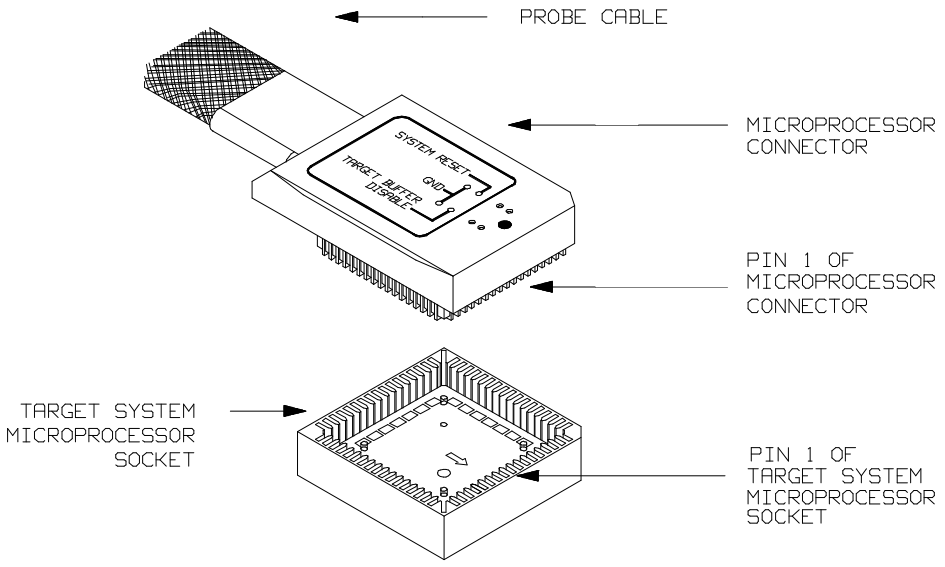


Figure 3-2. Installing into a 70136 PLCC type socket

Installing into a 70136 PGA Type Socket

The 70136 emulator is provided with an AMP 821574-1 socket and a pin protector in order to plug into the target system socket of an PGA type. You may use this AMP socket with the pin protector to connect the microprocessor connector to the target system.
To connect the microprocessor connector to the target system, proceeded with the following instructions.

- Remove the 70136 microprocessor (PGA type) from the target system socket. Note the location of pin A1 on the microprocessor and on the target system socket.
- Store the microprocessor in a protected environment (such as antistatic form).
- Place the microprocessor connector with an AMP socket and a pin protector (see figure 3-3), attached to the end of the probe cable, into the target system microprocessor socket.

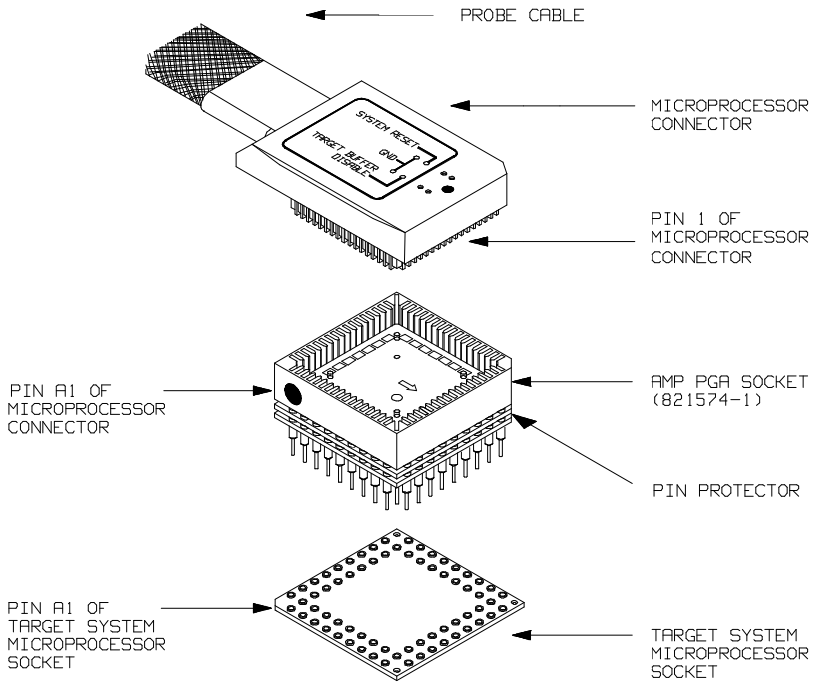


Figure 3-3. Installing into a 70136 PGA type socket

Installing into a 70136 QFP Type Socket

To connect the 70136 emulator microprocessor connector to the NEC EV-9200G-74 socket on the target system, you should use the adapter, HP PART NO. 64756-61612, that will allow the PLCC microprocessor connector to connect to the QFP socket.

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

- Note the location of pin 1 on the NEC EV-9200G-74 socket on the target system.
- Place the microprocessor connector with the adapter (see figure 3-4), attached to the end of the probe cable, into the target system microprocessor socket.

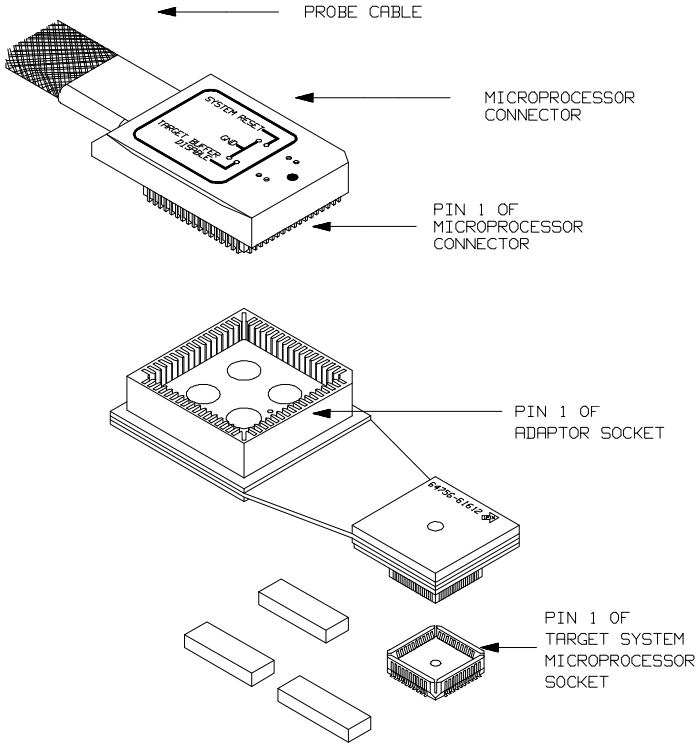


Figure 3-4. Installing into a 70136 QFP type socket

Installing into a 70236/70236A PGA Type Socket

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

- Remove the 70236 or 70236A microprocessor (PGA type) from the target system socket. Note the location of pin A1 on the microprocessor and on the target system socket.
- Store the microprocessor in a protected environment (such as antistatic form).
- Install the microprocessor connector into the target system microprocessor socket with a pin protector (see figure 3-5).

Caution

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

Installing into a 70236/70236A QFP Type Socket

To connect the 70236 or 70236A emulator microprocessor connector to the
NEC EV-9200GD-120 socket on the target system, you should use the NEC EV-9500GD-120 adapter that will allow the PGA microprocessor connector to connect to the QFP socket.

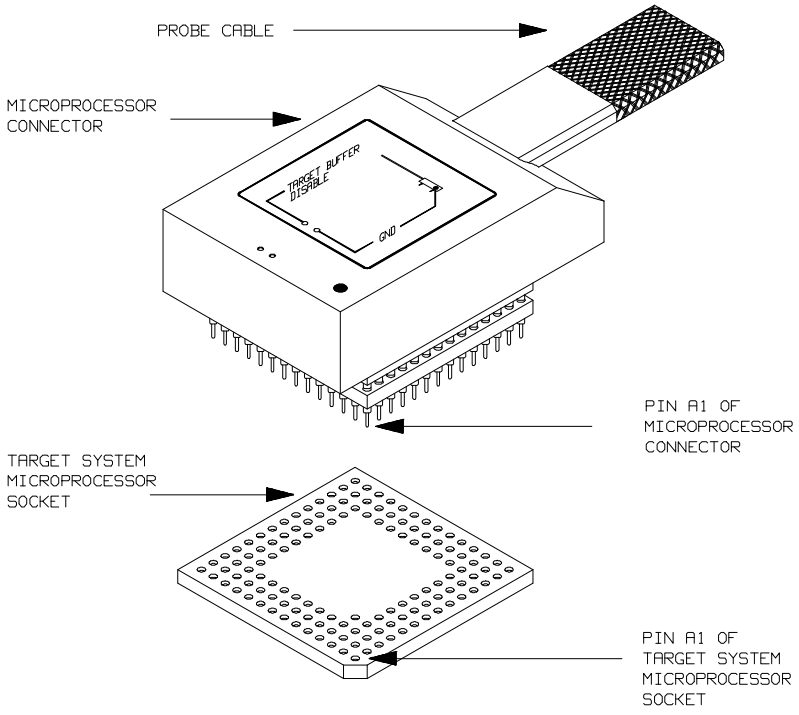


Figure 3-5 Installing into a 70236 PGA type socket

In-Circuit Configuration Options

The 70136 emulator provide configuration options for the following in-circuit emulation issues. Refer to the chapter on "Configuring the Emulator" for more information on these configuration options.

Using the Target System Clock Source

The default 70136, 70236 and 70236A emulator configuration selects the internal 16 MHz (system clock speed) clock as the emulator clock source.

You should configure the emulator to select an external target system clock source for the "in-circuit" emulation.

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 line while emulation memory is being accessed.

Note 

When you use the NEC uPD72291 coprocessor on your target system connected to 70136 microprocessor, the uPD72291 can access 70136 emulation memory on coprocessor memory read/write cycles.

In this case, you should reset the target system to connect the 70136 emulator to the uPD72291 coprocessor before starting emulation session.

Enabling NMI and RESET Input from the Target System

You can configure whether the emulator should accept or ignore the NMI and RESET signals from the target system.

Running the Emulator from Target Reset

You can specify that the emulator begins executing from target system reset. When the target system RESET line becomes active and then inactive, the emulator will start reset sequence (operation) as actual microprocessor.

At First, you must specify the emulator responds to RESET signal by the target system (see the "Enable RESET inputs from target system?" configuration in Chapter 4 of this manual).

To specify a run from target system reset, select:

```
run from reset <RESET>
```

The status now shows that the emulator is "Awaiting target reset". After the target system is reset, the status line message will change to show the appropriate emulator status.

Pin State in Background (70136)

While the emulator is running in the background monitor, probe pins are in the following state.

Address Bus	Same as foreground
Data Bus	Always high impedance except accessing target. When accessing target by background monitor, same as foreground.
R/$\overline{W}$,M/$\overline{IO}$ BUSST0	Setting the "Enable background cycles to target system? no", always high impedance except accessing target. When accessing target by background monitor, same as foreground. Setting the "Enable background cycles to target system? yes", always high level except accessing target. When accessing target by background monitor, same as foreground.
BUSST1	Setting the "Enable background cycles to target system? no", always high impedance except accessing target. When accessing target by background monitor, same as foreground. Setting the "Enable background cycles to target system? yes", always low level except accessing target. When accessing target by background monitor, same as foreground.
UBE	Setting the "Enable background cycles to target system? no", always high impedance except accessing target. When accessing target by background monitor, same as foreground. Setting the "Enable background cycles to target system? yes", Same as foreground.

Other	Same as foreground

Pin State in Background (70236/70236A)

While the emulator is running in the background monitor, probe pins are in the following state.

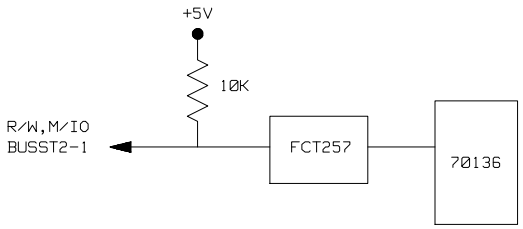
Address Bus	Same as foreground
Data Bus	Always high impedance except accessing target. When accessing target by background monitor, same as foreground.
R/$\overline{W}$,M/$\overline{IO}$, IORD,IOWR, MWR	Setting the "Enable background cycles to target system? no", always high impedance except accessing target. When accessing target by background monitor, same as foreground. Setting the "Enable background cycles to target system? yes", always high level except accessing target. When accessing target by background monitor, same as foreground.
MRD	Setting the "Enable background cycles to target system? no", always high impedance except accessing target. When accessing target by background monitor, same as foreground. Setting the "Enable background cycles to target system? yes" same as foreground except for emulation memory write. When accessing emulation memory write, low.
BUSST2-0, UBE,BCYST, DSTB,BUFEN	Setting the "Enable background cycles to target system? no", always high impedance except accessing target. When accessing target by background monitor, same as foreground. Setting the "Enable background cycles to target system? yes", Same as foreground.

Other	Same as foreground

Target System Interface (70136)

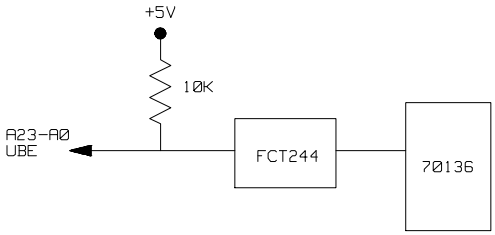
R/$\overline{W}$ M/$\overline{IO}$
BUSST2-1

These singals are connected to 70136 through FCT257 and 10K ohm pull-up register.

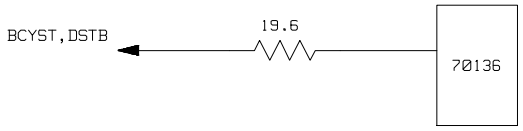


A23-A0
$\overline{UBE}$

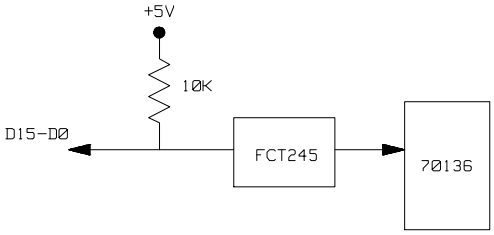
These singals are connected to 70136 through FCT244 and 10K ohm pull-up register.



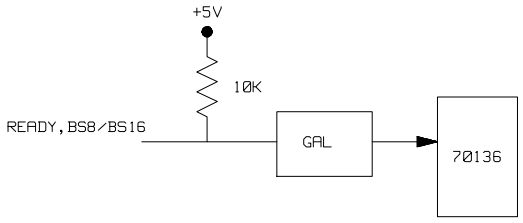
BCYST $\overline{\text{DSTB}}$ — These singals are connected to 70136 through 19.6 ohm.



D15-D0 — These singals are connected to 70136 through FCT245 and 10K ohm pull-up register.

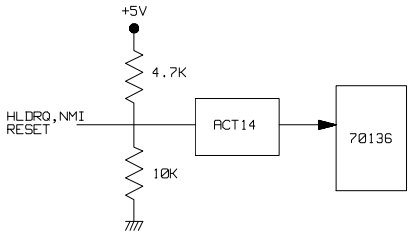


$\overline{\text{READY}}$
BS8/BS16 — These singals are connected to 70136 through GAL and 10K ohm pull-up register.



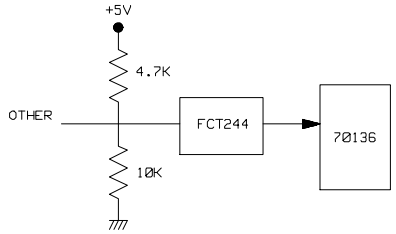
HLDRQ
NMI
RESET

These singals are connected to 70136 through ACT14 and 4.7K ohm pull-up and 10K ohm pull-down registers.



OTHER

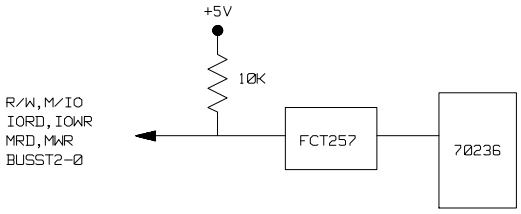
These singals are connected to 70136 through FCT244 and 4.7K ohm pull-up and 10K ohm pull-down registers.



Target System Interface (70236/70236A)

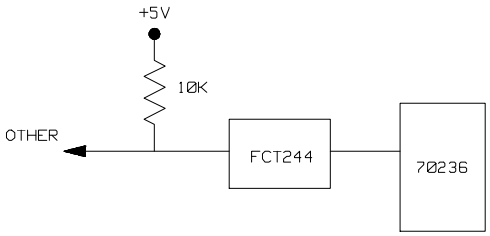
R/$\overline{W}$ M/$\overline{IO}$
$\overline{IORD}$ $\overline{IOWR}$
MRD MWR
BUSST2-0

These singals are connected to 70236/70236A through FCT257 and 10K ohm pull-up register.

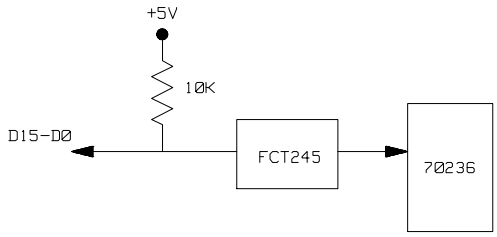


OTHER(INPUT)

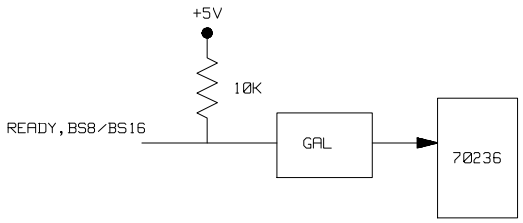
These singals are connected to 70236/70236A through FCT244 and 10K ohm pull-up register.



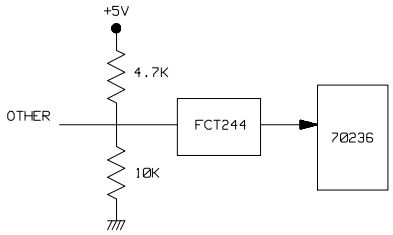
D15-D0 These singals are connected to 70236/70236A through FCT245 and 10K ohm pull-up register.



READY
BS8/BS16 These singals are connected to 70236/70236A through GAL and 10K ohm pull-up register.



OTHER(OUTPUT) These singals are connected to 70236/70236A through FCT244 and 4.7K ohm pull-up and 10K ohm pull-down registers.



4

Configuring the Emulator

Introduction

Your 70136 emulator can be used in all stages of target system development. For instance, you can run the emulator out-of-circuit when developing target system software, or you can use the emulator in-circuit when integrating software with target system hardware. Emulation memory can be used in place of, or along with, target system memory. You can use the emulator's internal clock or the target system clock. You can execute target programs in real-time or allow emulator execution to be diverted into the monitor when commands request access of target system resources (target system memory, register contents, etc.)

The emulator is a flexible instrument and it may be configured to suit your needs at any stage of the development process. This chapter describes the options available when configuring the 70136 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 emulation monitor type.
- Specifying the monitor location.
- Mapping memory.

Emulator Pod Configuration:

- Enabling RESET inputs from target system.
- Enabling NMI inputs from target system.
- Enabling READY inputs from target system.
- Selecting algorithm for physical run addresses.
- Selecting target memory and I/O access size.
- Enabling background cycles to target system (70136 Emulator).
- Enabling background cycles to target system (70236/70236A Emulator).
- Selecting emulation memory bus sizing signal.
- Selecting target memory bus sizing signal.
- Enabling break on reading page registers.
- Selecting the **AEX** (Address Extension) signal while background.
- Selecting FPU (Floating Point Unit) type for disassembly.

- Enabling responding to target HLDRQ (Hold Request) during background cycles
  (70236/70236A Emulator only).
- Selecting the number of wait states for internal DMA cycles
  (70236/70236A Emulator only).
- Enable internal DMA cycles during background cycles
  (70236/70236A Emulator only).

Debug/Trace Configuration:

- Enabling breaks on writes to ROM.
- Specifying tracing of foreground/background cycles.
- Specifying tracing of internal DMA cycles
  (70236/70236A Emulator only).
- Specifying tracing of refresh cycles
  (70236/70236A Emulator only).
- Specifying tracing of dummy cycles during HALT acknowledge
  (70236 Emulator only).

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

External Analyzer Configuration: See the *Analyzer Softkey Interface User's Guide.*

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 internal clock oscillator as the emulator clock source. The internal clock speed of the 70136, 70236 and 70236A are 16 MHz (system clock).
external	Selects an external target system clock source, from 2 MHz up to 16 MHz can be entered in using the 70136 emulator. In using the 70236 emulator, from 4 MHz to 32 MHz can be entered. In using the 70236A emulator, from 4 MHz to 40 MHz can be entered.

Note

When the 70136 emulator is plugged into the target system, you should use the external target system clock source to synchronize the emulator with the target system.

Note

Changing the clock source drives the emulator into the reset state. If you answer "yes" to the "Enter monitor after configuration?" question that follows, the emulator resets (due to the clock source change) then breaks into the monitor when the configuration is saved.

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.
- Display/modify I/O.

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.

Note

When program execution should take place in real-time and the emulator should break to the monitor to read page registers (refer to "Enable break on reading page registers?" section in this chapter), the following commands are not allowed with using physical or :<offset> address expression.

- Display/modify emulation memory.

Memory Configuration

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

program execution can take place in the "background" or "foreground" emulator modes.

In the *foreground* emulator mode, the emulator operates as would the target system processor.
In the *background* emulator mode, foreground execution is suspended so that the emulation processor may be used for communication with the system controller, typically to perform tasks which access target system resources.

A *background monitor* program operates entirely in the background emulator mode; that is, the monitor program does not execute as if it were part of the target program. The background monitor does not take up any processor address space and does not need to be linked to the target program. The monitor resides in dedicated background memory.

A *foreground monitor* program performs its tasks in the foreground emulator mode; that is, the monitor program executes as if it were part of the target program. Breaks into the monitor always put the emulator in the background mode; however, foreground monitors switch back to the foreground mode before performing monitor functions.

Note 

Halt instructions will cause "processor halted" emulation status.

The emulator breaks to the monitor when you display/modify registers, target system memory, or I/O in "processor halted" emulation status. Refer to "Trace dummy cycles during HALT acknowledge?" section in this chapter.

Note 

All memory mapper terms are deleted when the monitor type is changed!

---

background	The default emulator configuration selects the background monitor. A memory overlay is created and the background monitor is loaded into that area.

Note

While running in background monitor, the 70136 emulator ignores target system reset.

When the background monitor is selected, the execution of the monitor is hidden from the target system (except for background cycles). When you select the background monitor and the current monitor type is "foreground", you are asked the next question.

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

This question will be asked if you change the monitor type (in this case, you have changed the monitor type from "foreground" to "background"). This question reminds you that the map will be reset and allows you to confirm your decision.

no	The memory map is not reset, and the monitor type is not changed.
yes	This memory map is reset due to the change in monitor type.

2. Background monitor location?

This configuration allows you to specify the location of the background monitor program. When entering monitor block addresses, you must only specify addresses on 4K boundaries; otherwise, the configuration will be invalid, and the previous configuration will be restored. The location of background monitor may be important because background cycles of the 70136 emulator can be visible to the target system In default, the monitor is located on 0FF000 hex through 0FFFFF hex.

foreground — When you select the foreground monitor, processor address space is taken up. The foreground monitor takes up 4K bytes of memory. When the foreground monitor is selected, breaking into the monitor still occurs in a brief background state, but the rest of the monitor program, the saving of registers and the dispatching of emulation commands, is executed in foreground.

Note

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

When you select the foreground monitor and the current monitor type is "background", you are asked the next question.

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

This question will be asked if you change the monitor type (in this case, you have changed the monitor type from "background" to "foreground"). This question reminds you that the map will be reset and allows you to confirm your decision.

no — The memory map is not reset, and the monitor type is not changed.

yes — This memory map is reset due to the change in monitor type.

2. Foreground monitor location?

You can relocate the monitor to any 4K byte boundary. The location of a foreground monitor is important because it will occupy part of the processor address space. Foreground monitor locations must not overlap the locations of target system programs. When entering monitor block addresses, you must only specify addresses on 4K byte boundaries; otherwise, the configuration will be invalid, and the previous configuration will be restored.

Note

You should not load the foreground monitor provided with the 70136 emulator at the base address 0 or 0ff000 hex; the 70136 microprocessor's vector table is located.
And, You can not load the foreground monitor at the base address over 100000 hex.

3. Monitor filename?

This question allows you to specify the name of the foreground monitor program absolute file. Remember that you must assemble and link your foreground monitor starting at the 4K byte boundary specified for the previous "Foreground monitor location?" question.

The monitor program will loaded after you have answered all the configuration questions.

Only the 4 kilobytes of memory reserved for the monitor are loaded at the end of configuration; therefore, you should not link the foreground monitor to the user program. If it is important that the symbol database contain both monitor and user program symbols, you can create a different absolute file in which the monitor and user program are linked. Then, you can load this file after configuration.

Using the Foreground Monitor. When using the foreground monitor, your program should set up a stack. The foreground monitor assumes that there is a stack in the foreground program, and this stack is used to save PS, PC, and PSW upon entry into the monitor.

Mapping Memory

Depending on the emulator model number, emulation memory consists of 128k, 512k, 1M or 2M bytes, mappable in 256 byte blocks. However, you may use 124k, 508k, 1020k, or 2044k bytes of emulation memory for your target system, because 4 kilobytes of emulation memory specified by the "Foreground or background monitor location?" question is required for the execution of the monitor. The emulation memory system does not introduce wait states.

Note

You can insert wait states on accessing emulation memory. Refer to the "Enable READY input from the target system?" section in this chapter.

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

When a foreground monitor selected, a 4 kilobyte block is automatically mapped at the address specified by the "Foreground monitor location?" question.

Note

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

Blocks of memory can also be characterized as guarded memory. Guarded memory accesses will generate "break to monitor" requests. Writes to ROM will generate "break to monitor" requests if the "Enable breaks on writes to ROM?" configuration item is enabled (see the "Debug/Trace Configuration" section which follows).

Determining the Locations to be Mapped

Typically, assemblers generate relocatable files and linkers combine relocatable files to form the absolute file. The linker load map listing will show what locations your program will occupy in memory.

Defining the data bus size

The data bus size for memory accesses can be defined in this command. For example, enter the following command to map memory.

```
0h thru 7ffh emulation rom size16 <RETURN>
800h thru 9ffh emulation ram size8 <RETURN>
default target ram <RETURN>
end <RETURN>
```

From 0 hex through 7ff hex is mapped as emulation ROM with 16-bit data bus; from 800 hex through 9ff hex is mapped as emulation RAM with 8-bit data bus; the other memory ranges are mapped as target RAM with 16-bit data bus (if the data bus size is not specified in this command, the address ranges will be mapped with 16-bit data bus by default).

Note

The data bus size for memory accesses also can be defined from the BS8/BS16 input of the target system.
Refer to the "Enable emulation/target memory bus sizing signal" section.

Note

The data bus size of I/O accesses (external I/O only) is defined from the BS8/BS16 input of the target system.

Emulator Pod Configuration

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

Modify emulator pod configuration?

Enable RESET inputs from target system?

The 70136 emulator can respond or ignore target system reset while running in user program or waiting for target system reset (refer to "run from reset" command in the *Softkey Interface Reference* manual). While running in background monitor, the 70136 emulator ignores target system reset completely independent on this setting.

yes	Specify that, this is a default configuration, make the emulator to respond to reset from target system. In this configuration, emulator will accept reset and execute from reset vector (0FFFF0 hex) as same manner as actual microprocessor after reset is inactivated.
no	The emulator ignores reset signal from target system completely, even while in foreground (executing user program).

Enable NMI inputs from target system?

This question allows you to specify whether or not the emulation processor accepts NMI signal generated by the target system.

yes	The emulator accepts NMI signal generated by the target system. When the NMI is accepted, the emulator calls the NMI procedure as actual microprocessor. Therefore, you need to set up the NMI vector table, if you want to use the NMI interrupt.
no	The emulator ignores NMI signal from target system completely.

Note

You should not use step command when if target system can generates NMI.

When the emulator accepts NMI input in stepping, the following error message will be shown.

```
ERROR : Stepping failed
```

In this case, you should configure that the emulator ignores NMI input from the target system in this configuration setting.

Enable READY inputs from target system?

High-speed emulation memory provides no-wait-state operation. However, the emulator may optionally respond to the target system ready line while emulation memory is being accessed.

no	When the ready relationship is not locked to the target system, emulation memory accesses ignore ready signals from the target system (no wait states are inserted).
yes	When the ready relationship is locked to the target system, emulation memory accesses honor ready signals from the target system (wait states are inserted if requested).

Select Algorithm for physical run addresses

The run and step commands allow you to enter addresses in either logical form (segment:offset, e.g., 0F000H:0000H) or physical form (e.g., 0F000H). When a physical address (non-segmented) is entered with either a run or step command, the emulator must convert it to a logical (segment:offset) address.

minseg — Specifies that the physical run address is converted such that the low 16 bits of the address become the offset value. The physical address is right-shifted 4 bits and ANDed with 0F000H to yield the segment value.

```
logical_addr = ((phys_addr >> 4) & 0xf000):(phys_addr & 0xffff)
```

maxseg — Specifies that the low 4 bits of the physical address become the offset. The physical address is right-shifted 4 bits to yield the segment value.

```
logical_addr =  (phys_addr >> 4):(phys_addr & 0xf)
```

curseg — Specifies that the value entered with either a run or step command (0 thru 0ffff hex) becomes the offset. In this selecting, the current segment value is not changed.

```
logical_addr = (current segment):(entered value)
```

If you use logical addresses other than the three methods which follow, you must enter run and step addresses in logical form.

Select target memory and I/O access size

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

Bytes	Selecting the byte access mode specifies that the emulator will access target memory using upper and lower byte cycles (one byte at a time).
Words	Selecting the word access mode specifies that the emulator will access target memory using word cycles (one word at a time) at an even address. When the emulator read or write odd number of byte data, the emulator will read or write the last byte data using byte cycle. At an odd address, the emulator will access target memory using byte cycles.

The default emulator configuration selects the **byte** access size at power up initialization. Access mode specifications are saved; that is, when a command changes the access mode, the new access mode becomes the current default.

Enable background cycles to target system? (70136 Emulator)

This configuration allows you to select whether or not the 70136 emulator will drive the bus status lines (M/$\overline{IO}$, BUSST1, BUSST0, R/$\overline{W}$) on all background monitor cycles to the target system.

Note

All address bus (A23 to A0), $\overline{BCYST}$, and $\overline{DSTB}$ are always driven to the target system on all background monitor cycles independent on this configuration item.

All data bus (D15 to D0) are never driven to the target system on all background monitor cycles.

---

yes	Specifies that the emulator will drive the bus status lines to the target system. All cycles appear to the target system as memory read cycles (M/$\overline{IO}$ = 1, BUSST1 = 0, BUSST0 = 1, R/$\overline{W}$ = 1) from the address range of the monitor. It is possible to place the monitor at different locations if read cycles from the current range cause an undesired interaction (see the "Background monitor location?" in Memory Configuration).
no	When you select this option, the bus status lines (M/$\overline{IO}$, BUSST1, BUSST0, R/$\overline{W}$) are not driven to the target system.

Enable background cycles to target system? (70236/70236A Emulator)

This configuration allows you to select whether or not the 70236 emulator will drive the bus status lines (M/$\overline{IO}$, R/$\overline{W}$, BUSST2, BUSST1, BUSST0, $\overline{UBE}$, $\overline{BCYST}$, $\overline{DSTB}$) on all background monitor cycles to the target system.

Note

All address bus (A23 to A0), AEX, $\overline{BUSLOCK}$, $\overline{REFRQ}$ and HLDAK are always driven to the target system on all background monitor cycles independent on this configuration item.

All data bus (D15 to D0) are never driven to the target system on all background monitor cycles.

Note

The emulator will drive all bus lines on all DMA and refresh cycles in the background monitor to the target system.

---

yes	Specifies that the emulator will drive the bus status lines to the target system. All cycles appear to the target system as read cycles for memory (M/$\overline{IO}$ = 1, R/$\overline{W}$ = 1) from the address range of the monitor. It is possible to place the monitor at different locations if read cycles from the current range cause an undesired interaction (see the "Background monitor location?" in Memory Configuration).
no	When you select this option, the bus status lines (M/$\overline{IO}$, R/$\overline{W}$, BUSST2, BUSST1, BUSST0, $\overline{UBE}$, $\overline{BCYST}$, $\overline{DSTB}$) are not driven to the target system.

Select emulation memory bus sizing signal

emulator	Specifies that the bus size of emulation memory is selected from the setting of the map configuration. Refer to the "Mapping Memory" command description in Memory Configuration.
target	Specifies that the bus size of emulation memory is defined from the BS8/BS16 input of the target system.

Select target memory bus sizing signal

target	Specifies that the bus size of target memory is defined from the BS8/BS16 input of the target system.
emulator	Specifies that the bus size of target memory is selected from the setting of the map configuration. Refer to the "Mapping Memory" command description in Memory Configuration.

Note 

The data bus size of I/O accesses is only defined from the BS8/BS16 input of the target system.

Enable break on reading page registers?

This configuration item allows you to specify whether the emulator should break to the monitor to read page registers or whether the emulator should use the copy of page registers when the emulation system will convert physical address to extended address in the following commands.

- Display/modify memory with entering physical or :<OFFSET> address expression.
- Modify software breakpoints

yes — Specifies that the emulator should break to the monitor to get the current value of page registers on accesses to emulation/target memory.

no — Specifies that the emulator should use the copy of page registers which is renewed at breaking to the monitor or changing the value of page registers with using the following Softkey Interface command (refer to the *Softkey Interface Reference* manual).

```
modify register <PGR 1 .. PGR 64>
```

Note

You may specify not to break to the monitor to read page registers when you only use the normal address mode in your program or the value of page registers is not changed after initalizing while executing your program.

Select AEX signal while background

Select the **AEX** (Address Extension) signal level in background monitor cycles. This configuration option allows you to select the **AEX** signal level which is driven to the target system while in the background monitor cycles.

hold	Specifies that the emulator will hold the **AEX** signal with the level dependent on the last foreground address mode just before entering background monitor. When the program is running on normal address mode, the emulator will hold the **AEX** signal level low while the background monitor cycles with this configuration.
unhold	Specifies that the emulator will drive the **AEX** signal with the level dependent on the address mode in background monitor cycles. When you use the extended address in an emulation command in background monitor, the **AEX** signal will be driven to high level with this configuration.

Select FPU type for disassembly

Select assembler mnemonics for FPU (Floating Point Unit) to display memory.

72291	Specifies that mnemonics for NEC uPD72291 floating point processor will be used to display memory.
80287	Specifies that mnemonics for Intel 80287 numeric processor extension will be used to display memory.

Respond to target HLDRQ during background operation? (70236/70236A Emulator Only)

This configuration allows you to specify whether or not the emulator accepts HLDRQ (Hold Request) signal generated by the target system in background.

no	The emulator ignores HLDRQ signal from target system completely in background.
yes	The emulator accepts HLDRQ signal. When the HLDRQ is accepted, the emulator will respond as actual microprocessor.

Wait states for internal DMA cycles (70236/70236A Emulator Only)

When you want to trace internal DMA cycles correctly with using the emulator, you must set the number of wait states for internal DMA cycles.

The number is the same as the value of DMAW (Wait for the DMA cycle) of the WCY4 (programmable wait, cycle 4) register (I/O address FFF6 hex). See the "Trace internal DMA cycles?" in Trace configuration.

Enabling internal DMA during background operation? (70236/70236A Emulator Only)

This configuration allows you to specify whether or not the emulation processor’s internal DMA is allowed while in background.

yes	The internal DMA is allowed while in background.
no	The internal DMA is not allowed while in background.

Debug/Trace Configuration

The debug/trace configuration questions allows you to specify breaks on writes to ROM, enable/disable the software breakpoints feature, and specify that the analyzer trace foreground/background execution. To access the debug/trace 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; however, they cannot prevent writes to target system RAM locations which are mapped as ROM, even 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.

Note

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

Character	Information
N	Normal address mode (foreground)
E	Extended address mode (foreground)
M	Monitor cycle (background)

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.)

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 Internal DMA cycles? (70236/70236A Emulator only)

This question allows you to specify whether or not the analyzer trace the 70236 emulation processor's internal DMA cycles.

yes — Specifies that the analyzer will trace the internal DMA cycles.

no — Specifies that the analyzer will not trace the internal DMA cycles.

Trace refresh cycles? (70236/70236A Emulator only)

This question allows you to specify whether or not the analyzer trace the emulation processor's refresh cycles.

yes	Specifies that the analyzer will trace the refresh cycles.
no	Specifies that the analyzer will not trace the refresh cycles.

Trace dummy cycles during HALT acknowledge? (70236 Emulator only)

Whenever breaks occur during the emulator is halted, the HALT acknowledge cycle will be occurred one more time.
This question allows you to specify whether or not the emulation analyzer trace this HALT acknowledge cycles occurred by the breaks during the emulator is halted.

no	Specifies that the analyzer will not trace the dummy HALT acknowledge cycles.
yes	Specifies that the analyzer will trace the dummy HALT acknowledge cycles.

Note

Whenever breaks occur during the 70236 emulator is halted, the HALT acknowledge cycle will be occurred one more time.
The emulation analyzer always traces this HALT acknowledge cycles occurred by the breaks during the emulator is halted.

Note

If the 70236A emulator breaks occur during the emulator is halted, the HALT acknowledge cycle can not be occurred one more time. The emulator keeps into the monitor.
So this configration is not avaibale for the 70236A emulator.

Simulated I/O Configuration

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

External Analyzer Configuration

The external analyzer configuration options are described in the *Analyzer Softkey Interface User's Guide*.

Interactive Measurement Configuration

The interactive measurement configuration questions are described in the chapter on coordinated measurements in the *Softkey Interface Reference* manual. Examples of coordinated measurements that can be performed between the emulator and the emulation analyzer are found in the "Using the Emulator" chapter.

Saving a Configuration

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

Configuration file name? <FILE>

The name of the last configuration file is shown, or 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 will be saved to two files; the filename specified with extensions of ".EA" and ".EB". 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.

Ending out of 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

Configuration files which have been previously saved 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; it saves you from having 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

The "Getting Started" chapter shows you how to use the basic features of the 70136 emulator. This chapter describes the more in-depth features of the emulator.

This chapter discusses:

- Register names and classes.
- Features available via "pod_command".

This chapter shows you how to:

- Store the contents of memory into absolute files.
- Make coordinated measurements.

Register Names and Classes (70136 Emulator)

The following register names and classes are used with the display/modify registers commands in 70136 emulator.

BASIC(*) class

Register name	Description
AW, BW CW, DW BP, IX, IY DS0, DS1, SS SP, PC, PS, PSW	BASIC registers.

PGR class

(page registers)

Register name	Description
PGR1	PGR 1 register
PGR2	PGR 2 register
:	:
:	:
PGR63	PGR 63 register
PGR64	PGR 64 register
XAM	XAM register (Read only)

Register Names and Classes (70236/70236A Emulator)

The following register names and classes are used with the display/modify registers commands in 70236 emulator.

BASIC(*) class

Register name	Description
AW, BW	BASIC registers.
CW, DW	
BP, IX, IY	
DS0, DS1, SS	
SP, PC, PS, PSW	

PGR class

(Page registers)

Register name	Description
PGR1	PGR 1 register
PGR2	PGR 2 register
:	:
:	:
PGR63	PGR 63 register
PGR64	PGR 64 register
XAM	XAM register (Read only)

SIO class (System I/O registers)

Register name	Description
BSEL	Bank selection register
BADR	Bank address register
BRC	Boud rate counter
WMB0	Programmable wait, memory boundary 0 register
WCY1	Programmable wait, cycle 1 register
WCY0	Programmable wait, cycle 0 register
WAC	Programmable wait, memory address control register
TCKS	Timer clock selection register
SBCR	Stand-by control register
REFC	Refresh control register
WMB1	Programmable wait, memory boundary 1 register
WCY2	Programmable wait, cycle 2 register
WCY3	Programmable wait, cycle 3 register
WCY4	Programmable wait, cycle 4 register
SULA	SCU low address register
TULA	TCU low address register
IULA	ICU low address register
DULA	DMAU low address register
OPHA	On-chip peripheral high address register
OPSEL	On-chip peripheral selection register
SCTL	System control register

ICU class (Interrupt Control Unit registers)

Register name	Description	
IMKW	Interrupt mask word register	
IRQ	Interrupt request register	(Read only)
IIS	Interrupt in-service register	(Read only)
IPOL	Interrupt polling register	(Read only)
IPFW	Interrupt priority and finish word register (Write only)	
IMDW	Interrupt mode word register	(Write only)
IIW1	Interrupt initialize word 1 register	(Write only)
IIW2	Interrupt initialize word 2 register	(Write only)
IIW3	Interrupt initialize word 3 register	(Write only)
IIW4	Interrupt initialize word 4 register	(Write only)

Caution

When **ipol** register is displayed, interruptis are suspended until the FI command is published.

TCU class (Timer Control Unit registers)

Register name	Description	
TCT0	Timer/counter 0 register	
TST0	Timer status 0 register	(Read only)
TCT1	Timer/counter 1 register	
TST1	Timer status 1 register	(Read only)
TCT2	Timer/counter 2 register	
TST2	Timer status 2 register	(Read only)
TMD	Timer/counter mode register	(Write only)

SCU class

(Serial Control Unit registers)

Register name	Description	
SRB	Serial receive data buffer	(Read only)
SST	Serial status register	(Read only)
STB	Serial transmit data buffer	(Write only)
SCM	Serial command register	(Write only)
SMD	Serial mode register	(Write only)
SIMK	Serial interrupt mask register	(Write only)

DMA71 class

(DMA Control Unit registers (for uPD71071 mode)

Register name	Description	
DICM	DMA initialize register	(Write only)
DCH	DMA channel register	
DBC_DCC0	DMA base/current count register channel 0	
DBC_DCC1	DMA base/current count register channel 1	
DBC_DCC2	DMA base/current count register channel 2	
DBC_DCC3	DMA base/current count register channel 3	
DBA_DCA0	DMA base/current address register channel 0	
DBA_DCA1	DMA base/current address register channel 1	
DBA_DCA2	DMA base/current address register channel 2	
DBA_DCA3	DMA base/current address register channel 3	
DMD0	DMA mode control register channel 0	
DMD1	DMA mode control register channel 1	
DMD2	DMA mode control register channel 2	
DMD3	DMA mode control register channel 3	
DDC	DMA device control register	
DST	DMA status register	(Read only)
DMK	DMA mask register	

DMA37 class (DMA Control Unit register (for uPD71037 mode)

Register name	Description
CMD	DMA read status/write command register
BANK0	DMA bank register channel 0
BANK1	DMA bank register channel 1
BANK2	DMA bank register channel 2
BANK3	DMA bank register channel 3
ADR0	DMA current address register channel 0
ADR1	DMA current address register channel 1
ADR2	DMA current address register channel 2
ADR3	DMA current address register channel 3
CNT0	DMA current count register channel 0
CNT1	DMA current count register channel 1
CNT2	DMA current count register channel 2
CNT3	DMA current count register channel 3
SFRQ	Software DMA write request register (Write only)
SMSK	DMA write single mask register(Write only)
MODE	DMA write mode register(Write only)
CLBP	DMA clear byte pointer F/F (Write only)
INIT	DMA initialize register (Write only)
CMSK	DMA clear mask register (Write only)
AMSK	DMA write all mask register bit (Write only)

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. the 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.

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 of the most 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.

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

Note

Be careful when using the "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. Be aware that what you see in "modify configuration" will NOT reflect the HP 64700 pod's configuration if you change the pod's configuration with this command. Also, commands which 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.

Storing Memory Contents to an Absolute File

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

```
store memory 800h thru 84fh to absfile
<RETURN>
```

The command above causes the contents of memory locations 800H-84FH to be stored in the absolute file "absfile.X". Notice that the ".X" extension is appended to the specified filename.

Coordinated Measurements

For information on coordinated measurements and how to use them, refer to the "Coordinated Measurements" chapter in the *Softkey Interface Reference* manual.

A

Using the Foreground Monitor

Introduction

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.

The foreground monitors are supplied with the emulation software and can be found in the following path:

```
/usr/hp64000/monitor/*
```

The monitor programs named **Nfmon70136.s** and **Nfmon70236.s** are for the HP 64873 V series AxLS Cross Assembler/Linker.

Note

Use the appropriate monitor; "Nfmon70136.s" for the 70136 emulator and "Nfmon70236.s" for the 70236 and 70236A emulator. "Nfmon70136.s" foreground monitor program is used in this example. If your emulator is for the 70236 or 70236A, read this appendix by replacing "Nfmon70136" with "Nfmon70236".

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 fully 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. Foreground monitors allow the emulator to service real-time events, such as interrupts, while executing in the monitor. 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 (see the "Configuring the Emulator" chapter and the examples in this appendix).

You may link the foreground monitor with your code. However, if possible, linking the monitor separately is preferred. This allows the monitor to be downloaded before the rest of your program. Linking monitor programs separately is more work initially, but it should prove worthwhile overall, since the monitor can then be loaded efficiently during the configuration process at the beginning of a session.

An Example Using the Foreground Monitor

In the following example, we will illustrate how to use a foreground monitor with the sample program from the "Getting Started" chapter. By using the emulation analyzer, we will also show how the emulator switches from state to state using a foreground monitor.

For this example, we will be using the foreground monitor for the HP 64873 V series AxLS Cross Assembler/Linker. We will locate the monitor at 1000H; the sample program will be located at 10000H and 80000H.

```
$ cp /usr/hp64000/monitor/Nfmon70136.s .
<RETURN>
```

Modify EQU Statement

To use the monitor, you must modify the EQU statement near the top of the monitor listing to point to the base address where the monitor will be loaded.

```
$ chmod 644 Nfmon70136.s <RETURN>
$ vi Nfmon70136.s <RETURN>
```

Modifying Location of the Foreground Monitor

In this case, we will load the monitor at 1000H, so the modified EQU statement looks like this:

```
MONSEGMENT     EQU   00100H
```

You can load the monitor at any base address on a 4K byte boundary.

Note

You should not load the foreground monitor provided with the 70136 emulator at the base address 0 or 0ff000 hex; the 70136 microprocessor's vector table is located.
And, You can not load the foreground monitor at the base address over 100000 hex.

Assemble and Link the Monitor

You can assemble and link the foreground monitor program with the following commands (which assume that **/usr/hp64000/bin** is defined in the PATH environment variable):

```
$ asv33 -Lh Nfmon70136.s > Nfmon70136.lis
<RETURN>
$ ldv33 -c Nfmon70136.k -Lh > Nfmon70136.map
<RETURN>
```

The "Nfmon70136.k" linker command file is shown below.

```
LOAD      Nfmon70136.o
SEG       ??DATA1/??INIT=001ffdH
END
```

The "??DATA1/??INIT" is used in the HP 64873 V series AxLS Cross Assembler/Linker. You should set the "??DATA1/??INIT" to the value added the offset value (0FFDH) to the foreground monitor address (In this example, 1000H). When you want to relocate the foreground monitor, you should modify the "??DATA1/??INIT" value in the linker command file for the new foreground monitor address.

If you aren't ready to use the sample program, do that now. Refer to the "Getting Started" chapter to copy the sample program files to the current directory.

Modifying the Emulator Configuration

The following assumes you are modifying the default emulator configuration (that is, the configuration present after initial entry into the emulator or entry after a previous exit using "end release_system"). Enter all the default answers except those shown below.

Modify memory configuration? yes

You must modify the memory configuration so that you can select the foreground monitor and map memory.

Monitor type? foreground

Specifies that you will be using a foreground monitor program.

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

You must answer this question as shown to change the monitor type to foreground.

Monitor address? 1000h

Specifies that the monitor will reside in the 4K byte block from 1000H through 1FFFH.

Monitor file name? Nfmon70136

Enter the name of the foreground monitor absolute file. This file will be loaded at the end of configuration.

Mapping Memory for the Example

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

```
0h thru 03ffh emulation ram <RETURN>
10000h thru 1f3ffh emulation ram <RETURN>
80000h thru 8f7ffh emulation rom <RETURN>
default target ram <RETURN>
end <RETURN>
```

Modify debug/trace options? yes

You must answer this question as shown to access and modify the question below.

Trace background or foreground operation? both

Later in this chapter, trace examples show transitions from reset into the foreground monitor, from the monitor to the user program, and from the user program back into the monitor. Since the foreground monitor is actually entered via a few cycles in the emulator's built-in background monitor, we need to be able to view the background states. Answering this configuration question as shown allows both foreground and background emulation processor cycles to appear in the trace.

Configuration file name? fmoncfg

If you wish to save the configuration specified above, answer this question as shown.

Load the Program Code

Now it's time to load the sample program. You can load the sample program with the following command:

```
load cmd_rds <RETURN>
```

Tracing from Reset to Break

We want to see the monitor's transition from the reset state to running in the foreground monitor. First, put the emulator into its reset state with the command:

```
reset <RETURN>
```

The 70136 emulator breaks to the foreground monitor via a few background cycles. You can see the transition between reset and foreground monitor execution. Enter following command.

```
trace <RETURN>
```

After entering the command above, the "Emulation trace started" message appears on the status line. Enter the following command to break into the monitor.

```
break <RETURN>
```

The status line now shows that the emulator is "Running in monitor" and that the "Emulation trace complete". Enter the following command to display the trace.

```
display trace <RETURN>
```

```
Trace List                  Offset=0
Label:  Address   Data              Opcode or Status              time count
Base:     hex     hex                   mnemonic                  relative
after    0FFFF4     FFFF    FFFF  prefetch                  N      -----------
+001     000008     0310    0310  memory read               M        200     nS
+002     00000A     0100    0100  memory read               M        160     nS
+003     001310     C62E    C62E  prefetch                  M        320     nS
+004     001312     0E06    0E06  prefetch                  M        200     nS
+005     000FEE     F002    F002  undefined                 M        200     nS
+006     000FEC     FFFF    FFFF  undefined                 M        160     nS
+007     000FEA     0000    0000  undefined                 M        200     nS
+008     001314     0002    0002  prefetch                  M        120     nS
+009     001316     A32E    A32E  prefetch                  M        120     nS
+010     001310     A32E  MOV     PS:020E,#00                         80.    nS
+011     001318     00E6    00E6  prefetch                  M         40.    nS
+012     00120E     0000    xx00  memory write             M        200     nS
+013     001316     0000  MOV     PS:00E6,AW                          80.    nS
+014     00131A     892E    892E  prefetch                  M         40.    nS

STATUS:   N70136--Running in monitor      Emulation trace complete______...R....
display trace


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

The trace listing shows that the processor began executing code; it executed in background monitor. The "M"s in the trace listing indicate the background monitor cycles.

To see the transition from background monitor to the foreground monitor, press the <NEXT> key to page down until the background cycles go.

```
Trace List                  Offset=0
Label:  Address   Data               Opcode or Status              time count
Base:     hex     hex                    mnemonic                   relative
+141     0013F4     0000    0000  prefetch                    M         80.    nS
+142     0013EE     0000  NOP                                          80.    nS
+143     0013EF     FFBE  NOP                                         160     nS
+144     0013F6     0000    0000  prefetch                    M        120     nS
+145     0013F0     0000  NOP                                          80.    nS
+146     0013F1     FFFF  illegal opcode, data = 0F CF                240     nS
+147     0013F8     0000    0000  prefetch                    M         80.    nS
+148     000FEA     0500    0500  undefined                   M        160     nS
+149     000FEC     0100    0100  undefined                   M        200     nS
+150     000FEE     F002    F002  undefined                   M        200     nS
+151     001500     8C2E    8C2E  prefetch                    N        160     nS
+152     001502     FA16    FA16  prefetch                    N        160     nS
+153     001504     2E00    2E00  prefetch                    N        120     nS
+154     001506     2689    2689  prefetch                    N        120     nS
+155     001500     2689  MOV     PS:00FA,SS                           40.    nS

STATUS:   N70136--Running in monitor      Emulation trace complete_____...R....
display trace


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

You will see the transition from the background monitor to the foreground monitor in the display.

Tracing from Monitor to User Program

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

```
trace about entry <RETURN>
```

Because you'd like to see the states leading up to the transition from monitor to user program, trace "about" so that states before the trigger are captured.

Now, run the sample program:

```
run from transfer_address <RETURN>
```

Display the trace with the following command:

```
display trace  <RETURN>
```

The user program began execution at state 0. Now, you will know the processor executed the **RETI** instruction to transfer execution to the user program at state 0.

```
Trace List                  Offset=0
Label:  Address   Data               Opcode or Status              time count
Base:     hex     hex                    mnemonic                  relative
-007     001978     00CF    00CF  prefetch                  N         40.    nS
-006     0010EE     0FEA    0FEA  memory read               N        200     nS
-005     001978     0FEA  RETI                                       80.    nS
-004     00197A     0000    0000  prefetch                  N         40.    nS
-003     001FEA     0006    0006  memory read               N        200     nS
-002     001FEC     800C    800C  memory read               N        160     nS
-001     001FEE     F002    F002  memory read               N        200     nS
about    0800C6     0BEA    0BEA  prefetch                  N        200     nS
+001     0800C8     0C00    0C00  prefetch                  N        120     nS
+002     0800CA     B880    B880  prefetch                  N        120     nS
+003     0800CC     1000    1000  prefetch                  N        120     nS
+004     0800C6     1000  BR      FAR PTR 800CB                       80.    nS
+005     0800CE     D88E    D88E  prefetch                  N         40.    nS
+006     0800CB     B88E    B8xx  prefetch                  N        120     nS
+007     0800CC     1000    1000  prefetch                  N        120     nS

STATUS:   N70136--Running user program    Emulation trace complete______...R....
display trace


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

Tracing from User Program to Break

You can trace the execution from the user program to the foreground monitor due to a break condition. Since the foreground monitor occupies the address range from 1000h through 1fffh, we can simply trigger on any access to that range.

```
trace about range 1000h thru 1fffh <RETURN>
```

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

```
break <RETURN>
```

Now, display the trace with the following command:

```
display trace <RETURN>
```

Now, the trace listing shows that the processor entered the background state to make the transition.

```
Trace List                  Offset=0
Label:  Address   Data               Opcode or Status              time count
Base:     hex     hex                    mnemonic                   relative
-007     08005E     8A90    8A90  prefetch                  N          40.    nS
-006     08005C     8A90  BR      SHORT 080051                         80.    nS
-005     080060     FE46    FE46  prefetch                  N         120     nS
-004     080051     9046    90xx  prefetch                  N         120     nS
-003     080052     08A0    08A0  prefetch                  N         120     nS
-002     000008     0310    0310  memory read               M         280     nS
-001     00000A     0100    0100  memory read               M         160     nS
about    001310     C62E    C62E  prefetch                  M         320     nS
+001     001312     0E06    0E06  prefetch                  M         200     nS
+002     019004     F246    F246  undefined                 M         160     nS
+003     019002     8000    8000  undefined                 M         200     nS
+004     019000     0051    0051  undefined                 M         200     nS
+005     001314     0002    0002  prefetch                  M         120     nS
+006     001316     A32E    A32E  prefetch                  M         120     nS
+007     001310     A32E  MOV     PS:020E,#00                          80.    nS

STATUS:   N70136--Running in monitor       Emulation trace complete_____...R....
display trace


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

Single Step and Foreground Monitors

To use the "step" command to step through processor instructions with the foreground monitor listed in this chapter, you must modify the processor's interrupt vector table. The entry that you **must** modify is the "BRK flag" interrupt vector, located at 4H thru 7H. The "BRK flag" interrupt vector must point to the identifier UEE_BRK_FLAG in the foreground monitor. For example, to modify the "BRK flag" interrupt vector, enter the following commands:

```
load symbols Nfmon70136 <RETURN>
display local_symbols_in Nfmon70136: <RETURN>
```

To see the value of UEE_BRK_FLAG, press the <NEXT> key to page down until the UEE_BRK_FLAG is displayed.
You will see that the value of UEE_BRK_FLAG is 0100:0A09 hex.
To modify the "BRK flag" interrupt vector to point to the UEE_BRK_FLAG, enter the following command:

```
modify memory 4h words to 0A09H,0100H
<RETURN>
```

Now you can use the step feature. Enter:

```
display registers <RETURN>
load symbols cmd_rds <RETURN>
step from transfer_address <RETURN>
step <RETURN>
```

When you load the foreground monitor at the different base address, you should modify the "BRK flag" interrupt vector to point to the identifier UEE_BRK_FLAG with the same way.

Extended Address Mode

To use the foreground monitor in the extended mode, in defalt, you can not use page register 0, page register 64 and other one page register to locate the foreground monitor.

You **must** modify the processor's interrupt vector indicated "FGMON_VECNO" in the foreground monitor source.

You must set common stack area for the nomarl and exteded address mode, because the foreground moniter temporay move into the normal mode.

Limitations of Foreground Monitors

Listed below are limitations or restrictions present when using a foreground monitor.

Synchronized MeasurementsCMB

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

B

Using the Extended Mode

Introduction

This chapter will show you how to use the extended mode of the 70136 emulator with the Softkey Interface.

This chapter will:

- Describe the sample program used for this chapter's examples.

This chapter will show you how to:

- Load programs into emulation and target system memory.
- Enter emulation commands to view execution of the demo program.

This chapter discusses:

- Address expressions in emulation commands in the extended mode.
- Symbol hierarchy with SRU and HP-OMF V33 files.

Prerequisites

Before reading this chapter you should already know how the emulator operates. You should know how to use the Softkey Interface, and how to control the emulator from within the Softkey Interface. Refer to the "Getting Started" chapter to learn about the emulator.

A Look at the Sample Program

The sample program and the associated output files, including the HP-OMF V33 format executable files, have been shipped with the Softkey Interface; copy these files to the current directory with the following command:

```
$ cp
/usr/hp64000/demo/emul/hp64756/ext_mode/* .
<RETURN> (70136)
$ cp
/usr/hp64000/demo/emul/hp64757/ext_mode/* .
<RETURN>  (70236)
```

The following files are copied in your directory:

```
cmd_rds.c      cmd_rds.x        ex_cmd_rds.d    setup.s
cmd_rds.k      democonfig.EA    ex_cmd_rds.x    setup.x
```

The sample program consists of two separate tasks. The first task, **setup**, initializes the processor and switches into the extended mode. The **cmd_rds** task continuously reads values from **Cmd_Input**; when a value other than NULL is found, the program calls the **Write_Msg** function to copy a string to the **Msg_Dest** array.
Each task will be placed in its own 1-Megabyte memory space, with the **setup** task in the normal memory space and the **cmd_rds** task in the extended memory.

The two stand-alone absolute files are linked together by the HP 64875 V33/53 Extended Mode Locator product.
The file **ex_cmd_rds.x** is the final V33 executable file in the HP-OMF V33 file format.

Refer to the *HP 64875 NEC V33/53 Extended Mode Locator:User's Guide* to know how to assemble/link/locate the sample program.

```
;###############################################
;       Normal mode program: setup.s
;
;       Sets up all the page registers,
;       then breaks into extended mode
;
;###############################################

$modv33
                NAME    setup

                PUBLIC  main,page_init,brkcmd
                EXTRN   ?JUMP?cmd_rds?entry:FAR

vector1         EQU     20h       ;use vector 20h
pgr_addr        EQU     0ff00h    ;I/O addr of PGRs
num_pgrs        EQU     64        ;all of the PGRs
size_of_vector  EQU     4         ;4 bytes for vector
size_of_pgr     EQU     2         ;2 bytes for PGR
size_of_stack   EQU     10h       ;10h bytes for stack


program         SEGMENT AT 100h
                ASSUME  PS:program,DS0:data

;=============================================
;           RUN FROM HERE
;=============================================
main            PROC    FAR
; set up stack pointer
                MOV     AW,SEG stack_area
                MOV     SS,AW
                MOV     AW,OFFSET stack_area+size_of_stack
                MOV     SP,AW
; set up page registers
                CALL    init_pages
; set up break address
                CALL     set_vector
; fly to extended mode - never come back
brkcmd:         BRKXA    vector1
;
main            ENDP

;=============================================
;           SETUP PAGE REGISTERS
;=============================================
; assume we are in normal mode
init_pages      PROC    NEAR
; set up memory address of source
                MOV     AW,SEG page_init
                MOV     DS1,AW
                MOV     BP,OFFSET page_init
; set up I/O address of destination
                MOV     DW,pgr_addr
; set up count
                MOV     CW,num_pgrs
; now, write to the PGRs
```

Figure B-1. Sample program "setup.s"

```
loop:           MOV     AW,DS1:[BP]
                OUT     DW,AW
                INC     BP
                INC     BP
                INC     DW
                INC     DW
                DBNZ    loop
; done
                RET
init_pages      ENDP

;=========================================
;           SET ADDRESS TO FLY TO
;=========================================
set_vector      PROC     NEAR
;
addr1           EQU      vector1*size_of_vector
; set the segment of addr1 in DS0
                MOV      AW,0
                MOV      DS0,AW
; set the vector1
                MOV      AW,SEG ?JUMP?cmd_rds?entry
                MOV      DW,OFFSET  ?JUMP?cmd_rds?entry
                MOV      DS0:[WORD PTR addr1],DW
                MOV      DS0:[WORD PTR addr1+2],AW
; done
                RET
;
set_vector      ENDP

program         ENDS


data            SEGMENT WORD AT 200h
;=========================================
;           DATA TO SETUP TO PGRs
;=========================================
; elv33 sets the proper table here
page_init       DS      num_pgrs*size_of_pgr
data            ENDS


stack           SEGMENT WORD AT 300h
;=========================================
;           RESERVE STACK AREA
;=========================================
stack_area      DS      size_of_stack
stack           ENDS


                END     main
```

Figure B-1. Sample program "setup.s" (Cont'd)

Entering the Softkey Interface

Enter the Softkey Interface from the HP-UX shell with the following command.

```
 1  volatile char Cmd_Input;
 2  char Msg_Dest[0x20];
 3
 4  void Write_Msg (const char *s)
 5  {
 6          char *Dest_Ptr;
 7
 8          Dest_Ptr = Msg_Dest;
 9          while (*s != '\0')
10          {
11                  *Dest_Ptr = *s;
12                  Dest_Ptr++;
13                  s++;
14          }
15  }
16
17  main ()
18  {
19          static char Msg_A[] = "Command A Entered              ";
20          static char Msg_B[] = "Entered B Command              ";
21          static char Msg_I[] = "Invalid Command                ";
22          char c;
23
24          for (;;)
25          {
26                  Cmd_Input = '\0';
27                  while ((c = Cmd_Input) == '\0');
28                  switch (c) {
29                          case 'A' :
30                                  Write_Msg (Msg_A);
31                                  break;
32                          case 'B' :
33                                  Write_Msg (Msg_B);
34                                  break;
35                          default :
36                                  Write_Msg (Msg_I);
37                                  break;
38                  }
39          }
40  }
```

Figure B-2. Sample program "cmd_rds.c"

```
                                $ emul700 <emul_name> <RETURN>
ex_cmd_rds;

task setup
{
        taskname = setup;
        mode = normal_mode;
        base = 0;
};

task cmd_rds
{
        taskname = cmd_rds;
        mode = extended_mode;
        table = page_init;
        base = 100000h
};

end.
```

Figure B-3. The "ex_cmd_rds.d" description file

The "emul_name" in the command above is the logical emulator name given in the HP 64700 emulator device table (/usr/hp64000/etc/64700tab).
For example, the emulator name in the device table entry shown below is "v33".

```
#-------------+---------+--------------------+----+------+------+----+----+----
#             |         |                    |xpar| baud |parity|flow|stop|char
# logical name|processor|     physical       |mode| rate |      |    |bits|size
#  (14 chars) |  type   |      device        |    |      |      |XON |    |
#             |         |                    |OFF |      | NONE |RTS | 2  | 8
#-------------+---------+--------------------+----+------+------+----+----+----
#
v33            n70136     /dev/emcom23        OFF  230400  NONE  RTS   2    8
```

To load the configuration file copied above, enter the following command.

```
load configuration democonfig <RETURN>
```

Now you are ready to go ahead. Above configuration is used throughout this chapter.

Loading Absolute Files

To load the '**ex_cmd_rds.x**' executable file into the emulator, enter the following command:

```
load ex_cmd_rds.x <RETURN>
```

Symbol Hierarchy with SRU and HP-OMF V33 Files

The Symbol Retrieval Utility (SRU) allows the HP 64000-UX emulation software to read several different file formats. It is now possible to emulate using HP-OMF V33 files and get full symbol support. The HP-OMF V33 file format is generated by the HP 64875 V33/53 Extended Mode Locator product.
Since SRU is language independent, it will provide symbol information using the data in the object module format (executable) file.

Note

When you emulate using the extended mode in your program, you should generate the executable file with symbol information to load the emulator by HP 64875 V33/53 Extended Mode Locator.

The HP-OMF V33 file format provides a sophisticated view of the executable file and its symbols.
This view is more appropriate when dealing with symbols in high level languages (such as C) than when using assembly language. SRU is very resistant to language-specific symbol representations, providing a consistent view of program symbols.

The HP-OMF V33 file format uses symbol relationships that accommodate the concept of tasks.
Tasks reside at the highest level in the symbol hierarchy.
The HP-OMF V33 file format will create a task for each module which is linked by the HP 64875 V33/53 Extended Mode Locator. Tasks reside at a higher level than modules which are the unit of compilation or assembly; a task "owns" the module.

This symbol hierarchy can be seen when using HP-OMF V33 files and accessing and displaying symbols in emulation. A task is the child of the root symbol. A module is the child of a task symbol. Local symbols are accessed as children of file name symbols or children of modules symbols, depending upon the type of local symbol:

- Local symbols that are line numbers are accessed through the file name symbol.
- Local symbols that are not line numbers are accessed through the module name.

Now, with the HP-OMF V33 file, symbols are scoped as shown in the next page. (This example is for a different executable than shown above.)

Source references are created only for source lines that generate code. All source lines, however, will be displayed on the source code listing.

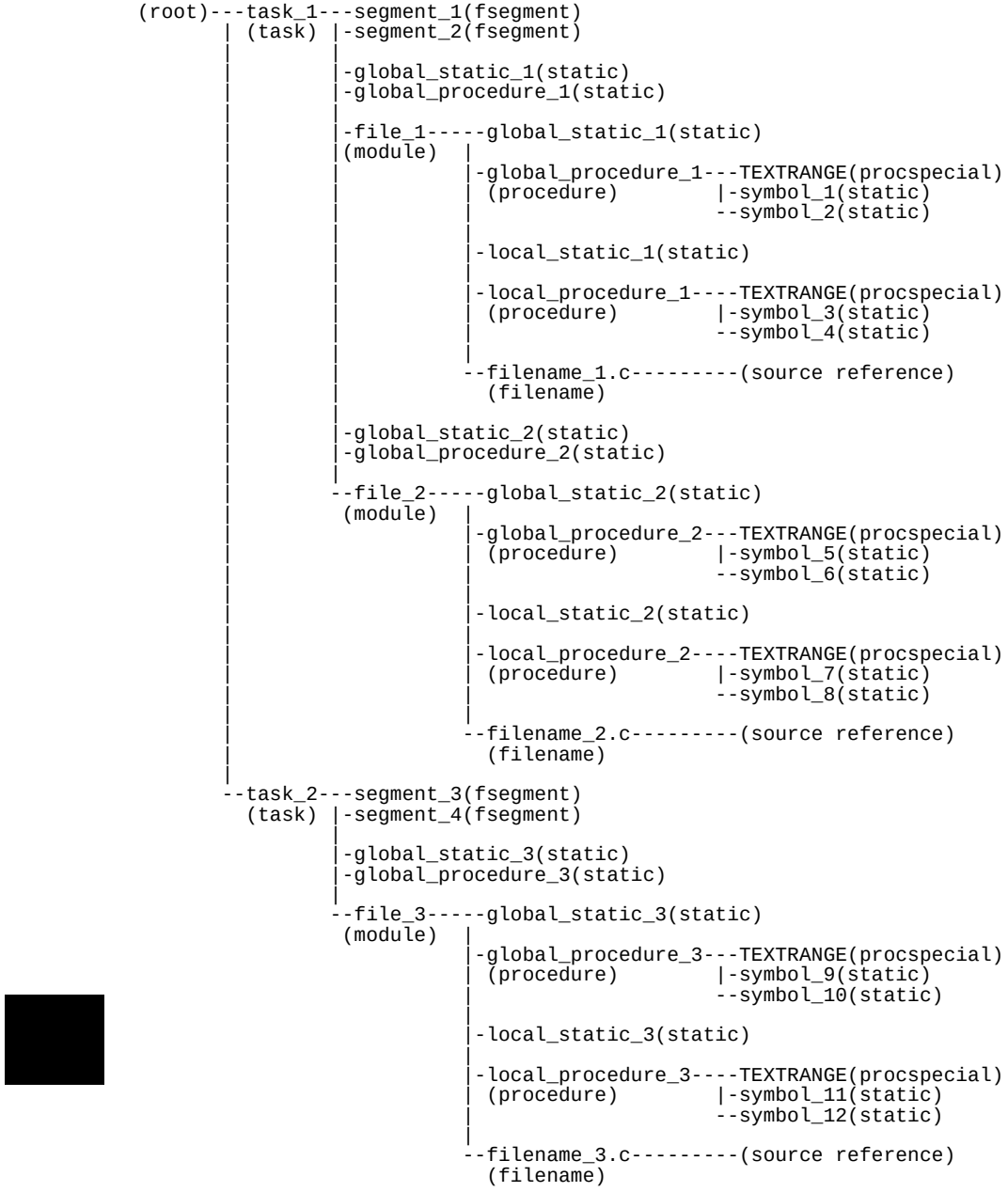
The most reliable results will be obtained when using syntax that tells the emulator whether the local symbols reside in the module symbol or in the source file symbol.

Note 

The following examples pertain to emulation with HP-OMF V33 files ONLY.

Note 

When you use the HP-OMF V33 file, you can not refer the symbol which does not belong any segments.


(root)---task_1---segment_1(fsegment)
(task) |-segment_2(fsegment)
|-global_static_1(static)
|-global_procedure_1(static)
|-file_1-----global_static_1(static)
(module)
|-global_procedure_1---TEXTRANGE(procspecial)
(procedure) |-symbol_1(static)
--symbol_2(static)
|-local_static_1(static)
|-local_procedure_1----TEXTRANGE(procspecial)
(procedure) |-symbol_3(static)
--symbol_4(static)
--filename_1.c---------(source reference)
(filename)
|-global_static_2(static)
|-global_procedure_2(static)
--file_2-----global_static_2(static)
(module)
|-global_procedure_2---TEXTRANGE(procspecial)
(procedure) |-symbol_5(static)
--symbol_6(static)
|-local_static_2(static)
|-local_procedure_2----TEXTRANGE(procspecial)
(procedure) |-symbol_7(static)
--symbol_8(static)
--filename_2.c---------(source reference)
(filename)
--task_2---segment_3(fsegment)
(task) |-segment_4(fsegment)
|-global_static_3(static)
|-global_procedure_3(static)
--file_3-----global_static_3(static)
(module)
|-global_procedure_3---TEXTRANGE(procspecial)
(procedure) |-symbol_9(static)
--symbol_10(static)
|-local_static_3(static)
|-local_procedure_3----TEXTRANGE(procspecial)
(procedure) |-symbol_11(static)
--symbol_12(static)
--filename_3.c---------(source reference)
(filename)

Displaying Symbols

If symbol information is present in the executable file, it is loaded along with the executable file (unless you use the "nosymbols" syntax). Both global symbols and symbols that are local to a program task can be displayed.

Global

To display global symbols, enter the following command.

```
display global_symbols <RETURN>
```

You will notice that only task symbols, "cmd_rds" and "setup", are displayed.

```
Global symbols in ex_cmd_rds
Task symbols
Task name ____________________________________________________________ ____
cmd_rds
setup









STATUS:   N70136--Running in monitor_____________________________________...R....
display global_symbols

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

Local

Task Symbols

Display global symbols will display the tasks in the executable. For example, if you issued the command from the root directory:

```
display local_symbols_in cmd_rds(task)
<RETURN>
```

or

```
display local_symbols_in cmd_rds <RETURN>
```

```
Symbols in cmd_rds(task)
Static symbols
Symbol name ____________________ Address range __ Segment _____________ Offset
Cmd_Input                  01:1009:0008            data                      0000
Err_Handler                01:805E:0057            lib                       0055
MM_CHECK_L                 01:1000:000A            libdata                   0000
MM_CHECK_X                 01:1000:000A            libdata                   0000
MONITOR_MESSAGE            01:1000:0002 - 0005     envdata                   0002
Msg_Dest                   01:1009:0009 - 0028     data                      0001
TOP_OF_STACK               01:1111:7F0A - 7F0B     userstack                 7EFE
USER_ENTRY                 01:800E:00FB            env                       00F9
USR_STACK                  01:1111:000C - 000D     userstack                 0000
Write_Msg                  01:8000:0000 - 0040     prog_cmd_rds              0000
XEnv_86_except             01:1000:0000 - 0001     envdata                   0000
_Cmd_Input                 01:1009:0008            data                      0000
_Msg_Dest                  01:1009:0009 - 0028     data                      0001
_Write_Msg                 01:8000:0000 - 0040     prog_cmd_rds              0000
__HEAP_PTR                 01:1000:0006 - 0009     envdata                   0006

STATUS:   cws:cmd_rds.cmd_rds_____________________________________________...R.... display
local_symbols_in cmd_rds(task)


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

This syntax used to access the children of the **task** cmd_rds.

Notice that the message of the first line on display announces the current working symbol is "cmd_rds(task)".

(See the *Symbolic Retrieval Utilty User's Guide* manual on SRU for information about current working symbols.)

Module Symbols

Since module symbols reside under task symbols in HP-OMF V33 hierarchy, module symbols are not considered global symbols. Module symbols are accessed as children of task symbols. Display global symbols will display the tasks in the executable. Modules can be shown by displaying the local symbols for a task. For example, if you issued the command from the directory cmd_rds(task):

```
display local_symbols_in cmd_rds <RETURN>
```

This syntax used to access the children of the **module** cmd_rds.

```
Symbols in cmd_rds(task).cmd_rds(module)
Procedure symbols
Procedure name _________________ Address range __ Segment _____________ Offset
Write_Msg                 01:8000:0000 - 0040     prog_cmd_rds              0000
main                      01:8000:0041 - 00E0     prog_cmd_rds              0041

Static symbols
Symbol name ____________________ Address range __ Segment _____________ Offset
Cmd_Input                 01:1009:0008            data                      0000
Msg_Dest                  01:1009:0009 - 0028     data                      0001
_Cmd_Input                01:1009:0008            data                      0000
_Msg_Dest                 01:1009:0009 - 0028     data                      0001
_Write_Msg                01:8000:0000            prog_cmd_rds              0000
_main                     01:8000:0041            prog_cmd_rds              0041

Filename symbols
Filename ____________________________________________________________________
/usr/hp64000/demo/emul/hp64756/ext_mode/cmd_rds.c

STATUS:   cws:cmd_rds.cmd_rds_____________________________________________...R....
display local_symbols_in  cmd_rds.cmd_rds


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

Notice that the current working symbol has changed to "cmd_rds(task).cmd_rds(module)" in the first line on display.

Non-Line-Number Symbols

To access non-line-number local symbols, in the file cmd_rds.c, use the following syntax:

```
display local_symbols_in
cmd_rds.cmd_rds<RETURN>
```

or

```
display local_symbols_in cmd_rds(module)
<RETURN>
```

This syntax is used to access the children of the ***module*** cmd_rds.

The second example will work if the **task** cmd_rds is the current working symbol.

```
Symbols in cmd_rds(task).cmd_rds(module)
Procedure symbols
Procedure name _________________ Address range __ Segment _____________ Offset
Write_Msg                01:8000:0000 - 0040     prog_cmd_rds              0000
main                     01:8000:0041 - 00E0     prog_cmd_rds              0041

Static symbols
Symbol name ____________________ Address range __ Segment _____________ Offset
Cmd_Input                01:1009:0008            data                      0000
Msg_Dest                 01:1009:0009 - 0028     data                      0001
_Cmd_Input               01:1009:0008            data                      0000
_Msg_Dest                01:1009:0009 - 0028     data                      0001
_Write_Msg               01:8000:0000            prog_cmd_rds              0000
_main                    01:8000:0041            prog_cmd_rds              0041

Filename symbols
Filename ____________________________________________________________________
/usr/hp64000/demo/emul/hp64756/ext_mode/cmd_rds.c

STATUS:   cws:cmd_rds.cmd_rds_____________________________________...R....
display local_symbols_in  cmd_rds.cmd_rds


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

Line-Number Symbols

Symbol accesses for line-number symbols require the file name in quotes. Use the following syntax to display local symbols which are line numbers for the file cmd_rds.c:

```
display local_symbols_in
cmd_rds(task).cmd_rds(module)."cmd_rds.c":
<RETURN>
```

or

```
display local_symbols_in
cmd_rds."cmd_rds.c": <RETURN>
```

The second example will work if the **task** cmd_rds is the current working symbol.

The quotes are used to specify the file name containing the line number symbols for the emulator.

```
Symbols in .../usr/hp64000/demo/emul/hp64756/ext_mode/cmd_rds.c":
Source reference symbols
Line range ____________________ Address range __ Segment _____________ Offset
#1-#5                    01:8000:0000 - 0009     prog_cmd_rds              0000
#6-#8                    01:8000:000A - 0013     prog_cmd_rds              000A
#9-#9                    01:8000:0014 - 0019     prog_cmd_rds              0014
#10-#11                  01:8000:001A - 0025     prog_cmd_rds              001A
#12-#12                  01:8000:0026 - 0029     prog_cmd_rds              0026
#13-#13                  01:8000:002A - 002D     prog_cmd_rds              002A
#14-#14                  01:8000:002E - 003C     prog_cmd_rds              002E
#15-#15                  01:8000:003D - 0040     prog_cmd_rds              003D
#16-#18                  01:8000:0041 - 004A     prog_cmd_rds              0041
#19-#24                  01:8000:004B            prog_cmd_rds              004B
#25-#26                  01:8000:004C - 0051     prog_cmd_rds              004C
#27-#27                  01:8000:0052 - 006A     prog_cmd_rds              0052
#28-#28                  01:8000:006B - 008D     prog_cmd_rds              006B
#29-#29                  01:8000:008E            prog_cmd_rds              008E
#30-#30                  01:8000:008F - 00A0     prog_cmd_rds              008F

STATUS:   cws:.../hp64000/demo/emul/hp64756/ext_mode/cmd_rds.c":_______...R....
display local_symbols_incmd_rds(task).cmd_rds(module)."cmd_rds.c":


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

It is not possible to display both types of local symbols (non-line-number and line-number symbols) with the same command.

The syntax used with HP format absolute files to display local symbols will work with HP-OMF V33 files if the **task** cmd_rds is the current working symbol. As in the above example, this will display only the line number symbols in file cmd_rds.c

```
display local_symbols_in cmd_rds.c: <RETURN>
```

Address Expression in Extended Mode

You can use the following address expression in emulation commands in extended mode of the emulator.

<HP-OMF V33 symbol>

You can use the symbol which is generated by the HP 64875 V33/53 Extended Mode Locator just same as the <TASK>::<OFFSET> address expression below.

<TASK>::<OFFSET>

This expression (TASK:0-0FF hex; SEGMENT:0-0FFFF hex; OFFSET:0-0FFFF hex) is the task, segment and offset portion of the logical address. Refer to the *HP 64875 NEC V33/53 Extended Mode Locator:User's Guide.*

fcode e <24-bit address>

This expression (0-0FFFFFF hex) with "fcode e" is a extended address in the 70136 address range.

fcode p <20-bit address>

This expression (0-0FFFFF hex) with "fcode p" is a physical address in the 70136 address range. In run or step commands, the emulation system converts this physical address to a :<OFFSET> address as specified by the "Select Algorithm for physical run addresses" configuration option in "Configuring the Emulator" chapter.

fcode none <20-bit address>

This expression (0-0FFFFF hex) with "fcode none" is a physical address in the 70136 address range. In run or step commands, the emulation system converts this physical address to a :<OFFSET> address as specified by the "Select Algorithm for physical run addresses" configuration option in "Configuring the Emulator" chapter.

:<OFFSET>

This expression (SEGMENT:0-0FFFF hex; OFFSET:0-0FFFF hex) is segment and offset portion of the logical address.

<I/O_ADDRESS>

This expression (0-0FFFF hex) is a 70136 I/O address. This expression should be used in I/O command

Display Memory

Use the following command to display memory for function main in file cmd_rds.c in task cmd_rds:

```
display memory cmd_rds.cmd_rds.main mnemonic
<RETURN>
set symbols on <RETURN>
```

This tells the emulator to look in task cmd_rds, module cmd_rds, and then for label main. You could also use the command when the current working symbol is "cmd_rds(task).cmd_rds(module)":

```
display memory main(procedure) mnemonic
<RETURN>
```

To include source lines in mnemonic memory displays, enter the following command:

```
set source on  <RETURN>
```

```
  Memory  :mnemonic :file = .../hp64000/demo/emul/hp64756/ext_mode/cmd_rds.c":
   address     label          data
      16
      17       main ()
      18       {
 01:8000:0041  cmd_rd._main  C8020000    PREPARE 0002,00
 01:8000:0045                1E          PUSH DS0
 01:8000:0046                B80910      MOV AW,#1009
 01:8000:0049                8ED8        MOV DS0,AW
      19            static char Msg_A[] = "Command A Entered              ";
      20            static char Msg_B[] = "Entered B Command              ";
      21            static char Msg_I[] = "Invalid Command                ";
      22            char c;
      23
      24            for (;;)
 01:8000:004B                90          NOP
      25            {
      26                 Cmd_Input = '\0';

STATUS:   N70136--Running in monitor_____________________________________...R....
set source on
```

Note

When you use the :<OFFSET> address expression in displaying memory command, you should enter the same value to enter address ranges.
When you use the <TASK>::<OFFSET> address expression in displaying memory command, you should enter the same <TASK> and value to enter address ranges.

Note

When you load the HP-OMF V33 format file, the symbols have <TASK>::<OFFSET> information. When you enter the following address expression in "display memory mnemonic" command, the symbols will be not displayed with mnemonic memory display because the address entered has no <TASK> information.

- fcode none <20-bit address>
- fcode p <20-bit address>
- :<OFFSET>

Note

When program execution should take place in real-time and the emulator should break to the monitor to read page registers (refer to "Configuring the Emulator" chapter), the commands showing above with the following address expressions which need physical to extended address conversion are not allowed in running user program.

- fcode none <20-bit address>
- fcode p <20-bit address>
- :<OFFSET>

If you entered, the following error message will be shown:

```
ERROR:  Restricted to real time runs
```

Using Software Breakpoints

Caution

If you change the relation between the physical address and the extended address (ex. change the value of page registers) after you set a software breakpoint with the following address expressions, the breakpoint interrupt instruction (F1 hex) is left in memory and the software break will not occur at the specified address.

- fcode none <20-bit address>
- fcode p <20-bit address>
- :<OFFSET>

Caution

Software breakpoints should not be set, cleared, enabled, or disabled 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.

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>
```

Setting a Software Breakpoint

To set a software breakpoint at the address of function main (source line 17) in file cmd_rds.c in task cmd_rds, enter the following command:

```
modify software_breakpoints set
cmd_rds.cmd_rds.main  <RETURN>
```

or

```
modify software_breakpoints set main
<RETURN>
```

The second example will work if the **task** cmd_rds is the current working symbol.

```
  Memory  :mnemonic :file = .../hp64000/demo/emul/hp64756/ext_mode/cmd_rds.c":
   address     label          data
      16
      17       main ()
      18       {
*01:8000:0041  cmd_rd._main  F1          illegal opcode, data = F1
 01:8000:0042                0200        ADD AL,[BW][IX]
 01:8000:0044                001EB809    ADD 09B8,BL
 01:8000:0048                108ED890    ADDC [BP-6F28],CL
      25            {
      26                Cmd_Input = '\0';
 01:8000:004C  main.Block_1  C606080000  MOV 0008,#00
 01:8000:0051                90          NOP
      27                while ((c = Cmd_Input) == '\0');
 01:8000:0052                EB05        BR SHORT pr|main.Block_1+00000D
 01:8000:0054                90          NOP
 01:8000:0055                90          NOP
 01:8000:0056                90          NOP

STATUS:   N70136--Running in monitor_____________________________________...R....
 modify  software_breakpoints  set cmd_rds.cmd_rds.main


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

Notice that an asterisk (*) appears next to the breakpoint address. The asterisk shows that a software breakpoint is pending at that address.

To use source line numbers in setting software breakpoints, you should change the current working symbol to the line numbers for the source file before setting software breakpoints.

For example, if you want to set a software breakpoint at the address of source line 17 in file cmd_rds.c in task cmd_rds, you should change the current working symbol to cmd_rds(task).cmd_rds(module)."cmd_rds.c" before setting software breakpoints.

Enter the following command:

```
cws cmd_rds.cmd_rds."cmd_rds.c": <REUTRN>
```

To set the software breakpoint, enter the following command:

```
modify software_breakpoints set line 17
<RETURN>
```

Running the Program

The "run" command causes the emulator to execute the user program. 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.

You can use the following address expression in the "run from" command in extended mode.

<HP-OMF V33 symbol>

You can use the symbol which is generated by the HP 64875 V33/53 Extended Mode Locator just same as the <TASK>::<OFFSET> address expression below.

<TASK>::<OFFSET>

This expression (TASK:0-0FF hex; SEGMENT:0-0FFFF hex; OFFSET:0-0FFFF hex) is task, segment and offset portion of the logical address. Refer to the *HP 64875 NEC V33/53 Extended Mode Locator:User's Guide.*

Note

When you enter <HP-OMF V33 symbol> or <TASK>::<OFFSET> address expression in the "run from" command, the <TASK> information entered is ignored by the emulation system.

When you enter the "run from" command in "running in monitor" status, the <TASK> which is active just before entering the monitor is used.
In "running user program" status, the current working <TASK> is used.

fcode p <20-bit address>

This expression (0-0FFFFF hex) with "fcode p" is a physical address in the 70136 address range. The emulation system converts this address to a :<OFFSET> address as specified by the "Select Algorithm for physical run addresses" configuration option in "Configuring the Emulator" chapter.

fcode none <20-bit address>

This expression (0-0FFFFF hex) with "fcode none" is a physical address in the 70136 address range. The emulation system converts this address to a :<OFFSET> address as specified by the "Select Algorithm for physical run addresses" configuration option in "Configuring the Emulator" chapter.

:<OFFSET>

This expression (SEGMENT:0-0FFFF hex; OFFSET:0-0FFFF hex) is the segment and offset portion of the logical address.

From Transfer Address

The "run from transfer_address" command specifies that the emulator start executing at a previously defined "start address". Transfer addresses are defined in assembly language source files with the END assembler directive (i.e., pseudo instruction) in the normal mode task. Enter:

```
run from transfer_address <RETURN>
```

```
  Memory  :@e :mnemonic :file = .../demo/emul/hp64756/ext_mode/cmd_rds.c":
   address     label          data
      16
      17       main ()
      18       {
>01:8000:0041  cmd_rd._main  C8020000    PREPARE 0002,00
 01:8000:0045                1E          PUSH DS0
 01:8000:0046                B80910      MOV AW,#1009
 01:8000:0049                8ED8        MOV DS0,AW
      19            static char Msg_A[] = "Command A Entered               ";
      20            static char Msg_B[] = "Entered B Command               ";
      21            static char Msg_I[] = "Invalid Command                 ";
      22            char c;
      23
      24            for (;;)
 01:8000:004B                90          NOP
      25            {
      26                 Cmd_Input = '\0';

STATUS:   N70136--Running in monitor      Software break:0180041@e.....R....
run from transfer_address


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

Notice the highlighted bar on the screen; it shows the current program counter.

Notice also that the asterisk is no longer next to the breakpoint address; this shows that the breakpoint has been hit and is no longer active.

In run or step commands, the emulation system converts this physical address to a :<OFFSET> address as specified by the "Select Algorithm for physical run addresses" configuration option in "Configuring the Emulator" chapter.

Stepping Through the Program

The step command allows you to step through program execution an instruction or a number of instructions at a time. You can step though the instructions associated with high-level program source lines. 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 source <RETURN>
```

Notice that the highlighted bar (the current program counter) moves to the instructions associated with the next source line.

Enter the "step source" command again by pressing:

```
<RETURN>,  <RETURN>
```

Notice that the emulator continues to step through the program and that the message "assembly steps taken: XXX" appears on the status line. This happens because the "while" test remains true, and the emulator never completes the execution of the assembly instructions associated with that source line. To stop the "step source" command, enter:

```
<CTRL>-c
```

Continue user program execution with the "run" command.

```
run <RETURN>
```

Modifying Memory

The sample program is a simple command interpreter. Commands are sent to the sample program through a "char" sized memory location, global variable **Cmd_Input**. You can use the modify memory feature to send a command to the sample program.

Use the following command to modify memory for static symbol **Cmd_Input** in file cmd_rds.c in task cmd_rds:

```
modify memory cmd_rds.cmd_rds.Cmd_Input
strings to 'A'  <RETURN>
```

or

```
modify memory Cmd_Input string to 'A'
<RETURN>
```

The second example will work if the current working symbol is "cmd_rds(task).cmd_rds(module)". This tells the emulator to look in task cmd_rds, module cmd_rds, and then for label Cmd_Input.

To verify that the program correctly copied the message "Command A Entered" to the **Msg_Dest** array, display the contents of the array with the following command:

```
display data Msg_Dest thru +1fh char
<RETURN>
```

Enter the following commands to verify that the program works for the other possible command inputs.

```
modify memory Cmd_Input string to 'B'
<RETURN>
modify memory Cmd_Input string to 'C'
<RETURN>
```

Notice that the display is updated when the memory contents change due (indirectly) to the "modify memory" command.

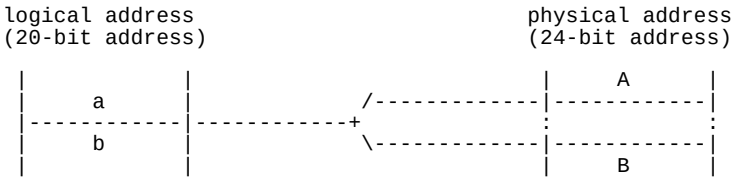
```
 Data  :update
   address     label         type        data
 01:1009:0009  cm._Msg_Dest  char[]   Command A Entered









STATUS:   N70136--Running user program_____________________________...R....
display data Msg_Dest thru +1fh char

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

Note

When you use the :<OFFSET> address expression in modifying memory command, you should enter the same value to enter address ranges.

When you use the <TASK>::<OFFSET> address expression in modifying memory command, you should enter the same <TASK> and value to enter address ranges.

When you use the <TASK>::<OFFSET> address expression in modifying memory command, you can not enter the address ranges which crosses the page boundary that is not physically continuous are refused.
<TASK>::<OFFSET> range must be within the area a or b for "modify memory" command.



Note

When program execution should take place in real-time and the emulator should break to the monitor to read page registers (refer to "Configuring the Emulator" chapter), the commands showing above with the following address expressions which need physical to extended address conversion are not allowed in running user program.

- fcode none <20-bit address>
- fcode p <20-bit address>
- :<OFFSET>

If you entered, the following error message will be shown:

```
ERROR:  Restricted to real time runs
```

Breaking into the Monitor

The "break" command causes emulator execution to break 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>
```

Displaying Registers

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

```
display registers <RETURN>
```

Note

You should not change the value of 70136 and 70236 page registers with using "modify io_port" command. You should use the "modify registers" command to change the value of page registers.

Refer to "Register Names and Classes" section in chapter 5.

```
Registe

Next PS:PC 18005C@e
    PC 005C    SP 7EF6    IX 0000    IY 0049    BP 7EFA
    PS 8000    SS 1111   DS0 1009   DS1 1009      [rrrrvdibszfafpic]
    AW 1000    BW 0000    CW 0000    DW 1009   PSW 1111001001000110









STATUS:   N70136--Running in monitor_____________________________________...R....
display registers

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

Stepping Through the Program

You can step through sample program instructions while displaying registers. For example, entering several step commands will give you a display similar to the following.

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

```
Registers

Next PS:PC 18005D@e
    PC 005D    SP 7EF6    IX 0000    IY 0049    BP 7EFA
    PS 8000    SS 1111   DS0 1009   DS1 1009      [rrrrvdibszfafpic]
    AW 1000    BW 0000    CW 0000    DW 1009   PSW 1111001001000110

Step_PC 18005D@e  NOP
Next PS:PC 18005E@e
    PC 005E    SP 7EF6    IX 0000    IY 0049    BP 7EFA
    PS 8000    SS 1111   DS0 1009   DS1 1009      [rrrrvdibszfafpic]
    AW 1000    BW 0000    CW 0000    DW 1009   PSW 1111001001000110

Step_PC 18005E@e  NOP
Next PS:PC 18005F@e
    PC 005F    SP 7EF6    IX 0000    IY 0049    BP 7EFA
    PS 8000    SS 1111   DS0 1009   DS1 1009      [rrrrvdibszfafpic]
    AW 1000    BW 0000    CW 0000    DW 1009   PSW 1111001001000110

STATUS:   N70136--Stepping complete_____________________________________...R....
step


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

You can use the following address expression in the "step from" command in extended mode.

<HP-OMF V33 symbol>

You can use the symbol which is generated by the HP 64875 V33/53 Extended Mode Locator just same as the <TASK>::<OFFSET> address expression below.

<TASK>::<OFFSET>

This expression (TASK:0-0FF hex; SEGMENT:0-0FFFF hex; OFFSET:0-0FFFF hex) is task, segment and offset portion of the logical address. Refer to the *HP 64875 NEC V33/53 Extended Mode Locator:User's Guide.*

Note 

When you enter <HP-OMF V33 symbol> or <TASK>::<OFFSET> address expression in the "step from" command, the <TASK> information entered is ignored by the emulation system.

When you enter the "step from" command in "running in monitor" status, the <TASK> which is active just before entering the monitor is used.
In "running user program" status, the current working <TASK> is used.

---

fcode p <20-bit address>

This expression (0-0FFFFF hex) with "fcode p" is a physical address in the 70136 address range. The emulation system converts this address to a :<OFFSET> address as specified by the "Select Algorithm for physical run addresses" configuration option in "Configuring the Emulator" chapter.

fcode none <20-bit address>

This expression (0-0FFFFF hex) with "fcode none" is a physical address in the 70136 address range. The emulation system converts this address to a :<OFFSET> address as specified by the "Select Algorithm for physical run addresses" configuration option in "Configuring the Emulator" chapter.

:<OFFSET>

This expression (SEGMENT:0-0FFFF hex; OFFSET:0-0FFFF hex) is the segment and offset portion of the logical address.

Note

There are a few cases in which the emulator can not step.
Step command is not accepted between each of the following instructions and the next instruction.

1) Manipulation instructions for sreg:
MOV sreg,reg16; MOV sreg,mem16; POP sreg.

2) Prefix instructions:
PS:, SS:, DS0:, DS1:,
REPC, REPNC, REP, REPE, REPZ, REPNE, REPNZ.

3) EI, RETI, DI, BUSLOCK.

Continue user program execution with the "run" command.

```
run <RETURN>
```

Using the Analyzer

HP 64700 emulators contain an emulation analyzer. The emulation analyzer monitors the internal emulation lines (address, data, and status). Optionally, you may have an additional 16 trace signals which monitor external input lines. 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 look at the execution of the sample program after the address of the first instruction in the **Write_Msg** function in file cmd_rds.c in task cmd_rds. To trigger on this address, enter:

```
trace about address
cmd_rds.cmd_rds.Write_Msg  <RETURN>
```

or

```
trace about address Write_Msg  <RETURN>
```

The second example will work if the current working symbol is "cmd_rds(task).cmd_rds(module)".

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

```
modify memory Cmd_Input string to 'A'
<RETURN>
```

The status line now shows "Emulation trace complete".

You can use the following address expression in the "trace" command in extended mode.

<HP-OMF V33 symbol>

You can use the symbol which is generated by the HP 64875 V33/53 Extended Mode Locator just same as the <TASK>::<OFFSET> address expression below.

<TASK>::<OFFSET>

This expression (TASK:0-0FF hex; SEGMENT:0-0FFFF hex; OFFSET:0-0FFFF hex) is task, segment and offset portion of the logical address. Refer to the *HP 64875 NEC V33/53 Extended Mode Locator:User's Guide.*

<24-bit address>

This expression (0-0FFFFFF hex) is a extended address in the 70136 address range.

Displaying the Trace

To display the trace, enter:

```
display trace <RETURN>
```

```
Trace List                  Offset=0
Label:      Address       Data    Opcode or Status w/ Source Lines   time count
Base:       symbols       hex           mnemonic w/symbols           relative
-007   B.Block_1+000009     8000  PUSH    DW                          80.    nS
-006   B.Block_1+000010     C483    C483  prefetch                    120    nS
-005   u|cmd_rds+007EE8     1009    1009  memory write                160    nS
-004   B.Block_1+00000A     1009  PUSH    AW                          80.    nS
-003   u|cmd_rds+007EE6     0029    0029  memory write                240    nS
-002   B.Block_1+00000B     0029  CALL    FAR PTR 80000               80.    nS
-001   B.Block_1+000012     EB04    EB04  prefetch                    40.    nS
about  cmd_r._Write_Msg     04C8    04C8  prefetch                    200    nS
+001   Write_Msg+000002     0000    0000  prefetch                    240    nS
+002   u|cmd_rds+007EE4     8000    8000  memory write                280    nS
+003   u|cmd_rds+007EE2     009E    009E  memory write                240    nS
      ##########.../emul/hp64756/ext_mode/cmd_rds.c - line     1 thru     5 ####
      volatile char Cmd_Input;
      char Msg_Dest[0x20];


STATUS:   N70136--Running user program    Emulation trace complete______...R....
display trace


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

Line 0 (labeled "about") in the trace list above shows the state which triggered the analyzer. The trigger state is always on line 0.

If there is data that does not appear on the screen, you can use <CTRL> **f** and <CTRL> **g** to roll the display left and right. The trace labels, shown on the second line of the display, are described earlier in this section.

To display the remaining lines of the trace, press the <PGDN> or <NEXT> key.

Storing Memory Contents to an Absolute File

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

```
store memory cmd_rds.cmd_rds.main thru +0ffh
to <absfile> <RETURN>
```

The command above causes the contents of the memory range from function main in file cmd_rds.c in task cmd_rds (100 hex) to be stored in the absolute file "absfile.X". Notice that the ".X" extension is appended to the specified filename.

Note

When you reload the absolute file made by "store memory" command with following address expressions, you should set up the same value to page registers
(PGR 1 - PGR 64) that you enter the "store memory" command. Otherwise, the memory image is not same as when you enter the "store memory" command.

You also should add the "fcode p" option to reload the absolute file.

- fcode none <20-bit address>
- fcode p <20-bit address>
- :<OFFSET>

Simulated I/O Configuration in the Extended Mode

When you use the simulated I/O feature in the extended mode, care should be taken to answer the simulated I/O configuration questions in "modify configuration" command.

When you set the simulated I/O Control Address, you should add the <TASK> information.

For example, The symbol "SIMIO_CA_ONE" is the default symbol associated with the first simulated I/O Control Address.

```
Simio control address 1? SIMIO_CA_ONE
```

If the symbol "SIMIO_CA_ONE" belongs the task <task1>, you should modify the simulated I/O Control Address as follows.

```
Simio control address 1? task1.SIMIO_CA_ONE
```

Refer to the *Simulated I/O* reference manual.

Notes

Index