HP 64752

70732 Emulator Softkey Interface

User's Guide

HEWLETT PACKARD

HP Part No. 64752-97003
Printed in U.S.A.
July 1994

Edition 2

Notice

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

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

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

Printing History

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

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

Edition 1	64752-97001, August 1993
Edition 2	**64752-97003, July 1994**

Using this Manual

This manual shows you how to use the following emulators with the Softkey Interface.

- HP 64752A 70732 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 demo board/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 70732 Emulator.** This chapter briefly introduces you to the concept of emulation and lists the basic features of the 70732 emulator.

Chapter 2 **Getting Started.** This chapter shows you how to use emulation commands by executing them on a sample program. This chapter describes the sample program and how to: load programs into the emulator, map memory,display and modify memory, display registers, step through program, 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 demo board/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, 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.** This appendix describes the advantages and disadvantages of foreground and background monitors and how to use foreground monitor.

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

Conventions

Example commands throughout the manual use the following conventions:

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

Notes

Contents

Illustrations

Notes

1

Introduction to the 70732 Emulator

Introduction

The topics in this chapter include:

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

Purpose of the Emulator

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

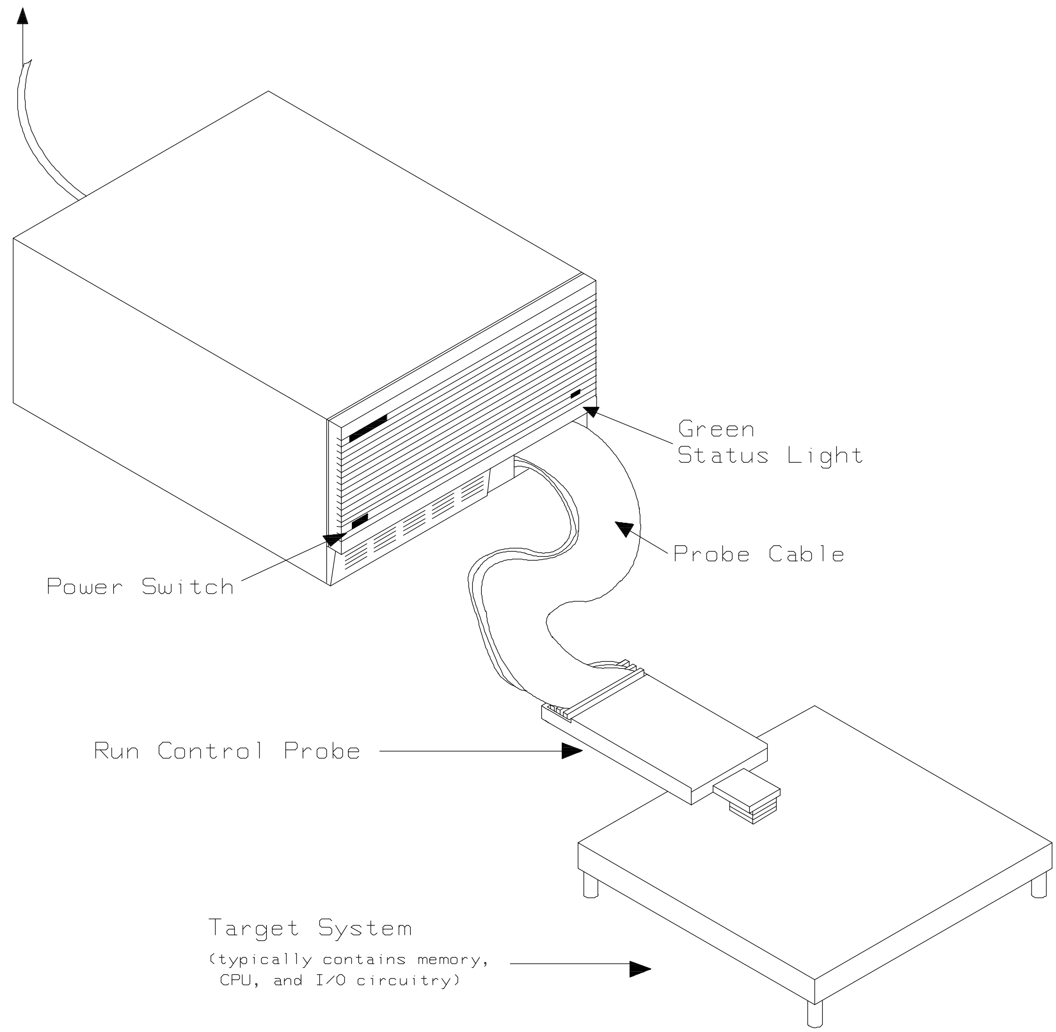


Figure 1-1 HP 64752A Emulator for uPD70732

Features of the 70732 Emulator

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

Supported Microprocessors

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

Clock Speeds

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

Emulation memory

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

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

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

Analysis

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

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

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

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

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

Registers

You can display or modify the 70732 internal register contents.

Single-Step

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

Breakpoints

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

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

Reset Support

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

Configurable Target System Interface

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

Foreground or Background Emulation Monitor

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

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

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

Real-Time Operation

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

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

Coverage

The 70732 emulator does not support coverage test.

Easy Products Upgrades

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

Limitations, Restrictions

Reset While in Background Monitor

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

User Interrupts While in Background Monitor

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

Interrupts While Executing Step Command

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

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

Note

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

Evaluation Chip

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

2

Getting Started

Introduction

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

Before You Begin

Prerequisites

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

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

2. Installed the Softkey Interface software on your computer. Refer to the *HP 64700 Series Installation/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 70732 emulator.

A Look at the Demo Program

The demo program is *spmt_demo* consisting of source program *spmt_demo.c* and *init_spmt.s*.

Where is the spmt_demo Software?

The demo program is shipped with the Softkey Interface and may be copied from the following directory.

/usr/hp64000/demo/emul/hp64752

Compiling the Demo Program

The demo program is written for and compiled/linked with the NEC Corporation CC732 C Compiler Package. The demo program was compiled with the following command.

```
$ cc732 -c -g spmt_demo.c <RETURN>
```

Linking the Demo Program

The following command was used to generate the absolute file. The "spmt_demo.d" linker command file is shown in figure 2-2.

```
$ ld732 -D spmt_demo.d -o spmt_demo.abs
spmt_demo.o <RETURN>
```

```
TEXT1:  !LOAD ?RX V0x0 {
        .text = $PROGBITS ?AX .text;
};

DATA:   !LOAD ?RW V0x20000 {
        .data  = $PROGBITS      ?AW;
        .sdata = $PROGBITS      ?AWG;
        .sbss  = $NOBITS        ?AWG;
        .bss  = $NOBITS         ?AW;
};
TEXT2:  !LOAD ?RX V0xfffffff0 {
        Reset_Entry = $PROGBITS ?AX Reset_Entry;
};

__tp_TEXT @ %TP_SYMBOL {TEXT1};
__gp_DATA @ %GP_SYMBOL &__tp_TEXT {DATA};
```

Figure 2-1 Linker Command File

Generate HP Absolute file

To generate HP absolute file for the Softkey Interface, you need to use "**v810cnv**" absolute file format converter. The **v810cnv** converter is provided with the Softkey Interface. To generate HP absolute file, enter the following command:

```
$ v810cnv spmt_demo <RETURN>
```

You will see that spmt_demo.X, spmt_demo.L, and spmt_demo.A are generated. The file *spmt_demo.X* contains the absolute code of the program. The file *spmt_demo.L* contains the list of global symbols. The files *spmt_demo.A* contains the list of local symbols for the respective files.

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.

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 70732 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 70732 emulator, enter:

make_sys emv810 <RETURN>

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

add <module_number> ***naming_it*** n70732 <RETURN>

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

end <RETURN>

If the measurement system and emulation module are named "emv810" and "n70732" as shown above, you can enter the emulation session with the following command:

emv810 default n70732 <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.

```
                  HPB3069-11001 A.05.10 16Mar93
                        N70732 SOFTKEY USER INTERFACE

                      A Hewlett-Packard Software Product
                      Copyright Hewlett-Packard Co. 1992

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

                           RESTRICTED RIGHTS LEGEND

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


STATUS:   Starting new session__________________________________________...R....



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

Figure 2-2 Softkey Interface Display

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.net).

```
#---------+------------+-----------+------------------------------------------
# Channel |  Logical   | Processor | Remainder of Information for the Channel
#  Type   |   Name     |   Type    | (IP address for LAN connections)
#---------+------------+-----------+------------------------------------------
   lan:      v810         n70732      21.17.9.143
```

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 option (question), refer to the "*Configuring the Emulator*" chapter.

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

```
modify configuration <RETURN>
```

Answer to the series of questions as below.

```
Enter monitor after configuration?  yes <RETURN>
Restrict to real-time runs?  no <RETURN>
Processor data bus width?  32 <RETURN>
Enable the instruction cache?  no <RETURN>
Modify memory configuration?  yes <RETURN>
Monitor type?  background <RETURN>
Value for address during background operation?  00000000H
```

Now you should be facing memory mapping screen. Three mapper terms must be specified for the demo program. Enter the following lines to map the program code area as emulation ROM, data area as emulation RAM.

```
    0h thru 1fffh emulation  rom <RETURN>
    20000h thru 20fffh emulation  ram <RETURN>
    0fffff000h thru 0ffffffffh 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 interactive measurement specification?  no <RETURN>
```

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

```
Configuration file name?  spmt_demo <RETURN>
```

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 command>          fork a shell and execute a shell command

pwd                       print the working directory
cd <directory>            change the working directory

pws                       print the default symbol scope
cws <SYMB>                change the working symbol - the working symbol also
                          gets updated when displaying local symbols and
                          displaying memory mnemonic


forward <UI> "command"    send the command in the quoted string from this user
                          interface to another one.  Replace <UI> with the name
                          of the other user interface as shown on the softkeys:

-More--(15%)
```

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

STATUS:   N70732--Emulation reset_______________________________________...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. If you wish to load only that portion of the absolute file that resides in memory mapped as emulation RAM or ROM, use the "load emul_mem" syntax. If you wish to load only the portion of the absolute file that resides in memory mapped as target RAM, use the "load user_mem" syntax. If you want both emulation and target memory to be loaded, do not specify "emul_mem" or "user_mem". For example:

load `spmt_demo <RETURN>`

Note

When loading a program if the status line shows

```
"ERROR:      No absolute file, No database:
spmt_demo
```

, you may NOT be in the directory that your program is in. To find out what directory you are in, enter:

```
! pwd <RETURN>
```

The `"!"` allows you to use an HP-UX shell command. To move into the correct directory, enter:

```
cd <directory path>  <RETURN>
```

You can also specify the pathname where your program resides. For example, you could enter:

load
`/usr/hp64000/demo/emul/hp64752/spmt_demo`
`<RETURN>`

Displaying Symbols

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

Global

To display global symbols, enter the following command.

display global_symbols <RETURN>

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 spmt_demo.X
Procedure symbols
Procedure name _________________ Address range __ Segment _____________ Offset
apply_controlle              0000073C - 000007F3                          071C
apply_productio              000004EC - 000005A7                          04CC
calculate_answe              000007F4 - 0000088F                          07D4
clear_buffer                 000002FC - 00000367                          02DC
endcommand                   000009FC - 00000A1B                          09DC
format_result                000005A8 - 00000613                          0588
get_next_token               000006A8 - 0000073B                          0688
initialze                    00000614 - 000006A7                          05F4
input_line                   00000020 - 00000073                          0000
lookup_token                 00000368 - 000003F3                          0348
main                         00000A1C - 00000A97                          09FC
math_library                 00000204 - 0000029F                          01E4
move_byte                    00000074 - 000000C3                          0054
outputline                   000002A0 - 000002FB                          0280
parse_command                000008FC - 00000987                          08DC

STATUS:   N70732--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 source file in which the symbols are defined. For example,

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

As you can see, the procedure symbols and static symbols in "spmt_demo.c" are displayed.
To list the next symbols, press the <PGDN> or <Next> key. the source reference symbols in "spmt_demo.c" will be displayed.

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.

```
Symbols in spmt_demo.c:
Procedure symbols
Procedure name _________________ Address range __ Segment _____________ Offset
apply_controlle               0000073C - 000007F3                          071C
apply_productio               000004EC - 000005A7                          04CC
calculate_answe               000007F4 - 0000088F                          07D4
clear_buffer                  000002FC - 00000367                          02DC
endcommand                    000009FC - 00000A1B                          09DC
format_result                 000005A8 - 00000613                          0588
get_next_token                000006A8 - 0000073B                          0688
initialze                     00000614 - 000006A7                          05F4
input_line                    00000020 - 00000073                          0000
lookup_token                  00000368 - 000003F3                          0348
main                          00000A1C - 00000A97                          09FC
math_library                  00000204 - 0000029F                          01E4
move_byte                     00000074 - 000000C3                          0054
outputline                    000002A0 - 000002FB                          0280
parse_command                 000008FC - 00000987                          08DC

STATUS:   cws: spmt_demo.c:_____________________________________________...R....
display local_symbols_in spmt_demo.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 `spmt_demo.c:` `<RETURN>`

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

```
Symbols in spmt_demo.c:
Source reference symbols
Line range ______________________ Address range __ Segment _____________ Offset
#1-#35                        00000020 - 0000002B                          0000
#36-#37                       0000002C - 00000037                          000C
#38-#39                       00000038 - 0000003B                          0018
#40-#40                       0000003C - 0000003F                          001C
#41-#41                       00000040 - 00000045                          0020
#42-#42                       00000046 - 00000049                          0026
#43-#43                       0000004A - 0000004F                          002A
#44-#44                       00000050 - 00000053                          0030
#45-#45                       00000054 - 0000006F                          0034
#46-#46                       00000070 - 00000073                          0050
#47-#49                       00000074 - 0000007F                          0054
#50-#51                       00000080 - 0000008B                          0060
#52-#53                       0000008C - 0000008F                          006C
#54-#54                       00000090 - 00000095                          0070
#55-#55                       00000096 - 00000099                          0076

STATUS:   N70732--Running in monitor____________________________________...R....
display local_symbols_in spmt_demo.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 demo program,

display memory main ***mnemonic*** <RETURN>

```
 Memory  :mnemonic :file = spmt_demo.c:
   address   data
  00000A1C  20A01800    MOVEA   0x0018,R0,R1
  00000A20  6108        SUB     R1,R3
  00000A22  E3DF1400    ST.W    R31,0x0014[R3]
  00000A26  6E8A        JBR     0x00000A94
  00000A28  4141        MOV     0x01,R10
  00000A2A  43DD1000    ST.W    R10,0x0010[R3]
  00000A2E  04DC2803    ST.W    R0,0x0328[R4]
  00000A32  508A        JBR     0x00000A82
  00000A34  FFAF5CFE    JAL     0x00000890
  00000A38  FFAFC4FE    JAL     0x000008FC
  00000A3C  44CD2803    LD.W    0x0328[R4],R10
  00000A40  2540        MOV     0x05,R1
  00000A42  4125        DIV     R1,R10
  00000A44  5E01        MOV     R30,R10
  00000A46  6AA10100    MOVEA   0x0001,R10,R11
  00000A4A  64DD0000    ST.W    R11,0x0000[R4]

STATUS:   N70732--Running in monitor____________________________________...R....
display memory main mnemonic


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

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.

Display Memory with Symbols

If you want to see symbol information with displaying memory in mnemonic format, the emulator Softkey Interface provides "set symbols" command. To see symbol information, enter the following command.

set symbols on <RETURN>

```
 Memory  :mnemonic :file = spmt_demo.c:   address  label          data  00000A1C
spmt_de:main  20A01800    MOVEA   0x0018,R0,R1
  00000A20                6108        SUB     R1,R3
  00000A22                E3DF1400    ST.W    R31,0x0014[R3]
  00000A26                6E8A        JBR     :main+00078
  00000A28                4141        MOV     0x01,R10
  00000A2A                43DD1000    ST.W    R10,0x0010[R3]
  00000A2E                04DC2803    ST.W    R0,0x0328[R4]
  00000A32                508A        JBR     :main+00066
  00000A34                FFAF5CFE    JAL     :request_command
  00000A38                FFAFC4FE    JAL     sp:parse_command
  00000A3C                44CD2803    LD.W    0x0328[R4],R10
  00000A40                2540        MOV     0x05,R1
  00000A42                4125        DIV     R1,R10
  00000A44                5E01        MOV     R30,R10
  00000A46                6AA10100    MOVEA   0x0001,R10,R11
  00000A4A                64DD0000    ST.W    R11,0x0000[R4]

STATUS:   N70732--Running in monitor____________________________________...R....
set symbols on


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

As you can see, the memory display shows symbol information.

Display Memory with Source Code

If you want to reference the source line information with displaying memory in mnemonic format, the emulator Softkey Interface provides "set source" command. To reference the source line information in inverse video, enter the following command:

set source on inverse_video on <RETURN>

```
 Memory  :mnemonic :file = spmt_demo.c:
   address  label          data
     371
     372    /******************* main program *******************/
     373
     374    main()
     375    {
  00000A1C  spmt_de:main  20A01800    MOVEA   0x0018,R0,R1
  00000A20                6108        SUB     R1,R3
  00000A22                E3DF1400    ST.W    R31,0x0014[R3]
  00000A26                6E8A        JBR     :main+00078
     376            int dummyv;
     377            dummyv = 1;
  00000A28                4141        MOV     0x01,R10
  00000A2A                43DD1000    ST.W    R10,0x0010[R3]
     378            tasknumber = 0;
  00000A2E                04DC2803    ST.W    R0,0x0328[R4]
  00000A32                508A        JBR     :main+00066

STATUS:   N70732--Running in monitor____________________________________...R....
set source on inverse_video on


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

To see the memory without source line referencing, enter the following command:

set source off <RETURN>

Running the Program

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

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>`

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).

Displaying Memory

The demo program "spmt_demo.c" alters memory.

Using Symbolic Addresses

In the following display, the memory range is displayed using symbolic addresses **data**.

The memory display window is periodically updated. For example, enter the following command:

display memory `data` ***thru*** `+7fh` ***blocked bytes*** `<RETURN>`

This command string is used to specify the range of memory from **data** to **data+7fh**.

```
 Memory  :bytes :access=bytes :blocked :repetitively
   address       data       :hex                                    :ascii
  0002031C-23    01   00   00   00   00   00   00   00       . . . .  . . . .
  00020324-2B    01   00   00   00   55   00   00   00       . . . .  U . . .
  0002032C-33    FF   FF   F6   DF   FF   FF   E8   FF       . . . .  . . . .
  00020334-3B    FB   FF   F8   DF   F7   FF   F4   FF       . . . .  . . . .
  0002033C-43    62   F7   70   FF   E9   FF   7F   FF       b . p .  . . . .
  00020344-4B    FA   FB   23   BF   82   FE   00   FF       . . # .  . . . .
  0002034C-53    3D   FF   00   7F   F0   F7   17   EF       = . . .  . . . .
  00020354-5B    16   FF   85   DF   B4   F7   80   6F       . . . .  . . . o
  0002035C-63    DD   FD   91   BF   F3   FF   2B   7F       . . . .  . . + .
  00020364-6B    DA   FE   62   6F   B5   F3   04   8F       . . b o  . . . .
  0002036C-73    50   7D   02   BF   96   FF   06   9F       P } . .  . . . .
  00020374-7B    67   FF   64   7E   12   F2   00   7F       g . d ~  . . . .
  0002037C-83    BD   FF   90   9F   FF   FF   F0   7E       . . . .  . . . ~
  00020384-8B    FF   FF   F2   7F   FF   FB   F4   DF       . . . .  . . . .
  0002038C-93    FF   FF   F0   CF   FF   FF   F4   7F       . . . .  . . . .
  00020394-9B    FF   FF   F0   EF   FF   FF   F8   DF       . . . .  . . . .

STATUS:   N70732--Running user program___________________________________...R....
display memory data thru data+7fh blocked bytes


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

Modifying Memory

You can use the modify memory command to send commands to the sample program. Memory locations **stackarea** and **stackarea+10h** correspond to memory address 20004 hex and 20014 hex respectivity. For example, to enter the ’10h’ at address 20004 and enter ’A’ at address 20014 : use the following commands.

display memory stackarea <RETURN>
modify memory stackarea ***to*** 10h <RETURN>
modify memory stackarea+10h ***string to*** ’A’ <RETURN>

After the memory location are modified, the memory display shows the following

```
 Memory  :bytes :access=bytes :blocked :update
   address       data       :hex                                   :ascii
  00020004-0B    10   FF   FF   FF   FF   FF   FF   FF      . . . .  . . . .
  0002000C-13    FF   FF   FF   FF   FF   FF   FF   FF      . . . .  . . . .
  00020014-1B    41   FF   FF   FF   FF   FF   FF   FF      A . . .  . . . .
  0002001C-23    FF   FF   FE   FF   FF   F7   FF   FF      . . . .  . . . .
  00020024-2B    FF   FE   FF   FF   FF   FF   FF   FF      . . . .  . . . .
  0002002C-33    FF   FF   FF   FF   F7   FF   FF   FF      . . . .  . . . .
  00020034-3B    FF   FF   FF   FF   FF   FF   FF   FF      . . . .  . . . .
  0002003C-43    FF   FF   FF   FF   FF   FF   F8   FF      . . . .  . . . .
  00020044-4B    FF   FF   F0   FF   FF   FF   F3   FF      . . . .  . . . .
  0002004C-53    FF   FF   FF   FF   FF   FF   FC   FF      . . . .  . . . .
  00020054-5B    FF   FF   FD   FF   FF   FF   F2   FF      . . . .  . . . .
  0002005C-63    FF   FF   F1   FF   FF   FF   F3   FF      . . . .  . . . .
  00020064-6B    FF   FF   F3   FF   FF   FF   FD   FF      . . . .  . . . .
  0002006C-73    FF   FF   FD   FF   FF   FF   F0   FF      . . . .  . . . .
  00020074-7B    FF   FF   FD   FF   FF   FF   F8   FF      . . . .  . . . .
  0002007C-83    FF   FF   FF   FF   FF   FF   FF   FF      . . . .  . . . .

STATUS:   N70732--Running user program__________________________________...R....
modify memory stackarea+10h string to 'A'


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

Breaking into the Monitor

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

break <RETURN>

Notice that the current address is pointed out with inverse video in displaying memory when the execution breaks to the monitor.

Using Software Breakpoints

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

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. Further, your program won’t work correctly.

Note

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

Note

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

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

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

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

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>

When software breakpoints are enabled and you set a software breakpoint, the 70732 breakpoint interrupt instruction (BRKRET) will be placed at the address specified. When the breakpoint interrupt instruction is executed, program execution will break into the monitor.

Setting a Software Breakpoint

To set a software breakpoint at line 68 of "spmt_demo.c", enter the following command.

modify software_breakpoints set line 77 <RETURN>

To see the address where the software breakpoint has been set, enter the following command:

display memory line 77 ***mnemonic*** <RETURN>
set source on inverse_video on <RETURN>

```
 Memory  :mnemonic :file = spmt_demo.c:
   address  label          data
      76            int i;
      77            for (i = 0; i  8; i++)
* 00000120                006C        BRKRET  0x00
  00000122                1000        MOV     R16,R0
  00000124                43CD1000    LD.W    0x0010[R3],R10
  00000128                484D        CMP     0x08,R10
  0000012A                229C        JGE     :scan_number+00038
      78            {
      79                      data = 0;
  0000012C                04DC1C03    ST.W    R0,0x031C[R4]
      80                      data = 1;
  00000130                4141        MOV     0x01,R10
  00000132                44DD1C03    ST.W    R10,0x031C[R4]
      81                      stack = 0;
  00000136                04DC2003    ST.W    R0,0x0320[R4]
      82            }

STATUS:   N70732--Running in monitor____________________________________...R....
set source on inverse_video on


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

The asterisk (*) in left side of the address lists points out that the software breakpoint has been set. The opcode at the software breakpoint address was replaced to the software breakpoint instruction (BRKRET).

Displaying Software Breakpoints

To display software breakpoints, enter the following command.

display software_breakpoints <RETURN>

```
Software breakpoints  :enabled
   address         label                                            status
  00000120         spmt_demo.c:                        line   77  pending











STATUS:   N70732--Running in monitor____________________________________...R....
display software_breakpoints


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

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

After the software breakpoint has been set, enter the following command to cause the emulator to continue executing the demo program.

run `<RETURN>`

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

The software breakpoint address is pointed out with inverse video in displaying memory in mnemonic format. To see the software breakpoint with memory, enter the following command.

display memory `line 77` ***mnemonic*** `<RETURN>`

Notice that the original opcode was replaced at the address that the software breakpoint has been set.

Clearing a Software Breakpoint

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

modify software_breakpoints clear `line 77 <RETURN>`

The breakpoint is removed from the list, and the original opcode is restored if the breakpoint was pending.

To clear all software breakpoints, you can enter the following command.

modify software_breakpoints clear `<RETURN>`

Displaying Registers

Enter the following command to display registers. You can display the basic registers, or an individual register. Refer to "REGISTER CLASS and NAME" section in "Using the Emulator" chapter .

display registers `<RETURN>`

```
Registers

Next PC 00000120
 PC 00000120  PSW 00008000
 R0-7   00000000 00000018 00000000 000203CC 00020000 00000000 00000000 00000000
 R8-15  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 R16-23 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 R24-31 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000714









STATUS:   N70732--Running in monitor      Software break: 000000120_____...R....
display registers


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

Stepping Through the Program

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

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

You will see the inverse-video moves according to the step execution. You can continue to step through the program just by pressing the <RETURN> key.

```
Registers

Next PC 00000120
 PC 00000120  PSW 00008000
 R0-7   00000000 00000018 00000000 000203CC 00020000 00000000 00000000 00000000
 R8-15  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 R16-23 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 R24-31 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000714

Step_PC 00000120  ST.W    R0,0x0010[R3]
Next PC 00000124
 PC 00000124  PSW 00008000
 R0-7   00000000 00000018 00000000 000203CC 00020000 00000000 00000000 00000000
 R8-15  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 R16-23 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 R24-31 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000714



STATUS:   N70732--Stepping complete_____________________________________...R....
step


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

You can step program execution by source lines, enter:

step source <RETURN>

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

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.

Source Line Referencing

A trace may be taken and displayed using source line referencing. Also, lines of the source program can be displayed with the trace list where the trace occurred.

To display the trace with source code in inverse video, enter the following command:

set source on inverse_video on <RETURN>

Specifying a Simple Trigger

Suppose you want you trace program execution after the point at address **semantic_check**. The following command make this trace specification.

trace after semantic_check <RETURN>

The STATUS message shows "Emulation trace started.".

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

run <RETURN>

The STATUS message shows "Emulation trace complete.".

Display the Trace

The trace listings which following are of program execution on the 70732 emulator. To see the trace list, enter the following command:

display trace <RETURN>

```
Trace List                 Offset=0
Label:      Address       Data    Opcode or Status w/ Source Lines
Base:       symbols       hex            mnemonic w/symbols
after  s:semantic_check 0018A020    0018A020 fetch after branch
      ##########spmt_demo.c - line   201 thru   203 #####################

      semantic_check()
      {
+001   s:semantic_check DFE30861  MOVEA   0x0018,R0,R1
                                    DFE30861 fetch
+002   :semant+00000004 8A800014  SUB     R1,R3                  G
                                    8A800014 fetch
+003   :semant+00000006 0010DC03  ST.W    R31,0x0014[R3]         G
                                    0010DC03 fetch
+004   :semant+0000000A 0010CD43  JBR     .Rsemantic_check       G
                                    0010CD43 fetch
+005        000203E0    000007A0    000007A0 data write word
+006   :semant+00000088 8B82181F    8B82181F fetch after branch

STATUS:   N70732--Running user program    Emulation trace complete______...R....
display trace


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

The trace list shows the trace after line (semantic_check()).

To list the next 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 <RETURN>

```
Trace List                Offset=0
Label:  Address   Data               Opcode or Status w/ Source Lines
Base:     hex     hex                            mnemonic
after  00000460 0018A020    0018A020 fetch after branch
      ##########spmt_demo.c - line   201 thru   203 ############################

      semantic_check()
      {
+001   00000460 DFE30861  MOVEA   0x0018,R0,R1
                            DFE30861 fetch
+002   00000464 8A800014  SUB     R1,R3                   GSS
                            8A800014 fetch
+003   00000466 0010DC03  ST.W    R31,0x0014[R3]          GSS
                            0010DC03 fetch
+004   0000046A 0010CD43  JBR     0x000004EA              GSS
                            0010CD43 fetch
+005   000203E0 000007A0    000007A0 data write word
+006   000004E8 8B82181F    8B82181F fetch after branch

STATUS:   N70732--Running user program    Emulation trace complete______...R....
set symbols off


  run     trace     step   display          modify   break     end    ---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 trace list specifies the following information.

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

Displaying Trace with Compress Mode

If you want to see more executed instructions on a display, the 70732 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>

```
Trace List                Offset=0
Label:  Address   Data               Opcode or Status w/ Source Lines
Base:     hex     hex                            mnemonic
      ##########spmt_demo.c - line   201 thru   203 ############################

      semantic_check()
      {
+001   00000460 DFE30861  MOVEA   0x0018,R0,R1
+002   00000464 8A800014  SUB     R1,R3                     GSS
+003   00000466 0010DC03  ST.W    R31,0x0014[R3]            GSS
+004   0000046A 0010CD43  JBR     0x000004EA                GSS
+005   000203E0 000007A0    000007A0 data write word
      ##########spmt_demo.c - line   216 #######################################
      }
+007   000004EA 0018A020  JBR     0x0000046C
      ##########spmt_demo.c - line   204 thru   205 ############################
              int i;
              for (i = 0; i  4; i++)
+009   0000046C 0010CD43  ST.W    R0,0x0010[R3]

STATUS:   N70732--Running user program    Emulation trace complete______...R....
display trace compress on


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

As you can see, the analysis trace display shows the analysis trace lists without fetch 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.

Trigger the Analyzer at an Instruction Execution State

The emulator analyzer can capture states of instruction execution. If you want to trigger the analyzer when an instruction at a desired address is executed, you should not set up the analyzer trigger condition to detect only the address. If you do so, the analyzer will be also triggered in case that the address is accessed to fetch the instruction, or read the data from address. You should use the "**exec**" status qualifier.Suppose that you want to trace the states of the execution after the instruction at *clear_buffer*of the *spmt_demo.c* file, enter the following command.

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

The message "Emulation trace started" will appear on the status line, and the status line now shows "Emulation trace complete".

```
Trace List                 Offset=0
Label:  Address   Data                Opcode or Status w/ Source Lines
Base:     hex     hex                             mnemonic
      ##########spmt_demo.c - line   153 thru   157 ############################

      /******************** level three *******************/

      clear_buffer()
      {
after  000002FC DFE30861  MOVEA   0x0018,R0,R1
                            DFE30861 fetch
+001   00000300 8A5E0014  SUB     R1,R3                  GSS
                            8A5E0014 fetch
+002   00000302 0010DC03  ST.W    R31,0x0014[R3]         GSS
                            0010DC03 fetch
+003   00000306 0010CD43  JBR     0x00000364             GSS
                            0010CD43 fetch
+004   000203E0 00000680    00000680 data write word

STATUS:   N70732--Running user program    Emulation trace complete______...R....
trace after clear_buffer status exec


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

The emulator has disassemble capability in trace listing. When the emulator disassembles instructions in stored trace information, the fetch cycles of each instruction are required. When you displayed the results of analyzer trace, some lines which include "INSTRUCTION--opcode unavailable" message may be displayed. Each line is instruction execution cycle at the address in the left side of the displayed because the fetch states for the instructions were not stored by the analyzer.

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

Disassembling trace by memory contents

You can specify whether the 70732 emulator read data from memory or from trace memory when the emulator display trace list. To specify, the 70732 emulator Softkey Interface provides display option. To read data from memory, enter the following command:

display trace mnemonic option disassemble_by_memory_contents <RETURN>

As you can see, "GSS" in the right side of trace list disappears. If "INSTRUCTION--opcode unavailable" messages were displayed on some lines, these line can be disassembled.

If you want the 70732 emulator to read data from trace list to disassemble, enter the following command:

display trace mnemonic option disassemble_by_trace_data <RETURN>

Emulator Analysis Status Qualifiers

The following analysis status qualifiers may also be used with the 70732 emulator.

```
backgrnd  0xxxxxxxxxxxxxxxy      background
addrerr   0xx1xxxx1xxxxxxxxy     bus lock
byte      0xx1xxxxxxxxxx0xxy     byte access
data      0xx1xxxxxxx010xxxy     data access
exec      0xxx1xxxxxxxxxxxxy     execute instruction
fault     0xx1xxxxxxx101xxxy     machine fault acknowledge
fetch     0xx1xxxxxx1011xxxy     code fetch
fetchbr   0xx1xxxxxx1001xxxy     code fetch after branch
foregrnd  01xxxxxxxxxxxxxxxy     foreground
grdacc    0xx1xx0xxxx0xxxxxy     guarded memory access
halt      0xx1xxxxxxx111xxxy     halt acknowledge
hold      0xx00xxxxxxxxxxxxy     hold acknowledge
halfwd    0xx1xxxxxxxxxx10xy     half word access
io        0xx1xxxxxxx110xxxy     I/O access
mem       0xx1xxxxxxx0xxxxxy     memory access
read      0xx1xxxxxx1xxxxxxy     read cycle
word      0xx1xxxxxxxxxx110y     word access
write     0xx1xxxxxx0xxxxxxy     write cycle
wrrom     0xx1x0xxxx00xxxxxy     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>

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 Topics

Introduction

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

This chapter will:

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

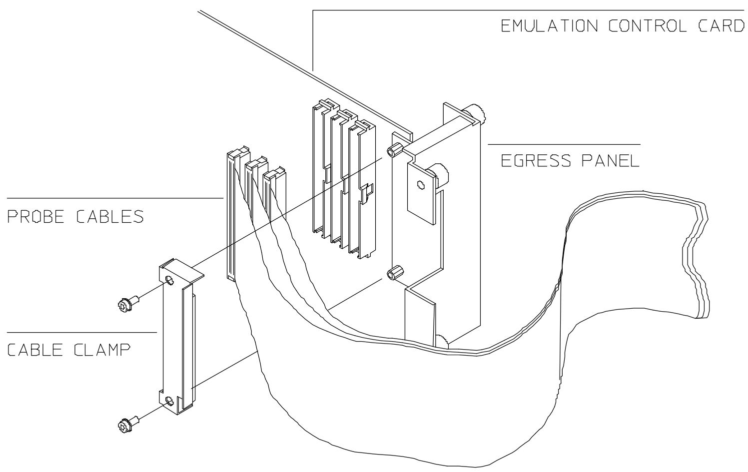
Prerequisites

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.

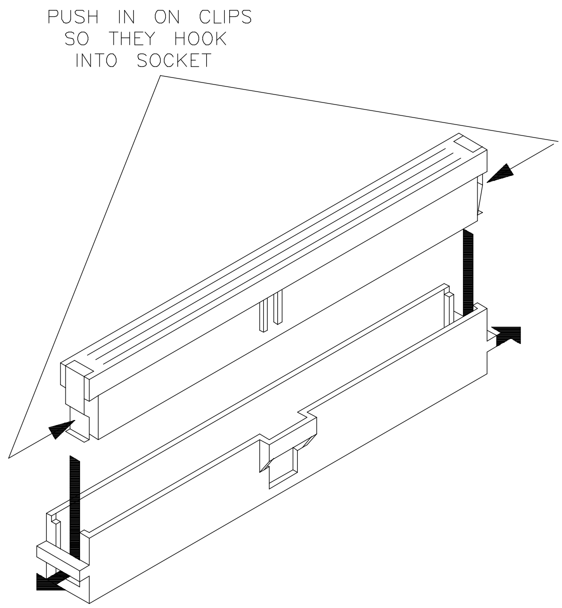
Installing the Emulation Probe Cable

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

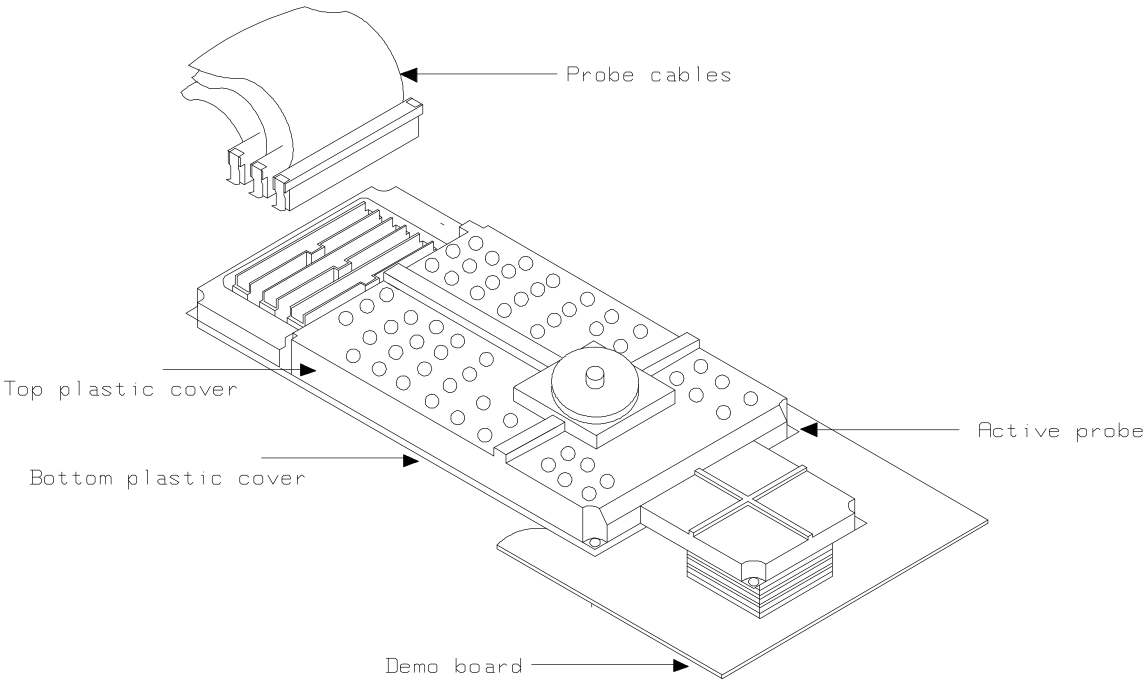
1. Secure the cable on the emulation control card with cable clamp and two screws.



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



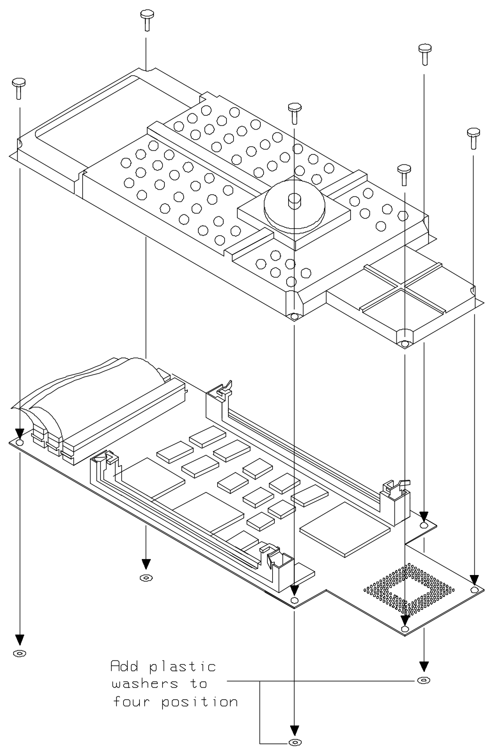
3. Connect the other ends of the cables to the emulation probe.



Installing the Emulation Memory Module

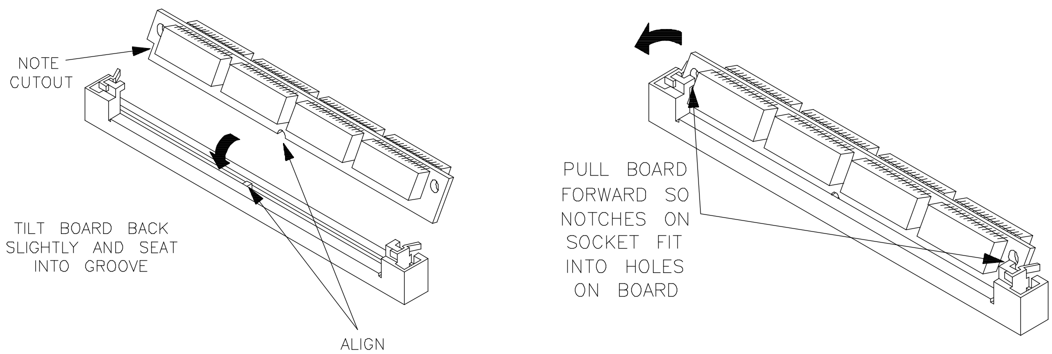
There are four types of emulation memory modules that can be inserted into sockets on the probe.

1. Remove plastic rivets that secure the plastic cover on the top of the emulator probe, and remove the cover. The bottom cover is only removed when you need to replace a defective active probe on the exchange program.



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

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



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

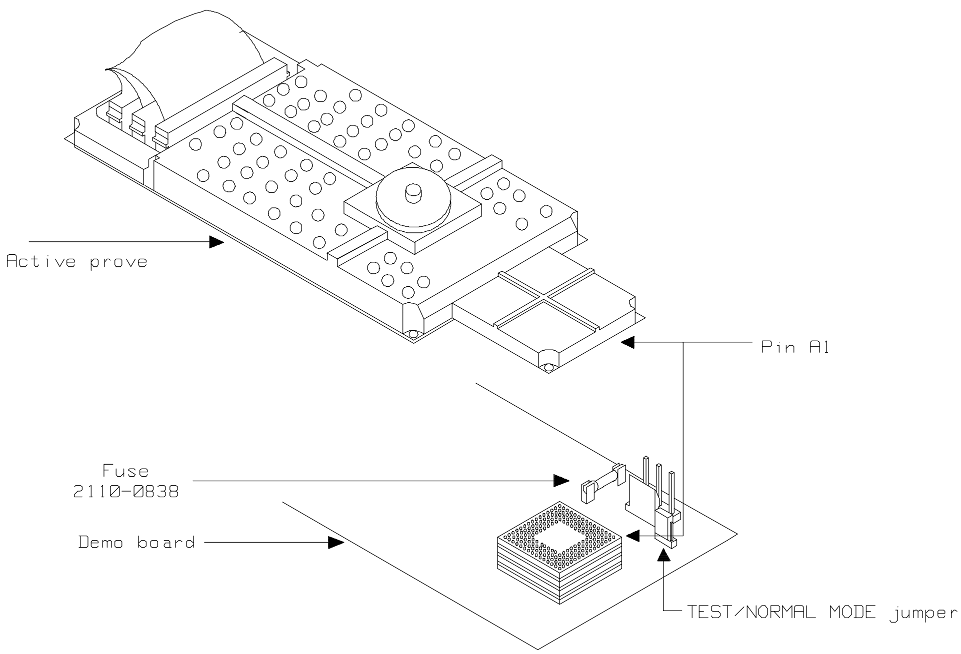
Installing into the Demo Target Board

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

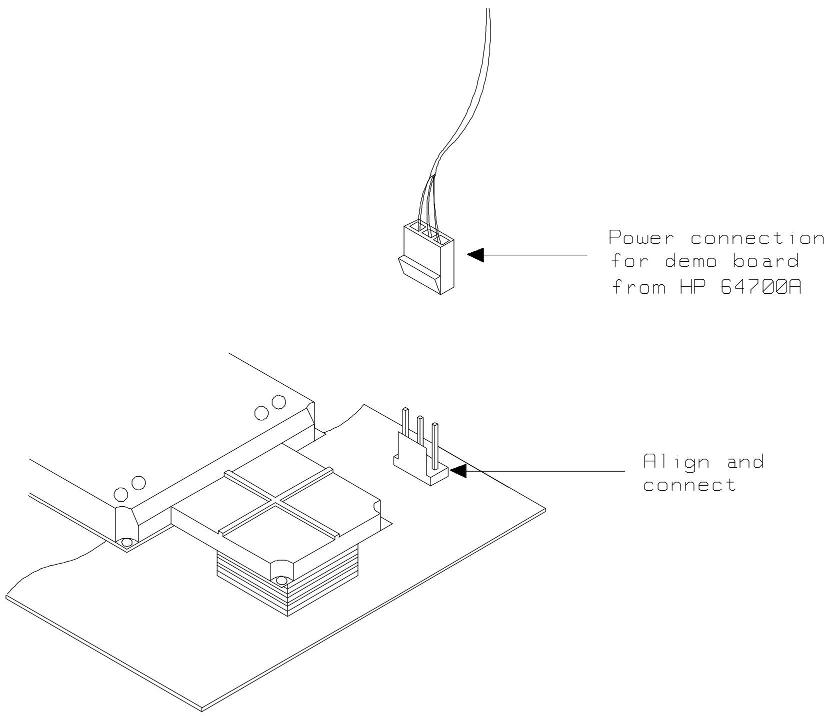
1. Remove front bezel and connect the power cable to connector the HP 64700B front panel. Refer to *HP 64700 Series Installation/Service* manual.

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

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



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



Installing the Emulator Probe into a Target System

The 70732 emulator probe has a 176-pin PGA connector;
The emulator probe is also provided with a conductive pin protector to protect the delicate gold-plated pins of the probe connector from damage due to impact. Since the protector is non-conductive, you may run performance verification with no adverse effects when the emulator is out-of-circuit.

Caution

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

Caution

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

Caution

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

Installing into a PGA Type Socket

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

1. Remove the 70732 microprocessor (PGA type) from the target system socket. Note the location of pin A1 on the microprocessor and on the target system socket.

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

3. Install the emulator probe into the target system microprocessor socket.

Caution

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

Note

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

Installing into a QFP Type Socket

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

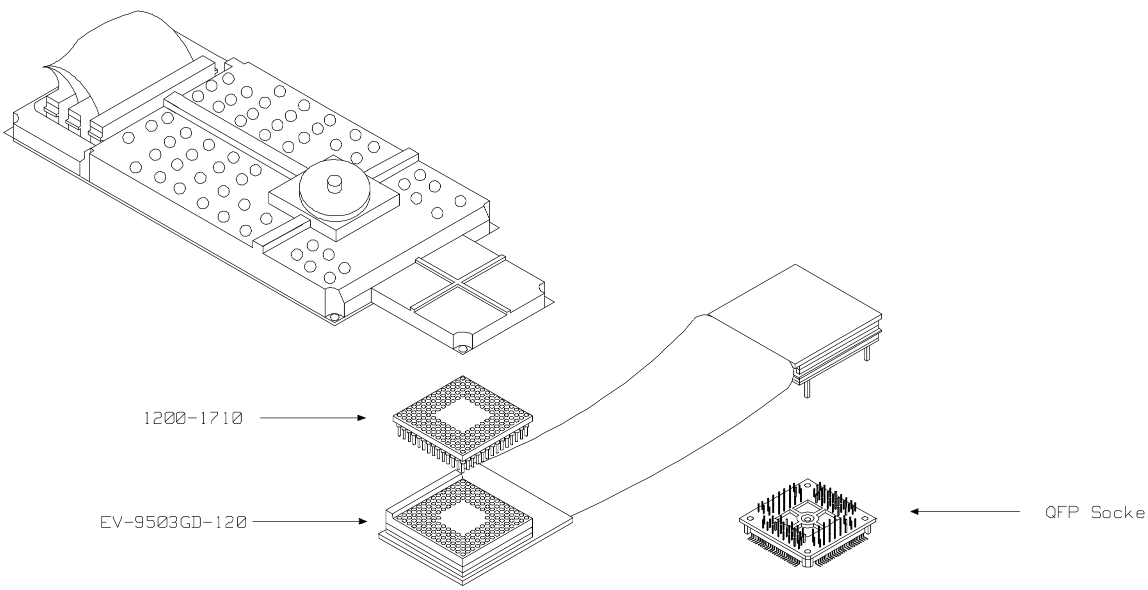
1. Attach the QPF socket to your target system.

2. Connect the NEC EV-9503-GD-120 to QPF socket on your target system.

3. Connect the IC-Socket(1200-1710) to the ZIP socket on NEC EV-9503-GD-120.

4. Place the 70732 emulator probe to the NEC EV-9503-GD-120 with IC-Socket.

Note Contact NEC Electronics Inc. to purchase QFP socket.



In-Circuit Configuration Options

The 70732 emulator 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.

Allowing the Target System to Insert Wait State

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.

Enabling SZRQ, NMI, HLDRQ and RESET Input from the Target System

You can configure whether the emulator should accept or ignore the SZRQ, NMI, HLDRQ 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 "Configuring the Emulator" chapter of this manual).

To specify a run from target system reset, enter the following command:

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.

Note

In the "Awaiting target reset" status, you can not break into the monitor. If you enter "run from reset" in the configuration that emulator does not accepted target system reset, you must reset the emulator.

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

1) Enter "run from reset"

2) Turn OFF your target system

3) Turn ON your target system

Note

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

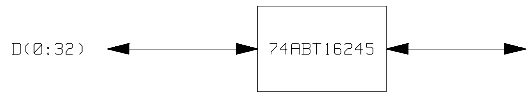
Pin State in Background

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

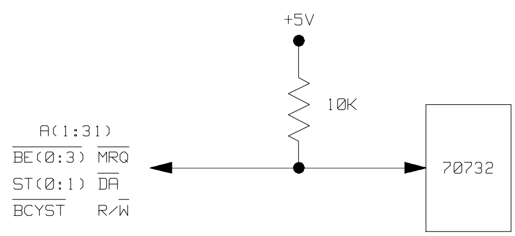
Address Bus	Same as foreground
Data Bus	Always high impedance otherwise you direct the emulator to access target memory. When accessing target memory, I/O by background monitor, same as foreground.
$\overline{\mathrm{DA}}$	When you specify that the emulator drives background cycles to target system, same as foreground. When you specify that the emulator does not drive background cycles to target system, always high impedance otherwise you direct the emulator to access target memory. When accessing target memory, I/O by background monitor, same as foreground.
R/$\overline{\mathrm{W}}$	Always high level, except accessing target memory, I/O by background monitor.
$\overline{\mathrm{BCYST}}$	When you specify that the emulator drives background cycles to target system, same as foreground. When you specify that the emulator does not drive background cycles to target system, always high level, except accessing target memory, I/O by background monitor.
Other	Same as foreground

Target System Interface

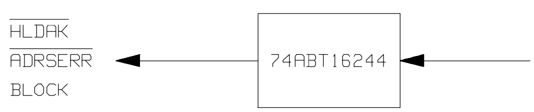
D0-D31 These signals are connected to 74ABT16245.



A(1:31) BE(0:3) BCYST DA ST(0:1) R/W MRQ These signals are connected to 70732 emulation processor through 10k ohm pull-up register.

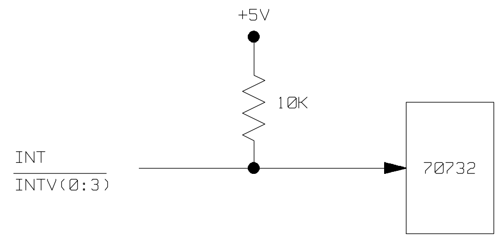


HLDAK BLOCK ADRSERR These signals are connected to 74ABT16244.



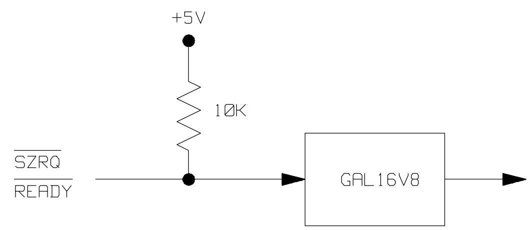
INT
$\overline{\text{INTV(0:3)}}$

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



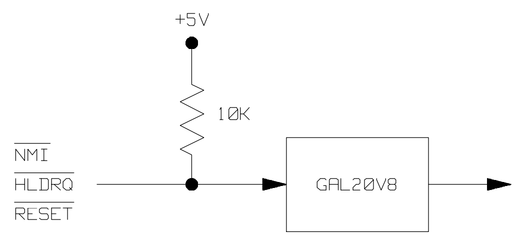
$\overline{\text{SZRQ}}$
$\overline{\text{READY}}$

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



$\overline{\text{HLDRQ}}$ $\overline{\text{NMI}}$
$\overline{\text{RESET}}$

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



4

Configuring the Emulator

Introduction

Your 70732 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 70732 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:

- Selecting monitor entry after configuration.
- Restricting to real-time execution.
- Selecting processor data bus width.
- Enabling the instruction cache.
- Keeping coherence of the cache.

Memory Configuration:

- Selecting the emulation monitor type.
- Specifying value for address during background operation.
- Mapping memory.

Emulator Pod Configuration:

- Inserting wait state at BANK0 emulation memory.
- Inserting wait state at BANK1 emulation memory.
- Enabling NMI input from target system.
- Enabling HLDRQ input from target system.
- Enabling RESET input from target system.
- Enabling READY input from target system.
- Enabling SZRQ input from target system.
- Selecting target memory access size.
- Driving background cycles to target system.

Debug/Trace Configuration:

- Enabling breaks on writes to ROM.
- Specifying tracing of foreground/background cycles.
-

- Selecting trace mode.
- Specifying tracing of fetch cycles.
- Specifying forcing to trace bus address.
- Selecting emulation analyzer speed.

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

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

General Emulator Configuration

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

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.

yes When reset to monitor is selected, the emulator will be running in the monitor after configuration is completed. 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?

This configuration allows to you specify whether program execution should take place in real-time or whether commands should be allowed to cause breaks to the monitor during program execution.

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.

Processor data bus width?

This configuration allows to you specify whether data bus width is 16bits or 32bits.

32	Data bus width will be 32 bits.
16	Data bus width will be 16 bits.

Note

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

Note

Changing this configuration setting will drive the emulator into a reset state.

Enable the instruction cache?

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

yes When the instruction cache is enable, the Cache Control Word Register(CHCW) determines whether the cache is ultimately enabled. Enabling the on-chip instruction cache memory improves performance of the processor and can greatly reduce the activity on the processor’s external bus.

no When the instruction cache is disabled, the emulator will prevent enabling the cache. Disabling the on-chip instruction cache memory will force the processor to always access external memory. The cache should be disabled whenever tracing program execution to force all external memory accesses to be visible to the analyzer.

Note

The 70732 emulator operates in accordance with this configuration instead of $\overline{\text{ICHEEN}}$ signal from target system. $\overline{\text{ICHEEN}}$ signal from target system is ignored.

Note

Changing this configuration setting will drive the emulator into a reset state.

Keep coherence of the cache?

This configuration allow you to specify whether or not memory is coherent with instruction cache when the emulator modify memory. You must answer this question, when you answered "yes" at the "Enable the instruction cache?" question.

yes When keeping cache coherence is enabled, the emulator breaks into the monitor to keep cache coherence whenever the emulator writes to the memory. The monitor checks the cache contents and update both cache and memory when the emulator tried to write the same address that has been cached.

no When keeping cache coherence is disabled, the emulator does not check the cache contents when the emulator writes to the memory. Therefore, the cache contents may different from the memory contents when they were modified by the emulator.

Note

When the monitor is restricted to real time and keeping cache coherence is enabled, the emulator can not modify emulation memory while the emulator is running the user program.

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 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 70732 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" or "user_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. Value for address during background operation?

You can select the value that will be driven to the target system on A31-A13 during background monitor operation. The value should be 32 bit address and must be multiple of 8K(2000H).

foreground When you select the build_in foreground monitor, processor address space is taken up. The foreground monitor takes up 8K 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. Monitor location?

You can relocate the monitor to any 8K 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 8K byte boundaries; otherwise, the configuration will be invalid, and the previous configuration will be restored.

user_foreground When you select the custom foreground monitor, processor address space is taken up. The foreground monitor takes up 8K 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. Monitor location?

You can relocate the monitor to any 8K 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 8K byte boundaries; otherwise, the configuration will be invalid, and the previous configuration will be restored.

3. Monitor filename?

This question allows you to specify the name of the custom foreground monitor program absolute file. Remember that you must assemble and link your foreground monitor starting at the 8K byte boundary specified for the previous "Monitor location?" question.

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

Only the 8k bytes 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.

Mapping Memory

The emulation memory consists of 256k, 512k 1M, 1.25M or 2Mbytes, mappable in 4k byte blocks. The emulator distinguish left side memory module(BANK 0) and right side ones(BANK 1) because you can select memory modules whose access speed is different on each bank. If you will use HP64712A/B, the emulation memory system does not introduce wait states. If you will use HP64171A/B and clock speed is less than 20MHz, the emulation memory system does not introduce wait state. If you will use HP64171A/B and clock speed is greater than 20MHz, the emulation memory system introduce one wait state.

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 you characterize memory ranges as emulation RAM/ROM, you can specify whether bank number is to be BANK 0(**b0**) or BANK 1(**b1**) and whether data bus size is to be 16(**d16**) or 32(**d32**) as attributes. When you do not specify bank number, the emulator interprets that bank number is "b0". If you do no specify data bus size, the emulator interprets that data bus size is "32".

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

b0_d32	Using emulation memory of bank 0 and data bus width is 32 bits.
b0_d16	Using emulation memory of bank 0 and data bus width is 16 bits.
b1_d32	Using emulation memory of bank 1 and data bus width is 32 bits.
b1_d16	Using emulation memory of bank 1 and data bus width is 16 bits.
b0	Using emulation memory of bank 0 and data bus width is 32 bits.
b1	Using emulation memory of bank 1 and data bus width is 32 bits.

When a foreground monitor selected, a 8k byte block is automatically mapped at the address specified by the "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.

Emulator Pod Configuration

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

Modify emulator pod configuration?

Inset wait state at BANK0 emulation memory?

This configuration allows you to specify whether or not the emulator insert wait state when BANK0 emulation memory is accessed.

yes	Inserting one wait state at BANK0 emulation memory.
no	Inserting no wait state at BANK0 emulation memory.

Inset wait state at BANK1 emulation memory?

This configuration allows you to specify whether or not the emulator insert wait state when BANK1 emulation memory is accessed.

yes Inserting one wait state at BANK1 emulation memory.

no Inserting no wait state at BANK1 emulation memory.

Note

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

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

Enable NMI input 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

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

Enable responding to HLDRQ signal?

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

yes The emulator accepts HLDRQ signal. When the HLDRQ is accepted, the emulator will respond as actual microprocessor.

no The emulator ignore HLDRQ signal from target system completely.

Enable RESET input from target system?

The 70732 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 70732 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 READY input 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.

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).

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).

Enable SZRQ input from target system?

This configuration allow you specify whether or not the emulator accepts SZRQ(Size Request) signal generated by the target system.

yes The emulator accepts SZRQ signal while accessing to emulation memory.

no The emulator ignores SZRQ signal while accessing to emulation memory. The emulator will determine the bus width of emulation memory by the attribute setting specified in the mapping process.

Target memory 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 byte cycles (one byte at a time).

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

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

any Selecting the any access mode specifies that the emulator will access target memory using a display/modify target memory command option. If option "words" is specified, access size will be set to "words". Other target memory commands such as "load" and "store" will use an access size of "bytes".

Drive background cycles to target system?

This configuration allows you specify whether or not the emulator drives background cycles to target system.

yes The emulator will drive address,and control strobes to the target system during background monitor operation. No write cycles will occur except those needed for modify target memory commands.

no The emulator will not drive BCYST and DA signals during background monitor operation. Other strobes and address are driven to the target system.

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.

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 mode?

This question allows you to specify whether or not the analyzer trace execution cycles. You must answer this question, when you answered "no" at the "Enable the instruction cache" question.

exe Specifies that the analyzer will trace execution cycles. A single emulation analyzer state will be generated at each time the processor executes an instruction. Mnemonic will be displayed only on these execution cycles. If the execution and the bus cycles are generated simultaneously, no bus addresses are captured by the analyzer.

bus Specifies that the analyzer will trace only actual bus cycles. You must answer this question, when you answered "exe" at the "Trace mode?" question.

Trace fetch cycles?

This question allows you to specify whether or not the analyzer trace the 70732 emulation processor’s refresh cycles. You must answer this question, when you answered "exe" at the "Trace mode?" question.

yes Specifies that the analyzer will trace fetch cycles.

no Specifies that the analyzer will not trace fetch cycles. Fetch data needed to mnemonic display will be read from the emulation or the target memory depending on the executed address. If the emulator failed to read memory, an error(s) is generated during mnemonic displaying and no mnemonic will be displayed.

Note

If you specify that the analyzer will trace only actual bus cycles, the analyzer will trace fetch cycles regardless of this configuration.

Force to trace bus address?

This question allows you specify whether or not forcing the analyzer to trace the address of bus cycles as its data. You must answer this question, when you answered "exe" at the "Trace mode?" question.

yes Specifies bus cycle addresses are traced as their data. In this mode, both execution and bus cycle addresses are shown and no data are available on the trace list.

no Specifies bus cycle addresses are available only when no execution was generated simultaneously.

Emulation analyzer speed?

This question allows you specify the emulation processor clock speed. The analyzer capabilities of time and state count are affected by the processor clock speed. You must answer this question, when you answered "exe" at the "Trace mode?" question, or when you use HP 64704A emulation bus analyzer.

slow Specifies the processor clock speed is less than or equal to 16.00 MHz. Both state and time counting are available.

fast Specifies the processor clock speed is less than or equal to 20.00 MHz. Only state counting are available.

veryfast Specifies the processor clock speed is greater than 20.00 MHz. Neither state or time counting are available.

Simulated I/O Configuration

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

Interactive Measurement Configuration

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

This chapter discuss:

- Manipulation in floating-point form
  - Register manipulation
  - Memory manipulation
- Register names and classes
- Hardware breakpoint
- Analyzer topics
  - Specifying trace configuration
  - Specifying data for trigger or store condition
- Features available via "pod_command"

This chapter shows you how to:

Emulation memory access

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

Manipulation in Short-real Format

You can display/modify register and memory in short-real format.

Register Manipulation

You can display/modify general purpose registers as 32-bit(single-precision) real numbers. The IEEE-754 standard data type is supported. To display/modify the general purpose registers as 32-bit(single-precision) real numbers, use the following register names with the **FLOAT** attribute.

- **FR0** thru **FR31** for short real(32-bit single-precision)

To display register R0 32-bit(single-precision) real numbers, enter;

display register FLOAT FR1 <RETURN>

```
Registers

 FR1 +5.4823596E+002










STATUS:   N70732--Running in monitor____________________________________...R....
display registers  FLOAT  FR1

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

To modify register R2 to the value 12345.678, enter;

modify register FLOAT FR2 to 12345.678
<RETURN>

Memory Manipulation

You can display/modify memory value as 32-bit(single-precision) real numbers. The IEEE-754 standard data type is supported. To access to the memory as 32-bit(single-precision) real numbers, use the following commands.

To display memory at 1000H as 32-bit(single-precision) real numbers, enter;

display memory 1000h ***real short*** <RETURN>

```
 Memory  :short real :update
   address      data  :real
  00001000      3.65837E+003
  00001004      6.58637E+003
  00001008      1.23699E+003
  0000100C     -6.36223E+003
  00001010      3.33654E-001
  00001014      2.54100E-004
  00001018     -2.54100E-004
  0000101C      3.43657E-001
  00001020     -3.40277E+038
  00001024      9.83237E+002
  00001028     -8.32367E+001
  0000102C      6.98743E-001
  00001030      1.58790E+004
  00001034      1.63215E+002
  00001038      1.96022E+005
  0000103C     -6.53000E-003

STATUS:   N70732--Running in monitor____________________________________...R....
display memory 1000h real short


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

To modify memory at 1004h to 123.456, enter;

modify memory 1004h real short to 123.456 <RETURN>

REGISTER CLASS and NAME

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

<REG_CLASS>

<REG_NAME> Description

***(All basic registers)**

PC PCW	BASIC registers.
R0 R1 R2 R3 R4	
R5 R6 R7 R8 R9	
R10 R11 R12 R13	
R14 R15 R16 R17	
R18 R19 R20 R21	
R22 R23 R24 R25	
R26 R27 R28 R29	
R30 R31	

SYS(System Control registers)

EIPC	Exception/Interrupt PC	
EIPSW	Exception/Interrupt PSW	
FEPC	Fatal error PC	
FEPSW	Fatal error PSW	
ECR	Exception cause	(Read Only)
PIR	Processor ID	(Read Only)
TKEW	Task control word	(Read Only)
CHCW	Cache control word	
SDTRE	Address trap	

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.

Analyzer Topics

You can specify trace configuration, To do this, you should answer question of the Debug/Trace configuration. Refer to the "Debug/Trace Configuration" section of the "Configuration the Emulator" chapter.

Trace actual bus cycles

You can specify that the analyzer trace only actual bus cycles. In this case, analyzer can not trace execution state. When you display trace list, the emulator disassembles with "fetch" states, and their disassembled processor mnemonics is displayed at the "fetch" states which are the first byte of the instruction. To trace actual bus cycles, you answer "bus" at the "Trace mode?" question of the Debug/Trace configuration. In this case, you will see trace list like following.

```
Trace List                Offset=0
Label:  Address   Data                        Opcode or Status
Base:     hex     hex                             mnemonic
after  0000008C 031CDC04  ST.W    R0,0x031C[R4]
+001   00000090 DD444141  MOV     0x01,R10
      =00000092           ST.W    R10,0x031C[R4]
+002   00000094 DC04031C    DC04031C fetch
      =00000096           ST.W    R0,0x0320[R4]
+003   0002031C 00000000    00000000 data write word
+004   00000098 CD430320    CD430320 fetch
      =0000009A           LD.W    0x0010[R3],R10
+005   0002031C 00000001    00000001 data write word
+006   0000009C 45410010    45410010 fetch
      =0000009E           ADD     0x01,R10
+007   00020320 00000000    00000000 data write word
+008   0002031C 00000001    00000001 data read word
+009   000000A0 0010DD43  ST.W    R10,0x0010[R3]
+010   000000A4 0010CD43  LD.W    0x0010[R3],R10

STATUS:   N70732--Running user program    Emulation trace complete______...R....
display trace


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

Disassembling from the higher half-word

When the analyzer trace actual bus states, you can force disassembly to begin with higher half-word of first trace state. If the disassembled trace list is not what you expected, enter the command as following.

display trace disassemble_from_line_number <LINE_NUMBER> ***high_half_word*** <RETURN>

Using the Storage Qualifier

You can use storage qualifier to trace only states with specific condition. When you specify states to be stored in the trace memory, you use the "only" option. To save states selectively, enter the command as following.

trace only <QUALIFIER> <RETURN>

Using analyzer counter

When you specify that the analyzer trace actual bus cycles, you can use the analyzer counter because the clock speed can be effectively halved even if clock speed is greater than 20MHz. Refer to "Debug/Trace Configuration" section in the "Configuring the Emulator" chapter. To count time, enter the command as following:

trace after <QUALIFIER> ***counting time***
<RETURN>

When you use the analyzer counter, you will see trace list like following.

```
Trace List                 Offset=0
Label:  Address   Data               Opcode or Status              time count
Base:     hex     hex                    mnemonic                   relative
after  00000000 0002BC20  MOVHI   0x0002,R0,R1                    ------------
+001   00000004 042CA061  MOVEA   0x042C,R1,R3                      120     nS
+002   00000008 0000BC20  MOVHI   0x0000,R0,R1                      120     nS
+003   0000000C 0000A0A1  MOVEA   0x0000,R1,R5                      120     nS
+004   00000010 0002BC20  MOVHI   0x0002,R0,R1                      120     nS
+005   00000014 0000A081  MOVEA   0x0000,R1,R4                      120     nS
+006   00000018 AC000485  ADD     R5,R4                             120     nS
      =0000001A           JAL     0x00000A1C
+007   0000001C 68000A02    68000A02 fetch                          120     nS
      =0000001E           HALT
+008   00000020 0018A020  MOVEA   0x0018,R0,R1                      120     nS
+009   00000A1C 0018A020  MOVEA   0x0018,R0,R1                      120     nS
+010   00000A20 DFE30861  SUB     R1,R3                             120     nS
      =00000A22           ST.W    R31,0x0014[R3]
+011   00000A24 8A6E0014    8A6E0014 fetch                          120     nS

STATUS:   N70732--Running user program    Emulation trace complete______...R....
display trace


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

Not trace fetch cycles

You can specify that the analyzer does not trace fetch cycles. Not to trace fetch cycles, you answer "no" at the "Trace fetch cycle?" question of the Debug/Trace configuration. In this case, the emulator read data from memory when the emulator disassembles trace list and you will see trace list like following.

```
Trace List                 Offset=0
Label:  Address   Data                        Opcode or Status
Base:     hex     hex                             mnemonic
after  00000254 DD444147  ST.W    R10,0x031C[R4]
+001   00000258 00000005  MOV     0x07,R10
                            00000005 data write word
+002   0002031C 00000006    00000006 data write word
+003   0000025A 41410320  ST.W    R10,0x031C[R4]
+004   0000025E 0320DD44  ST.W    R0,0x0320[R4]
+005   00000262 00000007  MOV     0x01,R10
                            00000007 data write word
+006   00000264 00000007  ST.W    R10,0x0320[R4]
+007   00020320 00000000    00000000 data write word
+008   00020320 00000001    00000001 data write word
+009   00000268 41430320  MOV     0x02,R10
+010   0000026A 0320DD44  ST.W    R10,0x0320[R4]
+011   0000026E 0010CD43  MOV     0x03,R10
+012   00000270 0010CD43  ST.W    R10,0x0320[R4]

STATUS:   N70732--Running user program    Emulation trace complete______...R....
display trace


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

Trace Bus Address

The 70732 emulator transfer execution address to analyzer preferentially and address of bus cycles may be missing on the trace list when execution state is generated simultaneously. You can specify that the analyzer trace address of bus cycles as its data. To force the analyzer to trace the address of bus cycles, you must answer "yes" at the "Force to trace bus address?" question of the Debug/Trace configuration. When you set this configuration, you will see trace list like following.

```
Trace List                      Offset=0
Label:  Address   Data                            Opcode or Status
Base:     hex     hex                                 mnemonic
after  0002031C 0002031C    ******** data write word
+001   0002031C 0002031C    ******** data read word
+002   000000A8 0002031C  CMP     0x06,R10
+003   000000AC 000000AC    ******** fetch
+004   000000AA 000000B0  JLT     0x0000008C
      =000000B0             ******** fetch
+005   0000008C 0000008C    ******** fetch after branch
+006   0000008C 00000090  ST.W    R0,0x031C[R4]
      =00000090             ******** fetch
+007   00000090 00000094  MOV     0x01,R10
      =00000094             ******** fetch
+008   00000092 0002031C  ST.W    R10,0x031C[R4]
      =0002031C             ******** data write word
+009   00000098 00000098    ******** fetch
+010   00000096 0002031C  ST.W    R0,0x0320[R4]

STATUS:   N70732--Running user program    Emulation trace complete______...R....
display trace


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

Specify Data for Trigger or Store Condition

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

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

The data condition will be considered as 00000041H. The bit 3 through bit 8 of data bus is unpredictable because of the byte data. You will unable to trigger as you desire. You should have entered as follows.

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

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

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

trace after 1000h ***data*** 0xxxx41xxh <RETURN>

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

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

```
Trace List                Offset=0
Label:  Address   Data                        Opcode or Status
Base:     hex     hex                             mnemonic
after  00010500 1001A080  MOVEA   0x1001,R0,R4
+001   00010504 1234A0A0  MOVEA   0x1234,R0,R5
+002   00010508 0000D4A4  ST.H    R5,0x0000[R4]
+003   0001050C 0000C4C4  LD.H    0x0000[R4],R6
+004   00010510 FFF0ABFF  JR      0x00010500
+005   00001000 00001234    ....1234 data write hword     ADR
+006   00001000 00001234    ....1234 data read hword      ADR
+007   00010500 1001A080  MOVEA   0x1001,R0,R4
+008   00010504 1234A0A0  MOVEA   0x1234,R0,R5
+009   00010508 0000D4A4  ST.H    R5,0x0000[R4]
+010   0001050C 0000C4C4  LD.H    0x0000[R4],R6
+011   00010510 FFF0ABFF  JR      0x00010500
+012   00001000 00001234    ....1234 data write hword     ADR
+013   00001000 00001234    ....1234 data read hword      ADR
+014   00010500 1001A080  MOVEA   0x1001,R0,R4

STATUS:   N70732--Running user program    Emulation trace complete______...R....
trace


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

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

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

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

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

trace find_sequence 1000h ***data*** 0xxxx5678
trigger after 1002h ***data*** 0xxxx1234h <RETURN>

Note

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

1) When you specify that the analyzer does not trace address of bus, you can not specify address in the analyzer trigger or store condition.
2) When you specify that the analyzer trace address of bus as its data, what you specify data in the analyzer trigger or store condition means that you specify address.
3) When execution state and bus state simultaneously, both states are stored in case that both states satisfy store condition.

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 your 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.0h)0

Accessing Emulation Memory

If you enter emulation memory display/modification commands while the user's program running, the 70732 emulation controller, not the emulation processor, holds the 70732 bus cycles for 12 clock cycles(not breaking into the monitor) in order to access to the emulation memory.

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.

Notes

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 HP 64752A emulator provides two kinds of foreground monitor. One is include in the emulator, the other is supplied with the emulation software and can be found in the following path:

`/usr/hp64000/monitor/`

The monitor program is named **fm70732.s**.

Comparison of Foreground and Background Monitors

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

Background Monitors

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

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

Processor resources such as interrupts are not fully taken by the background monitor.

However, all background monitors sacrifice some level of support for the target system. For example, 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 modifyed to handle special conditions.

Foreground Monitors

A *foreground* monitor may be required for more interrupt intensive applications. A foreground monitor is a block of code that runs in the same memory space as your program. Foreground monitors allow the emulator to service real-time events, such as interrupts, while executing in the monitor. For most multitasking, you will need to use a foreground monitor.

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

Foreground Monitor Selection

Then HP 64752A emulator provides two kinds of foreground monitor. One is included in the emulator, the other is provided with assemble source file.

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

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

Using Built-in Foreground Monitor

The 70732 emulator includes foreground monitor. The built-in foreground monitor saves your tasks for assembling, linking and loading the monitor. To use the built-in foreground monitor, all you have to do is to specify the location of the monitor.

Modifying the Emulator Configuration

The following assumes you are modifing 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.

Monitor address? 1000h

Specifies that the monitor will reside in the 8K byte block from 1000H through 2FFFH.

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

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

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

Using Custom Foreground Monitor

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

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

usr/hp64700/monitor/*

The monitor program is named **fm70732.s gfm70732.s**

If you use NEC K&R-C Compiler, you must modify the link directive file("fm70732.d") to use the custom foreground monitor.

```
.TEXT :  !LOAD   ?RX   V0x00006000 {
        .text =  $PROGBITS ?AX;
};
```

If you use Green Hills Software C-Compiler, you must modify the following statement of e monitor program("gfm70732.s") to use the custom foreground monitor.

```
MON_ADDR       =       0x000020000
```

The default monitor location is defined at address 0000H. You can load the monitor at any base address on a 8K byte boundary.

Assemble and Link the Monitor

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

```
$ as732 -w fm70723.s <RETURN>
$ ld732 -D fm70732.d -o fm70732.abs
 fm70732.o<RETURN>
$ v810cnv -x fm70732 <RETURN>
```

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

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

You 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.

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? user_foreground

Specifies that you will be using a foreground monitor program.

Monitor address? 6000h

Specifies that the monitor will reside in the 8K byte block from 6000H through 7FFFH.

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 file name? fm70732

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

An Example Using the Foreground Monitor

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

Mapping Memory for the Example

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

Modifing the Emulator Configuration

The following assumes that you have set up the foreground monitor and mapped for demo program. Answer as following.

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 demo program. You can load the demo program with the following command:

load `spmt_demo <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 70732 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  00000000 00008000    00008000 data write word      BGM       ------------
+001   00000004 FFFFFFF0    FFFFFFF0 data write word      BGM          80.    nS
+002   FFFFFFE0 00ACDFE0  ST.W    R31,0x00AC[R0]          BGM         440     nS
+003   FFFFFFE4 00A4DFA0  ST.W    R29,0x00A4[R0]          BGM          80.    nS
+004   FFFFFFE8 0200CFA0  LD.W    0x0200[R0],R29          BGM          80.    nS
+005   FFFFFFEC 9A00181D  JMP     [R29]                   BGM          80.    nS
      =FFFFFFEE           NOP                             BGM
+006   000000AC 00009000    00009000 data write word      BGM          80.    nS
+007   000000A4 00000000    00000000 data write word      BGM          80.    nS
+008   00000200 00006300    00006300 data read word       BGM          80.    nS
+009   FFFFFFF0 9A009A00  NOP                             BGM          80.    nS
      =FFFFFFF2           NOP                             BGM
+010   FFFFFFF4 9A009A00  NOP                             BGM          80.    nS
      =FFFFFFF6           NOP                             BGM
+011   00006300 0300A3E0  MOVEA   0x0300,R0,R31           BGM          80.    nS

STATUS:   N70732--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 "BGM"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
+021   00007A10 6C019A00  NOP                             BGM          80.    nS
      =00007A12           BRKRET  0x01                    BGM
+022   00007A14 00000000  MOV     R0,R0                   BGM          80.    nS
      =00007A16           MOV     R0,R0                   BGM
+023   00007A18 00000000  MOV     R0,R0                   BGM          80.    nS
      =00007A1A           MOV     R0,R0                   BGM
+024   00000000 00008000    00008000 data read word       BGM         360     nS
+025   00000004 00006380    00006380 data read word       BGM          80.    nS
+026   00006380 00F8C3FD  LD.B    0x00F8[R29],R31                      280     nS
+027   00006384 94984FE0  CMP     0x00,R31                              80.    nS
      =00006386           JNZ/JNE 0x0000641E
+028   00006388 0030DC1D  ST.W    R0,0x0030[R29]                        80.    nS
+029   000060F8 0030DC01    ......01 data read byte                    80.    nS
+030   0000638C 0034DC3D  ST.W    R1,0x0034[R29]                        80.    nS
+031   0000641C DC1D0010    DC1D0010 unused fetch                     200     nS

STATUS:   N70732--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 __start <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 demo program:

run from transfer_address <RETURN>

```
Trace List                Offset=0
Label:  Address   Data                Opcode or Status              time count
Base:     hex     hex                     mnemonic                   relative
-007    00007A0C 00A4CFBD    00A4CFBD fetch                            80.    nS
-006    00007A10 6C019A00    6C019A00 fetch                            80.    nS
-005    00006020 00000004    ......04 data write byte                  80.    nS
-004    00006038 00000000    00000000 data read word                   80.    nS
-003    000060A4 00000000    00000000 data read word                   80.    nS
-002    00007A14 00000000    00000000 fetch                            80.    nS
-001    00007A18 00000000    00000000 fetch                            80.    nS
about   00000000 00008000    00008000 data read word                  360     nS
+001    00000004 00000000    00000000 data read word                   80.    nS
+002    00000000 0002BC20  MOVHI   0x0002,R0,R1                       320     nS
+003    00000004 042CA061  MOVEA   0x042C,R1,R3                       120     nS
+004    00000008 0000BC20  MOVHI   0x0000,R0,R1                       120     nS
+005    0000000C 0000A0A1  MOVEA   0x0000,R1,R5                       120     nS
+006    00000010 0002BC20  MOVHI   0x0002,R0,R1                       120     nS
+007    00000014 0000A081  MOVEA   0x0000,R1,R4                       120     nS

STATUS:   N70732--Running user program    Emulation trace complete______...R....
display trace


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

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

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 6000h through 7fffh, we can simply trigger on any access to that range.

trace about range `6000h` ***thru*** `7fffh` <RETURN>

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

break <RETURN>

```
Trace List               Offset=0
Label:  Address   Data               Opcode or Status              time count
Base:     hex     hex                   mnemonic                     relative
-007   FFFFFFE8 0200CFA0    0200CFA0 fetch                BGM          80.    nS
-006   FFFFFFEC 9A00181D    9A00181D fetch                BGM          80.    nS
-005   000000AC 0000033E    0000033E data write word      BGM          80.    nS
-004   000000A4 00000000    00000000 data write word      BGM          80.    nS
-003   00000200 00006300    00006300 data read word       BGM          80.    nS
-002   FFFFFFF0 9A009A00    9A009A00 fetch                BGM          80.    nS
-001   FFFFFFF4 9A009A00    9A009A00 fetch                BGM          80.    nS
about  00006300 0300A3E0  MOVEA   0x0300,R0,R31           BGM          80.    nS
+001   00006304 CFFD0BBF  SUB     R31,R29                 BGM          80.    nS
      =00006306           LD.W    0x0004[R29],R31         BGM
+002   00006308 DFFD0004    DFFD0004 fetch                BGM          80.    nS
      =0000630A           ST.W    R31,0x00B0[R29]         BGM
+003   0000630C A3FD00B0    A3FD00B0 fetch                BGM          80.    nS
      =0000630E           MOVEA   0x0380,R29,R31          BGM
+004   00006310 DFFD0380    DFFD0380 fetch                BGM          80.    nS

STATUS:   N70732--Running in monitor      Emulation trace complete______...R....
display trace


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

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

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, use the background monitor.

Notes

B

Using the Format Converter

Introduction

Absolute files generated by NEC K&R-C Compiler, or Green Hills Software C-Compiler can not be loaded into the 70732 emulator directly. Therefore, the 70732 Softkey Interface provides a format converter.

How to use the Converter

The format converter generates HP format files from ELF or COFF format files for 70732.

To execute the converter program, use the following command:

```
$v810cnv [options] <file_name>
```

<file_name> is the name of ELF or COFF format file. The converter program will read the ELF or COFF format file. It will generate the following HP format files:

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

The converter accepts the following options.

-x This option specifies to generate HP format absolute file (with .X suffix).

-l This option specifies to generate HP format linker symbol file (with .L suffix).

-a This option specifies to generate HP format assembler symbol file for all of source module information available in the <file_name>. (with .A suffix).

-A *module* This option specifies to generate HP assembler symbol file specified. This option may appear as many as required. If option -a, described above, is used simultaneously, specifications by this option takes precedence so that assembler symbols for modules specified by this option are generated.

-f *module_list_file* This option specifies to read a list of modules to generate HP OMF assembler symbol files from module_list_file. Assembler symbol files associated to modules listed in module_list_file are generated. No other assembler symbol files are not generated. If option -a is used simultaneously, specifications by this option takes precedence so that assembler symbols for modules listed in module_list_file are generated.

-q Suppress warning messages.

-m *anonymous* This option specifies anonymous source module name used. Default anonymous module name is "anonymous". This module name does not have actual effect, however this should not overlap any of source module names in the input absolute load module. Normally the anonymous module name need not to be altered by this option, unless an anonymous module name overlaps to one of source module name in an input absolute load module file.

-T *symbol_name* This option specifies text pointer symbol name. Default text pointer symbol name is "__tp__TEXT". If you change text pointer symbol name in linker directive file, you must specify text pointer symbol name with this option. This option is effect when you convert ELF format file.

-G *symbol_name* This option specifies global pointer symbol name. Default global pointer symbol name is"__gp__TEXT". If you change global pointer symbol name in linker directive file, you must specify global pointer symbol name with this option. This option is effect when you convert ELF format file.

Note

When you use NEC K&R-C Compiler, you must not specify the following options.
"**-o**"(cc732)
"**-r**", "**-s**"(ld732)

When you use Green Hills Software C-Compiler, you must not specify the following options.
"**-O***"(cc810)
"**-s**", "**-srec**", "**-X**", "**-x**"(lx)

You must specify "**-g**" option when you compile your program

---

Restrictions and Considerations

Listed below are restrictions or considerations present when using the format converter.

You can use the [a-z],[A-Z],[0-9],[_](underscore) characters to indicate symbols. Any other characters will be replaced by "_". Symbols are truncated to 15 characters.

ELF Format File

Global symbols which are defined in assembler source files are included in "anonymous.A".
The symbols which are defined with "EQU" directive in assembler source files are not generated.
You can not define plural text pointer symbol nor global pointer symbol.

COFF Format File

The local symbols which are defined in assembler source files are not generated.

Error/Warning Messages

The following is error/warning messages which are specific to using format converter. The cause of the errors is described.

Error Messages

ELF/COFF format common

object file name is too long:

Cause: This error occurs when you attempt to specify the object file whose name is too long.

Action: You must specify the object file name in 511 characters.

object file format is unknown:

Cause: This error occurs when you attempt to specify unknown format file

Action: You must specify ELF/COFF format file.

can’t read input load module:

Cause: This error occures when the object file is not found.

fine unexpected eof in object file:

Cause: This error occures when the converter finds unexpected end of file in the object file.

can’t open input load module

Cause: This error occures when the converter can not open object file.

memory space over lapped

Cause: This error occures when you attempt to specify the object file whose memory space is over lapped.

Action: You must link correctly not to over lap.

can’t open module list file

Cause: This error occurs when you attempt to specify module list file that does not exist with "-f" option.

can’t allocate memory

Cause: Memory is short

can’t open for output

Cause: This error occurs when the converter can’t open output file.

can’t close file

Cause:	This error occurs when the converter can’t close file.

can’t write to file

Cause:	The converter can’t write to output file.

pathname too long

Cause:	This error occurs when you attempt to specify the pathname whose length is to long.
Action:	You must specify pathname in 511 characters.

no valid filename in

Cause:	This error occurs when the converter can’t write source file name in output file. This occurs when source files or directory which includes source files do not exist.

can’t get current directory

Cause:	This error occurs the other process remove the current directory when the converter running.

ELF Format

object file is illegal ELF format

Cause:	This error occurs when you attempt to specify the illegal ELF format file.

can’t find ELF header

Cause:	This error occurs when you attempt to specify the object file that does not have ELF header.

can’t find program header

Cause: This error occurs when you attempt to specify the object file that does not have program header.

can’t find section header

Cause: This error occurs when you attempt to specify the object file that does not have section header.

can’t find section header string table

Cause: This error occurs when you attempt to specify the object file that does not have section header string table.

can’t find symbol table

Cause: This error occurs when you attempt to specify the object file that does not have symbol table.

can’t find symbol string table

Cause: This error occurs when you attempt to spcify the object file that does not have symbol string table.

find illegal debug section

Cause: This error occurs when you attempt to specify the object file that has illegal debug section.

object file is not v810’s

Cause: This error occurs when you attempt to specify the object file that is not ELF format for v810.

COFF Format

object file is illegal COFF format file.

Cause:	The error occurs when you attempt to specify the illegal COFF format object file.

can’t find COFF file header

Cause:	This error occurs when you attempt to specify the object file that does not have header.

can’t find COFF option header

Cause:	This error occurs when attempt to specify the object file that does not have option header.

can’t find COFF section header

Cause:	This error occurs when you attempt to specify the object file that does not have section header.

object file is not executable

Cause:	This error occurs when you attempt to specify the relocatable file.
Action:	Link the relocatable file to generate executable file.

Warning Messages

ELF/COFF Format Common

symbol ’sym1’ truncated and converted to ’sym2’
symbol ’sym1’ truncated to ’sym2’
symbol ’sym1’ converted to ’sym2’

Cause: This warning occurs when symbol name’s length longer than 15 character or/and symbol name includes illegal character. ELF Format

can’t find Debug Table Section
can’t find Line number Table Section

Cause: This warning occurs when you attempt to specify the ELF format file that does not have Debug information

Action: When you link object files, you should specify "-s" option

can’t find TP symbol : ’*sym*’

Cause: This warning occurs when you does not attempt to specify text pointer symbol with "-T" option though you define "*sym*" as text pointer symbol in linker directive file. Or, you define plural text pointer symbols.

Action: When you convert object file, you should specify text pointer symbol with "-T" option.

can’t find GP symbol : *’sym’*

Cause: This warning occurs when you does not attempt to specify global pointer symbol with "-G" option though you define "*sym*" as global pointer symbol in linker directive file. Or, you define plural global pointer symbols.

Action: When you convert object file, you should specify global pointer symbol with "-G" option.

COFF Format

line number stripped from the file.
local symbol stripped from the file.

Cause: This warning occurs when you link files with option that strip the line number information or/and local symbols information.

Action: When you link files, you should not specify the option that strip the line number information or/and local symbols information.

object file has no symbol table object file has no global symbol

Cause: This warning occurs when you generate the object file from assemble source files only, or you link with option that strip all symbols information.

Index

Notes