HP 64768

70433 Emulator Terminal Interface

User's Guide

HP Part No. 64768-97004
Printed in Japan
May 1995

Edition 3

Notice

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

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

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

Printing History

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

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

Edition 1	64768-97000, March 1993
Edition 2	64768-97002, October 1993
Edition 3	**64768-97004, May 1995**

Using this Manual

This manual will show you how to use HP 64768 emulators with the Terminal Interface.

This manual will:

- Show you how to use emulation commands by executing them on a sample program and describing their results.
- Show 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.
- Show you how to use the emulator in-circuit (connected to a target system).
- Describe the command syntax which is specific to the 64768 emulator.

This manual will not:

- Describe every available option to the emulation commands; this is done in the HP 64700 Emulators Terminal Interface: User's Reference.

For the most part, the HP 647680A and HP 64768B emulators all operate the same way. Differences of between the emulators are described where they exist. Both the HP 64768A and HP 64768B emulators will be referred to as the "HP 64768 emulator" or "70433 emulator".

Organization

Chapter 1 Introduction to the 64768 Emulator. This chapter briefly introduces you to the concept of emulation and lists the basic features of the 64768 emulator.

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

Chapter 3 Emulation Topics. This chapter shows you how to: restrict the emulator to real-time execution, use the analyzer trigger to cause breaks, and run the emulator from target system reset.

Chapter 4 In-Circuit Emulation Topics. This chapter shows you how to: install the emulator probe into a target system, select a target system clock source, allow the target system to insert wait states, and use the features which allow you to debug target system ROM.

Appendix A 64768 Emulator Specific Command Syntax. This appendix describes the command syntax which is specific to the 64768 emulator. Included are: emulator configuration items, address syntax, display and access modes.

Appendix B Using the Optional Foreground Monitor. This appendix describes how to use the foreground monitor.

Contents

Illustrations

Tables

Notes

1

Introduction to the 64768 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 64768 emulator is designed to replace the 70433 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.

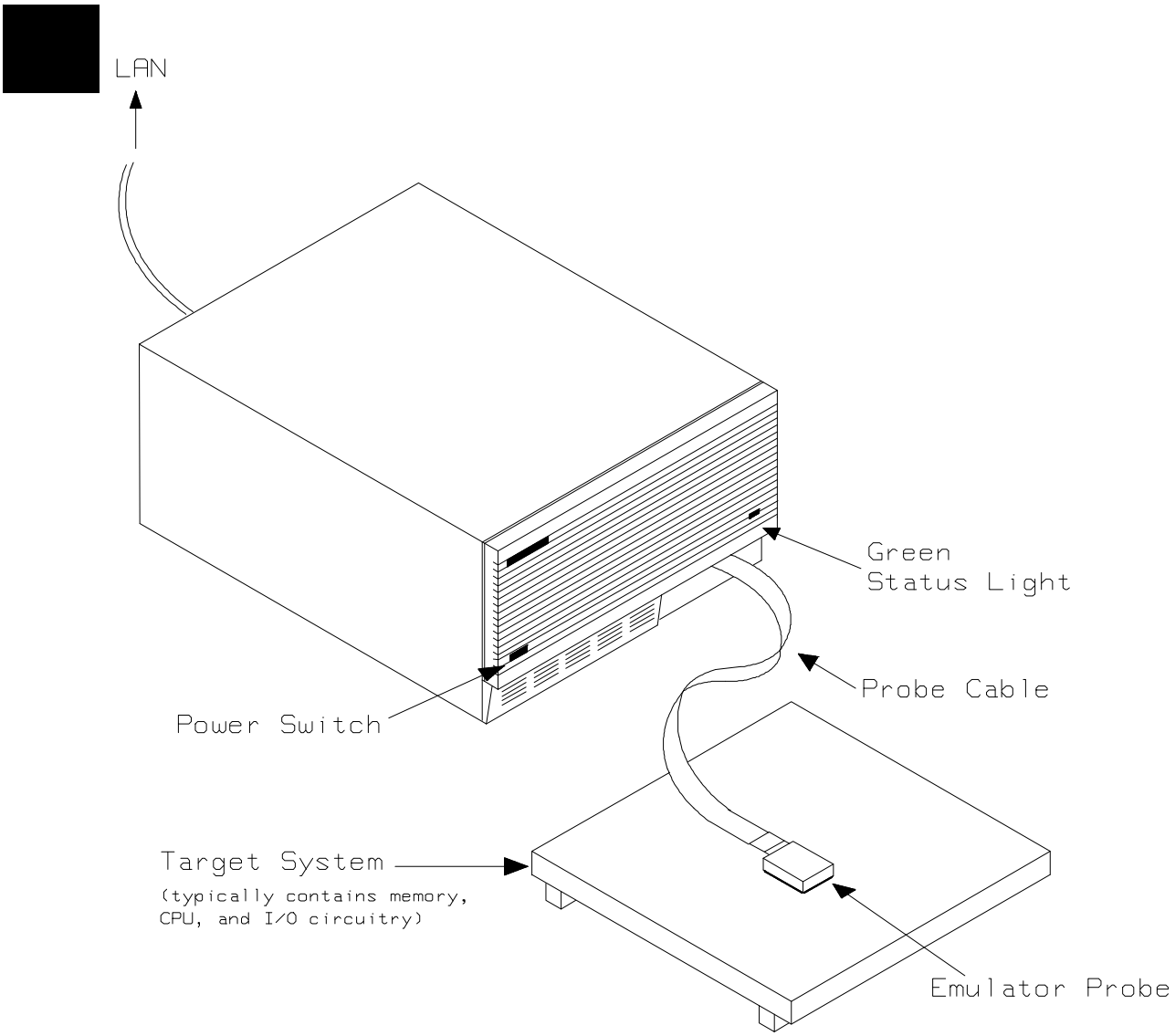


Figure 1-1 HP 64768 Emulator for uPD70433

1-2 Introduction

Features of the HP 64768 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 132-pin PGA type of 70433 microprocessor is supported.The HP 64768 emulator probe has a 132-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 HP 64768A emulator runs with an internal clock speed of 12.5MHz (system clock), or with target system clocks from 4 to 25 MHz.

The HP 64768B emulator runs with an internal clock speed of 12.5MHz (system clock), or with target system clocks from 4 to 32 MHz.

Emulation memory

The HP 64768 emulator is used with one of the following Emulation Memory Cards.

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

You can define up to 16 memory ranges (at 256 byte boundaries and at least 256 byte in length). The monitor occupies 2K bytes leaving 126K,510K,1022K bytes of emulation memory which you may use.You can characterize memory ranges as emulation RAM, emulation ROM, target system RAM, target system ROM, or guarded memory. The emulator generates an error message when accesses are made to guarded memory locations. You can also configure the emulator so that writes to memory defined as ROM cause emulator execution to break out of target program execution.

Analysis

The HP 64768 emulator is used with one of the following analyzers which allows you to trace code execution and processor activity.

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

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

Registers

You can display or modify the 70433 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 70433 BRK 3 instruction to provide software breakpoint. When you define a software breakpoint, the emulator places a BRK 3 instruction at the specified address; after the BRK 3 instruction causes emulator execution to break out of your program, the emulator replaces BRK 3 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 70433 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.)

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

Easy Products Upgrades

Because the HP 64700 Series development tools (emulator, analyzer, LAN board) contain programmable parts, it is possible to reprogram the firmware and some of the hardware without disassembling the HP 64700A 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, Hold Request While in Background Monitor

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

User Interrupts While in Background Monitor

If you use the background monitor, NMI and INTP0-5 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.

Unbreaking into the Monitor

The emulator can not break into the monitor if the microprocessor is in hold state. The emulator will break into the monitor after hold state because break request is suspended.

The emulator can not break into the monitor if the microprocessor is in reset state by RESET signal from target system.

CLKOUT enable bit

CLKOUT signal can be enabled/disabled by ENCLK bit, which is bit 5 of PRC register. You must not clear ENCLK bit(ENCLK bit is "1" in

reset). The emulator will not work properly if CLKOUT signal is disabled

DMA Support

Direct memory access to emulation memory by external DMA controller is not permitted.

Accessing SFR

When you access SFR(Special Function Registers), you must use reg commands. If you access SFR with m commands, you will access to the actual memory you mapped(as target system ROM or RAM, emulation ROM or RAM).

Accessing Reserved Area of I/O Space

When you access reserved area of I/O space(0FF80h-0FFFFh) with "io" commands in the background monitor, the emulator operates exceptionally. When you display reserved area of I/O space, theemulator displays "FFh" . When you modify reserved area of I/Ospace, the emulator does not modify value.

Evaluation Chip

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

Notes

2

Getting Started

Introduction

This chapter will lead you through a basic, step by step tutorial that shows how to use the HP 64768 emulator for the 70433 microprocessor.

This chapter will:

- Describe the sample program used for this chapter's examples.
- Show you how to use the "help" facility.
- Show you how to use the memory mapper.
- Show you how to enter emulation commands to view execution of the sample program. The commands described in this chapter include:
  - Displaying and modifying memory
  - Stepping
  - Displaying registers
  - Defining macros
  - Searching memory
  - Running
  - Breaking
  - Using software breakpoints
  - Copying memory
  - Testing coverage

Before You Begin

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

1. Completed hardware installation of the HP 64700 emulator in the configuration you intend to use for your work:
   - Standalone configuration
   - Transparent configuration
   - Remote configuration
   - Local Area Network configuration

   References: *HP 64700 Series Installation/Service* manual

2. If you are using the Remote configuration, you must have completed installation and configuration of a terminal emulator program which will allow your host to act as a terminal connected to the emulator. In addition, you must start the terminal emulator program before you can work the examples in this chapter.

3. If you have properly completed steps 1 and 2 above, you should be able to hit <RETURN> (or <ENTER> on some keyboards) and get one of the following command prompts on your terminal screen:

```
U>
R>
M>
```

   If you do not see one of these command prompts, retrace your steps through the hardware and software installation procedures outlined in the manuals above, verifying all connections and procedural steps.

   In any case, you must have a command prompt on your terminal screen before proceeding with the tutorial.

A Look at the Sample Program

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

```
NAME    cmd_rds

PUBLIC  Msgs,Init,Cmd_Input,Msg_Dest

COMN    SEGMENT PARA COMMON 'COMN'
;*****************************************************
;  Command input byte.
;*****************************************************
Cmd_Input       DB      ?
;*****************************************************
;  Destination of the command message.
;*****************************************************
Msg_Dest        DB      20H DUP (?)
                EVEN
                DW      6FH DUP (?)    ; Stack area.
Stk             LABEL   WORD
COMN    ENDS

DATA    SEGMENT PARA PUBLIC 'DATA'
Msgs            LABEL   BYTE
Msg_A           DB      "Command A entered "
Msg_B           DB      "Command B entered "
Msg_I           DB      "Invalid Command   "
End_Msgs        LABEL   BYTE
DATA    ENDS

CODE    SEGMENT PARA PUBLIC 'CODE'
                ASSUME  PS:CODE,DS0:DATA,DS1:COMN,SS:COMN
;*****************************************************
;  The following instructions initialize segment
;  regsiters and set up the stack pointer.
;*****************************************************
Init:           MOV     AW,DATA
                MOV     DS0,AW
                MOV     AW,COMN
                MOV     DS1,AW
                MOV     SS,AW
                MOV     SP,OFFSET Stk
;*****************************************************
;  Clear previous command
;*****************************************************
Read_Cmd:       MOV     Cmd_Input,0
                NOP
;*****************************************************
;  Read command input byte.  If no command has been
;  entered, continue to scan for command input.
;*****************************************************
Scan:           MOV     AL,Cmd_Input
                CMP     AL,0
                BE      Scan
;*****************************************************
;  A command has been entered.  Check if it is
;  command A, command B, or invalid.
;*****************************************************
```

Figure 2-1 Sample program listing

```
Exe_Cmd:        CMP     AL,41H
                BE      Cmd_A
                CMP     AL,42H
                BE      Cmd_B
                BR      Cmd_I
;*****************************************************
;
;  Command A is entered.  CW = the number of bytes in
;  message A.  BP = location of the message.  Jump to
;  the routine which writes the message.
;*****************************************************
;
Cmd_A:          MOV     CW,Msg_B-Msg_A
                MOV     IX,OFFSET Msg_A
                BR      Write_Msg
;*****************************************************
;
;  Command B is entered.
;*****************************************************
;
Cmd_B:          MOV     CW,Msg_I-Msg_B
                MOV     IX,OFFSET Msg_B
                BR      Write_Msg
;*****************************************************
;
;  An invalid command is entered.
;*****************************************************
;
Cmd_I:          MOV     CW,End_Msgs-Msg_I
                MOV     IX,OFFSET Msg_I
;*****************************************************
;
;  Message is written to the destination.
;*****************************************************
;
Write_Msg:      MOV     IY,OFFSET Msg_Dest
                REP MOVBK       Msg_Dest,Msgs
;*****************************************************
;
;  The rest of the destination area is filled
;  with zeros.
;*****************************************************
;
Fill_Dest:      XOR     AL,AL
                MOV     CW,OFFSET Msg_Dest+20H
                SUB     CW,IY
                REP STM Msg_Dest
;*****************************************************
;
;  Go back and scan for next command
;*****************************************************
;
                BR      Read_Cmd
CODE    ENDS
                END     Init
```

Figure 2-1 Sample program (Cont'd)

Data Declarations

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

Initialization

The program instructions from the **Init** label to the **Read_Cmd** label perform initialization. The segment registers are loaded and the stack pointer is set up.

Reading Input

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

Processing Commands

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

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

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

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

The instructions at **Cmd_A**, **Cmd_B**, and **Cmd_I** load register CW with the length of the message to be displayed and register IX with the starting location of the appropriate message. Then, execution transfers to **Write_Msg** where the appropriate message is written to the destination location, **Msg_Dest**.

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

The Destination Area

The area at COMN segment declares memory storage for the command input byte, the destination area, and the stack area.

Using the "help" Facility

The HP 64700 Series emulator's Terminal Interface provides an excellent help facility to provide you with quick information about the various commands and their options. From any system prompt, you can enter "**help**" or "**?**" as shown below.

R> **help**

```
help  - display help information

  help <group>         - print help for desired group
  help -s <group>      - print short help for desired group
  help <command>       - print help for desired command
  help                 - print this help screen

--- VALID <group> NAMES ---
  gram     - system grammar
  proc     - processor specific grammar

  sys      - system commands
  emul     - emulation commands
  trc      - analyzer trace commands
  *        - all command groups
```

Commands are grouped into various classes. To see the commands grouped into a particular class, you can use the help command with that group. Viewing the group help information in short form will cause the commands or the grammar to be listed without any description.

For example, if you want to get some information for group gram, enter "**help gram**". Following help information should be displayed.

R> **help gram**

```
gram - system grammar
------------------------------------------------------------------------------
--- SPECIAL CHARACTERS ---
  #  - comment delimiter    ;  - command separator    Ctl C - abort signal
  {} - command grouping     "" - ascii string         ''    - ascii string
  Ctl R - command recall    Ctl B - recall backwards

--- EXPRESSION EVALUATOR ---
  number bases:  t-ten   y-binary   q-octal   o-octal   h-hex
  repetition and time counts default to decimal - all else default to hex
  operators:     ()  ~  *  /  %  +  -  <<  <<<  >>  >>>  &  ^  |  &&

--- PARAMETER SUBSTITUTION ---
  &token& - pseudo-parameter included in macro definition
          - cannot contain any white space between & pairs
          - performs positional substitution when macro is invoked
  Example
     Macro definition:  mac getfile={load -hbs"transfer -t &file&"}
     Macro invocation:  getfile MYFILE.o
     Expanded command:  load -hbs"transfer -t MYFILE.o"
```

Help information exists for each command. Additionally, there is help information for each of the emulator configuration items.

Becoming Familiar with the System Prompts

A number of prompts are used by the HP 64700 Series emulators. Each of them has a different meaning, and contains information about the status of the emulator before and after the commands execute. These prompts may seem cryptic at first, but there are two ways you can find out what a certain prompt means.

Using "help proc" to View Prompt Description

The first way you can find information on the various system prompts is to look at the **proc** help text.

R> **help proc**

```
--- Address format -----
32 bit (seg:off) logical, extended 32 bit (seg::off) logical,
24 physical, or 9 bit physical (@iram)

--- Emulation Prompt Status Characters ---
  U - running user code      M - running in monitor
  c - slow clock             r - target reset
  R - emulation reset        s - stop
  h - halt                   b - slow bus cycle
  g - hold                   T - awaiting target reset
  W - awaiting CMB ready     ? - unknown state


--- Analyzer STATUS Field Equates ---
  exec   - execute instruction    dma     - DMA memory access
  fetch  - program fetch          int     - interrupt acknowledge
  read   - read                   refresh - refresh cycle
  write  - write                  halt    - halt
  mem    - memory access          hold    - hold acknowledge
  io     - I/O accesscpu          stop    - stop
  sfr    - SFR access             wrrom   - write to rom
  cpu    - cpu cycle              grd     - guarded memory access
  ms     - macro service          fg      - foreground
  memio  - mem  io dma            bg      - background
  memsfr - mem  SFR dma
```

Using the Emulation Status Command (es) for Description of Current Prompt

When using the emulator, you will notice that the prompt changes after entering certain commands. If you are not familiar with a new prompt and would like information about that prompt only, enter the **es** (emulation status) command for more information about the current status.

U> **es**

```
N70433--Running user program
```

Initializing the Emulator

If you plan to follow this tutorial by entering commands on your emulator as shown in this chapter, verify that no one else is using the emulator. To initialize the emulator, enter the following command:

R> **init**

```
# Limited initialization completed
```

The **init** command with no options causes a limited initialization, also known as a warm start initialization. Warm start initialization does not

affect system configuration. However, the **init** command will reset emulator and analyzer configurations. The **init** command:

- Resets the memory map.
- Resets the emulator configuration items.
- Resets the break conditions.
- Clears software breakpoints.

The **init** command does not:

- Clear any macros.
- Clear any emulation memory locations; mapper terms are deleted, but if you respecify the same mapper terms, you will find that the emulation memory contents are the same.

Other Types of Initialization

There are two options to **init**. The **-p** option specifies a powerup initialization, also known as a cold start initialization. It initializes the emulator, analyzer, system controller, and communications port; additionally, performance verification tests are run.

The **-c** option also specifies a cold start initialization, except that performance verification tests are not run.

Set Up the Proper Emulation Configuration

Emulation configuration is needed to adapting to your specific development. As you have initialized the emulator, the emulation configuration items have default value.

Set Up Emulation Condition

The emulator allows you to set the emulator's configuration setting with the **cf** command. Enter the **help cf** to view the information with the configuration command.

R> **help cf**

```
cf - display or set emulation configuration

    cf                - display current settings for all config items
    cf <item>         - display current setting for specified <item>
    cf <item>=<value> - set new <value> for specified <item>
    cf <item> <item>=<value> <item> - set and display can be combined

  help cf <item>      - display long help for specified <item>

   --- VALID CONFIGURATION          NAMES ---
     clk      - select clock source
     dsize    - select data bus width
     hold     - en/dis HLDRQ input from the target system
     mne      - select mnemonic for inverse assembly
     mon      - select foreground or background monitor
     nmi      - en/dis NMI from the target system
     rad      - select run address translation method
     rdy      - relationship between emulator and target ready
     rrt      - en/dis restriction to real time runs
     rsp      - specify stack pointer after emulation reset
     rst      - en/dis RESET input from the target system
     tdma     - en/dis tracing DMA cycles
     trfsh    - en/dis tracing of refresh cycles
```

To view the current emulator configuration setting, enter the following command.

R> **cf**

```
cf clk=int
cf dsize=16
cf hold=en
cf mne=70433
cf mon=bg
cf nmi=en
cf rad=minseg
cf rdy=lk
cf rrt=dis
cf rsp=0000:8000
cf rst=en
cf tdma=en
cf trfsh=en
```

The individual configuration items won't be explained in this section; refer to the "CONFIG_ITEMS" in the "64768 Emulator Specific Command Syntax" appendix for details

Set Up Access/Display Modes

To avoid problems later while modifying and displaying memory locations, enter the following command:

```
R> mo -ab -db
```

This sets the access and display modes for memory operation to byte.(if they are left at the default mode of word,the memory modification and display examples will not function correctly.)

Mapping Memory

Depending on the memory board, emulation memory consists of 128K , 512K or 1M bytes, mappable in 256 byte blocks. The monitor occupies 2K bytes, leaving 126K , 510K or 1022K bytes of emulation memory which you may use. The emulation memory system does not introduce wait states.

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

Note

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

Blocks of memory can also be characterized as guarded memory. Guarded memory accesses will generate "break to monitor" requests. Writes to ROM will also generate "break to monitor" requests if the **rom** break condition is enabled. Memory is mapped with the **map** command. To view the memory mapping options, enter:

M> **help map**

```
map - display or modify the processor memory map
  map                          - display the current map structure
  map <addr>..<addr> <type>    - define address range as memory type
  map other <type>             - define all other ranges as memory type
  map -d <term#>               - delete specified map term
  map -d *                     - delete all map terms
--- VALID <type> OPTIONS ---
  eram  -  emulation ram
  erom  -  emulation rom
  tram  -  target ram
  trom  -  target rom
  grd   -  guarded memory
```

Enter the **map** command with no options to view the default map structure.

M> **map**

```
# remaining number of terms   : 16
# remaining emulation memory  : 1f8000h bytes
map  other tram
```

Which Memory Locations Should be Mapped?

Typically, assemblers generate relocatable files and linkers combine relocatable files to form the absolute file. A linker load map listing will show what memory locations your program will occupy. One for the sample program is shown below.

```
Hewlett-Packard ldv20  Tue Jul  2 13:18:08 1991

HPB1445-19300 A.03.00 09Oct91  Copr. HP 1990
Command line: ldv20  -c cmd_rds.k -Lh

NAME cmd_rds
SEG /COMN=400h
SEG /DATA=600h
SEG /CODE=800h
LOAD cmd_rds.o
END

OUTPUT MODULE NAME:   cmd_rds
OUTPUT MODULE FORMAT: HP64000 absolute

MODULE SUMMARY
--------------
```

```
MODULE    SEGMENT   CLASS     HP SECTION  START  END

cmd_rds   /usr/hp64000/demo/emul/hp64768/cmd_rds.o
          CODE      CODE      PROG        00800  00859
          COMN      COMN      DATA        00400  004FF
          DATA      DATA      COMMON      00600  00635

SEGMENT SUMMARY
---------------

SEGMENT   CLASS     GROUP     START  END    LENGTH  ALIGNMENT     COMBINE

COMN      COMN                00400  004FF  00100   Paragraph     Common
DATA      DATA                00600  00635  00036   Paragraph     Public
CODE      CODE                00800  00859  0005A   Paragraph     Public
??SEG                         00000  00000  00000   Paragraph     Public
??DATA1   ??INIT              00000  00002  00003   Byte          Common

START ADDRESS:   00080:00000 - 00800
Load completed.
```

From the load map listing, you can see that the sample program occupies three address ranges. The program area, which contains the opcodes and operands, occupies locations 800 through 859 hex. The data area, which contains the ASCII values of the messages the program transfers, occupies locations 600 through 635 hex. The destination area, which contains the command input byte and the locations of the message destination, occupies locations 400 through 4FF hex.

Since the program writes to the destination area, the mapper block of destination area should not be characterized as ROM. Enter the following commands to map memory for the sample program and display the memory map.

```
R> map 0..4ff eram
R> map 600..9ff erom
R> map
```

```
# remaining number of terms   : 14
# remaining emulation memory  : 1ef00h bytes
map  0000000..00004ff  eram    # term 1
map  0000600..00009ff  erom    # term 2
map  other tram
```

When mapping memory for your target system programs, you should characterize emulation memory locations containing programs and constants (locations which should not be written) as ROM. This will prevent programs and constants from being written over accidentally. Break will occur when instructions or commands attempt to do so (if the **rom** break condition is enabled).

Getting the Sample Program into Emulation Memory

This section assumes you are using the emulator in one of the following three configurations:

1. Connected only to a terminal, which is called the standalone configuration. In the standalone configuration, you must modify memory to load the sample program.

2. Connected between a terminal and a host computer, which is called the transparent configuration. In the transparent configuration, you can load the sample program by downloading from the "other" port.

3. Connected to a host computer and accessed via a terminal emulation program (for example, the terminal window of the PC Interface). This configurations is called remote configurations. In the remote configuration, you can load the sample program by downloading from the same port.

Standalone Configuration

If you are operating the emulator in the standalone configuration, the only way to load the sample program into emulation memory is by modifying emulation memory locations with the **m** (memory display/modification) command.

You can enter the sample program into memory with the **m** command as shown below.

```
R> m -db 800=0b8,60,0,8e,0d8,0b8,40,0,8e,0c0,8e,0d0,0bc,0,1,26
R> m -db 810=0c6,6,0,0,0,26,0a0,0,0,3c,0,74,0f8,3c,41,74
R> m -db 820=0d,90,90,3c,42,74,12,90,90,0eb,19,90,90,90,0b9,12
R> m -db  830=0,0be,0,0,0eb,14,90,90,90,0b9,12,0,0be,12,0,0eb
R> m -db 840=09,90,90,90,0b9,12,0,0be,24,0,0bf,1,0,0f3,0a4,32
R> m -db 850=0c0,0b9,21,0,2b,0cf,0f3,0aa,0eb,0b5
R> m  -db 600="Command A entered Command B entered Invalid command   "
```

After entering the opcodes and operands, you would typically display memory in mnemonic format to verify that the values entered are correct (see the example below). If any errors exist, you can modify individual locations. Also, you can use the **cp** (copy memory)

command if, for example, a byte has been left out, but the locations which follow are correct.

Note

Be careful about using this method to enter programs from the listings of relocatable source files. If source files appear in relocatable sections, the address values of references to locations in other relocatable sections are not resolved until link-time. The correct values of these address operands will not appear in the assembler listing.

Transparent Configuration

If your emulator is connected between a terminal and a host computer, you can download programs into memory using the **load** command with the **-o** (from other port) option. The **load** command will accept absolute files in the following formats:

- HP absolute.
- Intel hexadecimal.
- Tektronix hexadecimal.
- Motorola S-records.

The examples which follow will show you the methods used to download HP absolute files and the other types of absolute files.

HP Absolutes

Downloading HP format absolute files requires the **transfer** protocol. The example below assumes that the **transfer** utility has been installed on the host computer (HP 64884 for HP 9000 Series 500, or HP 64885 for HP 9000 Series 300).

Note

Notice that the transfer command on the host computer is terminated with the <ESCAPE>**g** characters; by default, these are the characters which temporarily suspend the transparent mode to allow the emulator to receive data or commands.

R> **load -hbo** <RETURN> <RETURN>
$ **transfer -rtb cmd_rds.X** <ESCAPE>g

```
 ####
R>
```

Other Supported Absolute Files

The example which follows shows how to download Intel hexadecimal files by the same method (but different **load** options) can be used by load Tektronix hexadecimal and Motorola S-record files as well.

R> **load -io** <RETURN> <RETURN>
$ **cat ihexfile <ESCAPE>g**

```
 #####
Data records = 00003  Checksum error = 00000
R>
```

Remote Configuration

If the emulator is connected to a host computer, and you are accessing the emulator from the host computer via a terminal emulation program, you can also download files with the **load** command. However, in the remote configuration, files are loaded from the same port that commands are entered from. For example, if you wish to download a Tektronix hexadecimal file from a Vectra personal computer, you would enter the following commands.

R> **load -t** <RETURN>

After you have entered the **load** command, exit from the terminal emulation program to the MS-DOS operating system. Then, copy your hexadecimal file to the port connected to the emulator, for example:

C:\copy thexfile com1: <RETURN>

Now you can return to the terminal emulation program and verify that the file was loaded correctly.

For More Information

For more information on downloading absolute files, refer to the **load** command description in the *HP 64700 Emulators Terminal Interface: User's Reference* manual.

Loading an ASCII Symbol File

The 64768 emulator supports the use of symbolic references in the terminal interface. The symbols can be loaded with a program file, as is the case with Intel OMF files . They can also be loaded from an ASCII text file on a host system.

The symbols used are defined in a file using a text editor, or any other means to create the file. Refer to the *HP 64700-Series Emulators Terminal Interface Reference* for information on the format of the file. The file is then transferred to the emulator using the **load** command.

You can create a text file named "cmd_rds.SYM" on your HP-UX host system. The file will look something like as follows.

```
#
cmd_rds:
Init 800
Read_Cmd 80f
Scan 815
Exe_Cmd 81d
Cmd_A 82e
Cmd_B 839
Cmd_I 844
Write_Msg 84a
Fill_Dest 84f
#
```

Use the "-S" option on the **load** command to transfer the file.

```
R> load -Sos "cat cmd_rds.SYM"
```

The symbols can then be manipulated with the "sym" command, and used in commands at the command line. If the load is not successful, the nature of the error will be reported.

R> **sym**

```
sym cmd_rds:Init=000800
sym cmd_rds:Read_Cmd=00080f
sym cmd_rds:Scan=000815
sym cmd_rds:Exe_Cmd=00081d
sym cmd_rds:Cmd_A=00082e
sym cmd_rds:Cmd_B=000839
sym cmd_rds:Cmd_I=000844
sym cmd_rds:Write_Msg=00084a
sym cmd_rds:Fill_Dest=00084f
```

Refer to the "Execution Topics" section in the "Emulation Topics" chapter and the "ADDRESS" section in the "64768 Emulator Specific Command Syntax" appendix.

Displaying Memory In Mnemonic Format

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

R> **m -dm 800..858**

```
0000800  -              MOV AW,0060
0000803  -              MOV DS0,AW | MOV AW,0040
0000808  -              MOV DS1,AW | MOV SS,AW | MOV SP,010
000080f  -              MOV DS1:BYTE PTR 0000,00
0000815  -              MOV AL,DS1:BYTE PTR 0000
000081a  -              CMP AL,00
000081c  -              BE/Z 00815
000081e  -              CMP AL,41
0000820  -              BE/Z 0082e
0000821  -              NOP
0000822  -              NOP
0000823  -              CMP AL,42
0000825  -              BE/Z 00439
0000827  -              NOP
0000828  -              NOP
0000829  -              BR SHORT 00844
000082b  -              NOP
000082c  -              NOP
000082d  -              NOP
000082e  -              MOV CW,0012
0000831  -              MOV IX,0000
0000834  -              BR SHORT 0084a
0000836  -              NOP
0000837  -              NOP
0000838  -              NOP
0000839  -              MOV CW,0012
000083c  -              MOV IX,0012
000083f  -              BR SHORT 0084a
0000841  -              NOP
0000842  -              NOP
```

```
0000843  -               NOP
0000844  -               MOV CW,0012
0000847  -               MOV IX,0024
000084a  -               MOV IY,0001
000084d  -               REP/E/Z MOVBKB
000084f  -               XOR AL,AL
0000851  -               MOV CW,0021
0000854  -               SUB CW,IY
0000856  -               REP/E/Z STMB
0000858  -               BR SHORT 0080f
```

If you display memory in mnemonic format and do not recognize the instructions listed or see some illegal instructions or opcodes, go back and make sure the memory locations you have typed are mapped properly. If the memory map is not the problem, recheck the linker load map listing to verify that the absolute addresses of the program match with the locations you are trying to display.

If you have loaded symbols with the sample program, the display will include the symbols in the memory display.

```
0000800  cmd_rds:Init     MOV AW,0060
0000803  -                MOV DS0,AW | MOV AW,0040
0000808  -                MOV DS1,AW | MOV SS,AW | MOV SP,010
000080f  md_rds:Read_Cmd  MOV DS1:BYTE PTR 0000,00
0000815  cmd_rds:Scan     MOV AL,DS1:BYTE PTR 0000
0000819  -                CMP AL,00
000081b  -                BE/Z cmd_Rds:Scan
000081d  cmd_rds:Exe_Cmd  CMP AL,41
000081f  -                BE/Z cmd_rds:Cmd_A
0000821  -                NOP
0000822  -                NOP
0000823  -                CMP AL,42
0000825  -                BE/Z cmd_rds:Cmd_B
0000827  -                NOP
0000828  -                NOP
0000829  -                BR SHORT cmd_Rds:Cmd_I
000082b  -                NOP
000082c  -                NOP
000082d  -                NOP
000082e  cmd_rds:Cmd_A    MOV CW,0012
0000831  -                MOV IX,0000
0000834  -                BR SHORT cmd_rds:Write_Msg
0000836  -                NOP
0000837  -                NOP
0000838  -                NOP
0000839  cmd_rds:Cmd_B    MOV CW,0012
000083c  -                MOV IX,0012
000083f  -                BR SHORT cmd_rds;Write_Msg
0000841  -                NOP
0000842  -                NOP
0000843  -                NOP
0000844  cmd_rds:Cmd_I    MOV CW,0012
0000847  -                MOV IX,0024
000084a  d_rds:Write_Msg  MOV IY,0001
000084d  -                REP/E/Z MOVBKB
```

```
000084f  d_rds:Fill_Dest  XOR AL,AL
0000851  -                MOV CW,0021
0000854  -                SUB CW,IY
0000856  -                REP/E/Z STMB
0000858  -                BR SHORT cmd_rds:Read_Cmd
```

Note

The command processor retains the name of the last module referenced. If a symbol does not contain a module name, the list of global symbols is searched. If the symbol is not found, the list of user symbols is searched. If the symbol is still not found, the system searches the last module referenced. If it doesn't find it there, the rest of the modules are searched.

Stepping Through the Program

The emulator allows you to execute one instruction or a number of instructions with the s (step) command. Enter the **help s** to view the options available with the step command.

R> **help s**

```
s - step emulation processor

  s                     - step one from current PC
  s <count>             - step <count> from current PC
  s <count> $           - step <count> from current PC
  s <count> <addr>      - step <count> from <addr>
  s -q <count> <addr>   - step <count> from <addr>, quiet mode
  s -w <count> <addr>   - step <count> from <addr>, whisper mode

--- NOTES ---
  STEPCOUNT MUST BE SPECIFIED IF ADDRESS IS SPECIFIED!
  If <addr> is not specified, default is to step from current PC.
  A <count> of 0 implies step forever.
```

A step count of 0 will cause the stepping to continue "forever" (until some break condition, such as "write to ROM", is encountered, or until you enter <CTRL>c). The following command will step from the first address of the sample program.

R> **s 1 0:800**

```
00000:00800  cmd_rds:Init     MOV AW,0060
PC = 00000:00803
```

Note

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

1) Instructions that manipulate for sreg or xreg: MOV sreg,reg16, MOV sreg,mem16, POP sreg, MOV xsreg,reg16, MOV xsreg,mem16, POP xsreg.
2) Prefix instructions: PS:, SS:, DS0:, DS1:, DS2:, DS3:, IRAM, REPC, REPNC, REP, REPE, REPZ, REPNE, REPNZ, BUSLOCK.
3) EI, DI, RETI, RETRBI.
4) POP PSW.
5) FINT.
6) BRKCS, BRK 3, BRK imm8, BRKV(V=1), CHKIND(mem32>reg16 or (mem32+2)<reg16), FPO1 fp-op; FPO1 fp-op,mem;, FPO2 fp-op;, FPO2 fp-op,mem;, TSKSW, MOVSPB, MOVSPA.
7) RSTWDT.
8) The first instruction of interrupt processing routine.

Displaying Registers

The step command shown above executed the MOV AW,0060H instruction. Enter the following command to view the contents of the registers.

M> **reg ***

```
reg ps=0000 pc=0803 psw=f002 aw=0060 bw=0000 cw=0000 dw=0000 sp=8000 bp=0000
reg ix=0000 iy=0000 ds0=0000 ds1=0000 ds2=0000 ds3=0000 ss=0000
```

The register contents are displayed in a "register modify" command format. This allows you to save the output of the **reg** command to a command file which may later be used to restore the register contents.

(Refer to the **po** (port options) command description in the *erminal Interface: User's Reference* for more information on command files.)

Refer to the "REGISTER CLASS and NAME" section in the "64768 Emulator Specific Command Syntax" appendix for more information on the register names and classes.

Combining Commands

More than one command may be entered in a single command line. The commands must be separated by semicolons (;). For example, you could execute the next instruction(s) and display the registers by entering the following.

M> **s;reg**

```
00000:00803  -                MOV DS0,AW | MOV AW,0040
PC = 00000:00808
reg ps=0000 pc=0808 psw=f002 aw=0040 bw=0000 cw=0000 dw=0000 sp=8000 bp=0000
reg ix=0000 iy=0000 ds0=0060 ds1=0000 ds2=0000 ds3=0000 ss=0040
```

The sample above shows you that MOV DS0,AW and MOV AW,0040H are executed by step command. Refer to the Note above, for the reason why two instructions are executed.

Using Macros

Suppose you want to continue stepping through the program and displaying registers after each step. You could continue entering s commands followed by **reg** commands, but you may find this tiresome. It is easier to use a macro to perform a sequence of commands which will be entered again and again.

Macros allow you to combine and store commands. For example, to define a macro which will display registers after every step, enter the following command.

M> **mac st={s;reg}**

Once the st macro has been defined, you can use it as you would use any other command.

M> **st**

```
# s ; reg
00000:00808  -                MOV DS1,AW | MOV SS,AW | MOV SP,010
PC = 00000:0080f
reg ps=0000 pc=080f psw=f002 aw=0040 bw=0000 cw=0000 dw=0000 sp=0100 bp=0000
reg ix=0000 iy=0000 ds0=0060 ds1=0040 ds2=0000 ds3=0000 ss=0040
```

Command Recall

The command recall feature is yet another, easier way to enter commands again and again. You can press <CTRL>**r** to recall the commands which have just been entered. If you go past the command of interest, you can press <CTRL>**b** to move forward through the list of saved commands. To continue stepping through the sample program, you could repeatedly press <CTRL>**r** to recall and <RETURN> to execute the **st** macro.

Repeating Commands

The **rep** command is also helpful when entering commands repetitively. You can repeat the execution of macros as well as normal commands. For example, you could enter the following command to cause the **st** macro to be executed four times.

M> **rep 4 st**

```
# s ; reg
00000:0040f                   MOV DS1:0000,00
PC = 00000:00815
reg ps=0000 pc=0815 psw=f002 aw=0040 bw=0000 cw=0000 dw=0000 sp=0100 bp=0000
reg ix=0000 iy=0000 ds0=0060 ds1=0040 ds2=0000 ds3=0000 ss=0040
# s ; reg
00000:00815  -                MOV AL,DS1:BYTE PTR 0000,00
PC = 00000:00819
reg ps=0000 pc=0819 psw=f002 aw=0000 bw=0000 cw=0000 dw=0000 sp=0100 bp=0000
reg ix=0000 iy=0000 ds0=0060 ds1=0040 ds2=0000 ds3=0000 ss=0040
# s ; reg
00000:00819  -                CMP AL,00
PC = 00000:0081b
reg ps=0000 pc=081b psw=f046 aw=0000 bw=0000 cw=0000 dw=0000 sp=0100 bp=0000
reg ix=0000 iy=0000 ds0=0060 ds1=0040 ds2=0000 ds3=0000 ss=0040
# s ; reg
00000:0081b                   MOV AL,DS1:0000
PC = 00000:00815
reg ps=0000 pc=0815 psw=f046 aw=0000 bw=0000 cw=0000 dw=0000 sp=0100 bp=0000
reg ix=0000 iy=0000 ds0=0060 ds1=0040 ds2=0000 ds3=0000 ss=0040
```

Command Line Editing

The terminal interface supports the use of HP-UX **ksh(1)**-like editing of the command line. The default is for the command line editing feature to be disabled to be compatible with earlier versions of the interface. Use the **cl** command to enable command line editing.

M> **cl -e**

Refer to "Command Line Editing" in the *HP 64700-Series Emulators Terminal Interface Reference* for information on using the command line editing feature.

Modifying Memory

The preceding step and register commands show the sample program is executing Scan loop, where it continually reads the command input byte to check if a command had been entered. Use the **m** (memory) command to modify the command input byte.

M> **m 400=41**

To verify that 41H has been written to 400H, enter the following command.

M> **m -db 400**

```
0000400..0000400  41
```

When memory was displayed in byte format earlier, the display mode was changed to "byte". The display and access modes from previous commands are saved and they become the defaults.

Specifying the Access and Display Modes

There are a couple different ways to modify the display and access modes. One is to explicitly specify the mode with the command you are entering, as with the command **m -db 400**. The **mo** (display and access mode) command is another way to change the default mode. For example, to display the current modes, define the display mode as "word", and redisplay 400H, enter the following commands.

M> **mo**

```
mo -ab -db
```

M> **mo -dw**
M> **m 400**

```
0000400..0000400  0041
```

To continue the rest of program.

M> **r**
U>

Display the **Msg_Dest** memory locations (destination of the message, 401H) to verify that the program moved the correct ASCII bytes. At this time you want to see correct byte values, so "-db" option (display with byte) is used.

U> **m -db 401..420**

```
0000401..0000410  43 6f 6d 61 6e 64 20 41 20 65 6e 74 65 72 65
0000411..0000420  64 20 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Running the Sample Program

The emulator allows you to execute a program in memory with the **r** command. The **r** command by itself causes the emulator to begin executing at the current program counter address. The following command will begin running the sample program from 800h.

M> **r 800**

Note

The defaults number base for address and data values within HP 64700 Terminal Interface is hexadecimal. Other number bases may be specified. Refer to the "Expressions" chapter or the *HP 64700 Terminal Interface Reference* manual for further details.

The **r rst** command specifies that the emulator begin to executing from target system reset (see the "Execution Topics" section in the "In-Circuit Emulation" chapter).

Searching Memory for Data

The **ser** (search memory for data) command is another way to verify that the program did what it was supposed to do.

U> **ser 400..420="Command A entered "**

```
pattern match at address: 0000401
```

If any part of the data specified in the ser command is not found, no match is displayed (No message displayed).

Breaking into the Monitor

You can use the break command **(b)** command to generate a break to the monitor. While the break will occur as soon as possible, the actual stopping point may be many cycles after the break request (depending on the type of instruction being executed and whether the processor is in a special state).

```
U> b
M>
```

Note

If DMA transfer is in progress with BURST or DEMAND-RELEASE transfer mode, the emulator breaks into monitor after such DMA transfers are completed.

Using Software Breakpoints

Software breakpoints are handled by the 70433 BRK 3 instruction. When you define or enable a software breakpoint(whit the **bp** command), the emulator will replace the opcode at the software breakpoint address with a breakpoint interrupt instruction(BRK 3).

Caution

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

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 BRK 3 instructions, you cannot define software breakpoints in target ROM.

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

When software breakpoints are enabled and the emulator detects the BRK 3 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 BRK 3 interrupt was generated by an enabled software breakpoint or by a BRK 3 instruction in your target program.

If the BRK 3 interrupt was generated by a software breakpoint, execution breaks to the monitor, and the breakpoint interrupt instruction(BRK 3) is replaced by the original opcode. A subsequent run or step command will execute from this address.

If the BRK 3 interrupt was generated by a BRK 3 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.

Displaying and Modifying the Break Conditions

Before you can define software breakpoints, you must enable software breakpoints with the **bc** (break conditions) command. To view the default break conditions and change the software breakpoint condition, enter the **bc** command with no option. This command displays current configuration of break conditions.

M> **bc**

```
bc -d bp #disable
bc -e rom #enable
bc -d bnct #disable
bc -d cmbt #disable
bc -d trig1 #disable
bc -d trig2 #disable
```

To enable the software break point feature enter

M> **bc -e bp**

Defining a Software Breakpoint

Now that the software breakpoint feature is enabled, you can define software breakpoints. Enter the following command to break on the address of the **Cmd_I** (address 844H) label.

M> **bp 844**
M> **bp**

```
### BREAKPOINT FEATURE IS ENABLED ###
bp 0000844 #enabled
```

Run the program, and verify that execution broke at the appropriate address.

M> **r 0:800**
U> **m 400=43**

```
!ASYNC_STAT 615! Software breakpoint: 00000:00844
```

M> **st**

```
# s;reg
00000:00844  -          MOV CW,0012
PC = 00000:00847
reg ps=0000 pc=0847 psw=f002 aw=00043 bw=0000 cw=0012 dw=0000 sp=0100 bp=0000
reg ix=0012 iy=0021 ds0=0060 ds1=0040 ds2=0000 ds3=0000 ss=0040
```

When a breakpoint is hit, it becomes disabled. You can use the **-e** option with the **bp** command to re-enable the software breakpoint.

M> **bp**

```
### BREAKPOINT FEATURE IS ENABLED ###
bp 0000844 #disabled
```

```
                                M> bp -e 000844
                                M> bp
### BREAKPOINT FEATURE IS ENABLED ###
bp 0000844 #enabled
                                M> r
                                U> m 400=43
!ASYNC_STAT  615! Software breakpoint: 00000:00844
                                M> bp
### BREAKPOINT FEATURE IS ENABLED ###
bp 0000844 #disabled
```

Refer to the "Execution Topics" section in the "Emulation Topics" chapter and "ADDRESS" section in the "64768 Emulato r Specific Command Syntax" appendix.

Using the Analyzer

Predefined Trace Labels

Three trace labels are predefined in the 64768 emulator. You can view these labels by entering the **tlb** (trace label) command with no options.

```
                                M> tlb
#### Emulation trace labels
tlb addr 0..23
tlb data 24..39
tlb stat 40..57
```

Predefined Status Equates

Common values for the 64768 status trace signals have been predefined. You can view these predefined equates by entering the **equ** command with no options.

```
                                M> equ
```

```
### Equates ###
equ bg=0xxxxxxxxxxxxxxx0xy
equ cpu=0x0000xx01xxx1xxxxy
equ dma=0xx0000xx11xxx1xxxxy
equ exec=0xxxx01xxxxxxxxxxxxy
equ fetch=0xx0000xx10xxx1xxxxy
equ fg=0xxxxxxxxxxxxxxx1xy
equ grd=0xxxx00xxxxxxx1x0xxy
equ halt=0xx11xxxxxxxxx1xxxxy
equ hold=0xxxxxxxxxxxxx0xxxxy
equ int=0xxxx10xxxxxxxxxxxxy
equ io=0xx0000xx01x001xxxxy
equ mem=0x10000xxxxxxx1xxxxy
equ memio=0xx00000111xxx1xxxxy
equ memsfr=0xx00000011xxx1xxxxy
equ ms=1x0000xx01xxx1xxxxy
equ read=0xx0000xxx10xx1xxxxy
equ refresh=0xx0000xx001xx1xxxxy
equ sfr=0xx0000xx01x111xxxxy
equ stop=0xx10xxxxxxxxx1xxxxy
equ write=0xx0000xxx11xx1xxxxy
equ wrrom=0xxxx00xxxxxxx10xxxy
```

These equates may be used to specify values for the **stat** trace label when qualifying trace conditions.

Specifying a Simple Trigger

The **tg** analyzer command is a simple way to specify a condition on which to trigger the analyzer. Suppose you wish to trace the states of the program after the read of a "B" (42 hex) command from the command input byte. Enter the following commands to set up the trace, run the program, issue the trace, and display the trace status. (Note that the analyzer is to search for a lower byte read of 42H because the address is even.)

M> **tg addr=400 and data=0xx42**

If you wish to trace the odd address and the data, enter the following command to set up the trace (Note that the data value should be entered like as **0xx42 or 42xx** when you specify that CPU's data bus size is 16 bits in configuration.): tg addr=401 and data=42xx

M> **t**

```
emulation trace started
```

M> **r 0:800**
U> **ts**

```
--- Emulation Trace Status ---
New User trace running
Arm ignored
Trigger not in memory
Arm to trigger ?
States ? (512) ?..?
Sequence term 1
Occurrence left 1
```

The trace status shows that the trigger condition has not been found. You would not expect the trigger to be found because no commands have been entered. Modify the command input byte to "B"(42H) and display the trace status again.

U> **m 400=42**
U> **ts**

```
---Emulation Trace Status ---
New User trace complete
Arm ignored
Trigger in memory
Arm to trigger ?
States 512 (512) 0..511
Sequence term 2
Occurrence left 1
```

The trace status shows that the trigger has been found, and that 512 states have been stored in trace memory. Enter the following command to display the first 20 states of the trace.

U> **tl -t 20**

```
  Line   addr,H  70433 mnemonic,H                      count,R  seq
 -----   ------  ---------------------------------  ---------  ---
     0   000400    xx42  read mem                         ---   +
     1   000819  INSTRUCTION--opcode unavailable       0.320 uS   .
     2   00081c    3cf8  fetch                         1.600 uS   .
     3   00081b  INSTRUCTION--opcode unavailable       0.320 uS   .
     4   08b923    xxxx  refresh                       1.600 uS   .
     5   00081e    7441  fetch                         1.920 uS   .
     6   00081d  CMP AL,41                           0.320 uS   .
     7   000820    900d  fetch                         1.600 uS   .
     8   00081f  BE/Z 0082e                           0.320 uS   .
     9   000822    3c90  fetch                         1.600 uS   .
    10   000821  NOP                                 0.320 uS   .
    11   000822  NOP                                 1.280 uS   .
    12   000824    7442  fetch                         0.320 uS   .
    13   000823  CMP AL,42                           0.960 uS   .
    14   08b925    xxxx  refresh                       0.960 uS   .
    15   000826    9012  fetch                         1.920 uS   .
    16   000825  BE/Z 00839                          0.320 uS   .
    17   000828    eb90  fetch                         1.600 uS   .
    18   000839    b9xx  fetch                         1.920 uS   .
    19   00083a    0012  fetch                         1.920 uS   .
```

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

To list the next lines of the trace, enter the following command.

U> **tl**

```
 Line   addr,H  70433 mnemonic,H                      count,R  seq
-----   ------  ---------------------------------  ---------  ---
   20   000839  MOV CW,0012                         0.320 uS   .
   21   08b927    xxxx  refresh                     1.600 uS   .
   22   00083c    12be  fetch                       1.920 uS   .
   23   00083c  MOV IX,0012                         0.320 uS   .
   24   00083e    eb00  fetch                       1.600 uS   .
   25   000840    9009  fetch                       1.920 uS   .
   26   00083f  BR SHORT 0084a                      0.320 uS   .
   27   000842    9090  fetch                       1.600 uS   .
   28   00084a    01bf  fetch                       1.920 uS   .
   29   00084a  MOV IY,0001                         0.320 uS   .
   30   08b929    xxxx  refresh                     1.600 uS   .
   31   00084c    f300  fetch                       1.920 uS   .
   32   00084e    32a4  fetch                       1.920 uS   .
   33   00084d  REP/E/Z MOVBKB                      0.640 uS   .
   34   000850    b9c0  fetch                       1.280 uS   .
   35   000852    0021  fetch                       1.920 uS   .
   36   08b92b    xxxx  refresh                     1.920 uS   .
   37   000612    xx43  read mem                   1.920 uS   .
   38   000401    43xx  write mem                   2.880 uS   .
   39   000613    6fxx  read mem                   1.920 uS   .
```

Trigger Position

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

- s start
- c center
- e end

When s(start) trigger position is selected, the trigger is positioned at the start of the trace list. You can trace the states after the trigger state.

When c(center) trigger position is selected, the trigger is positioned at the center of the trace list. You can trace the states around the trigger.

When e(end) trigger position is selected, the trigger is positioned at the end of the trace list. You can trace the state before the trigger.

In the above section, you have traced the states of the program after a certain state, because the default trigger position was s(start). If you

want to trace the states of the program around a certain state, you need to change the trigger position.

For example, if you wish to trace the transition to the command A process, change the trigger position to "center" and specify the trigger condition.

To specify the trigger position, enter the following command.

U> **tp c**

Specify the trigger condition by typing

U> **tg addr=82e and stat=exec**

Enter the trace command to start the trace.

U> **t**

```
Emulation trace started
```

Modify the command input byte to "A" and display the trace status again.

U> **m 400=41**
U> **ts**

```
--- Emulation Trace Status ---
New User trace  complete
Arm ignored
Trigger not in memory
Arm to trigger ?
States 512 (512) -257..254
Sequence term 2
Occurrence left 1
```

The trace status shows that the trigger has been found. Enter the following command to display the states about the execution state of address 82eH.

U> **tl -10..9**

```
 Line   addr,H  70433 mnemonic,H                      count,R  seq
-----   ------  ---------------------------------  ---------  ---
  -10   000819  CMP AL,00                           0.320 uS   .
   -9   00081c    3cf8  fetch                       1.600 uS   .
   -8   00081b  BE/Z 00815                          0.320 uS   .
   -7   0489f1    xxxx  refresh                     1.600 uS   .
   -6   00081e    7441  fetch                       1.920 uS   .
   -5   00081d  CMP AL,41                           0.320 uS   .
   -4   000820    900d  fetch                       1.600 uS   .
   -3   00081f  BE/Z 0082e                          0.320 uS   .
   -2   000822    3c90  fetch                       1.600 uS   .
   -1   00082e    12b9  fetch                       1.920 uS   .
    0   00082e  MOV CW,0012                         0.320 uS   +
    1   0489f3    xxxx  refresh                     1.600 uS   .
    2   000830    be00  fetch                       1.920 uS   .
    3   000832    0000  fetch                       1.920 uS   .
    4   000831  MOV IX,0000                         0.320 uS   .
    5   000834    14eb  fetch                       1.600 uS   .
    6   000834  BR SHORT 0084a                      0.320 uS   .
    7   000836    9090  fetch                       1.600 uS   .
    8   0489f5    xxxx  refresh                     1.920 uS   .
    9   00084a    01bf  fetch                       1.920 uS   .
```

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

For a Complete Description

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

Copying Memory

The **cp** (copy memory) command gives you the ability to copy the contents of one range of memory to another. This is a handy feature to test things like the relocatability of programs, etc. To test if the sample program is relocatable within the same segment, enter the following command to copy the program to an unused, but mapped, area of emulation memory. After the program is copied, run it from its new start address to verify that the program is indeed relocatable.

```
U> cp 900=800..859
U> r 0:900
U>
```

The prompt shows that the emulator is executing user code, so it looks as if the program is relocatable. You may want to issue a simple trace to verify that the program works while running from its new location.

U> **tg any**
U> **t**

```
Emulation trace started
```

U> **tl**

```
   Line   addr,H  70433 mnemonic,H                      count,R  seq
  -----   ------  ---------------------------------  ---------  ---
      0   000915  INSTRUCTION--opcode unavailable          ---   +
      1   000918    3c00  fetch                       1.280 uS   .
      2   00091a    7400  fetch                       1.920 uS   .
      3   000400    xx00  read mem                    1.920 uS   .
      4   000919  CMP AL,00                          0.320 uS   .
      5   00091c    3cf8  fetch                       1.600 uS   .
      6   00091b  BE/Z 00915                         0.320 uS   .
      7   06193d    xxxx  refresh                     1.600 uS   .
      8   000915    26xx  fetch                       1.920 uS   .
      9   000916    00a0  fetch                       1.920 uS   .
     10   000915  MOV AL,DS1:BYTE PTR 0000           0.640 uS   .
     11   000918    3c00  fetch                       1.280 uS   .
     12   00091a    7400  fetch                       1.920 uS   .
     13   000400    xx00  read mem                    1.920 uS   .
     14   000919  CMP AL,00                          0.320 uS   .
     15   06193f    xxxx  refresh                     1.600 uS   .
     16   00091c    3cf8  fetch                       1.920 uS   .
     17   00091b  BE/Z 00915                         0.320 uS   .
     18   00091e    7441  fetch                       1.600 uS   .
     19   000915    26xx  fetch                       1.920 uS   .
```

Testing for Coverage

For each byte of emulation memory, there is an additional bit of emulation RAM used by the emulator to provide coverage testing. When the emulator is executing the target program and an access is made to a byte in emulation memory, the corresponding bit of coverage memory is set. With the **cov** command, you can see which bytes in a range of emulation memory have (or have not) been accessed.

For example, suppose you want to determine how extensive some test input is in exercising a program (in other words, how much of the program is covered by using the test input). You can run the program with the test input and then use the **cov** command to display which locations in the program range were accessed.

The examples which follow use the cov command to perform coverage testing on the sample program. Before performing coverage tests, reset all coverage bits to non-accessed by entering the following command.

U> **cov -r**

Run the program from the start address (00000:00400H) and use the cov command to display how much of the program is accessed before any commands are entered (refer to the "Execution Topics" section in the "Emulation Topics" chapter and "ADDRESS" section in the "64768 Emulator Specific Command Syntax" appendix).

U> **r 800**
R> **cov -a 800..859**

```
# coverage list  -  list of address ranges accessed
0000800..000081f

percentage of memory accessed: % 35.6
```

Now enter the sample program commands "A", "B", and an invalid command ("C" will do); display the coverage bits for the address range of the sample program after each command. You can see that more of the sample program address range is covered after each command is entered.

U> **m 400=41**
U> **cov -a 800..859**

```
# coverage list  -  list of address ranges accessed
0000800..0000823
000082e..0000839
000084a..0000859

percentage of memory accessed:  % 71.1
```

U> **m 400=42**
U> **cov -a 800..859**

```
# coverage list  -  list of address ranges accessed
0000800..0000829
000082e..0000843
000084a..0000859
percentage of memory accessed:  % 88.9
```

U> **m 400=43**
U> **cov -a 800..859**

```
# coverage list  -  list of address ranges accessed
0000800..0000859

percentage of memory accessed: % 100.0
```

Resetting the Emulator

To reset the emulator, enter the following command.

```
U> rst
R>
```

The emulator is held in a reset state (suspended) until a **b** (break), **r** (run), or **s** (step) command is entered. A CMB execute signal will also cause the emulator to run if reset.

The **-m** option to the **rst** command specifies that the emulator begin executing in the monitor after reset instead of remaining in the suspended state.

```
R> rst -m
M>
```

Notes

3

Emulation Topics

Introduction

Many of the topics described in this chapter involve the commands which are unique to the 64768 emulator such as the **cf** command which allows you t o specify emulator configuration.

A reference-type description of the 64768 emulator configuration items can be found in the "CONFIG_ITEMS" section in the "64768 Emulator Specific Command Syntax" appendix.

This chapter will:

- Describe how to run in real-time and how to break on an analyzer trigger,how to specify the address form, how to specify the default interpretation of physical run addresses, and how to break on an analyzer trigger. These topics are related to program execution in general.
- Describe how to locate the monitor, These topics are related to the monitor options.
- Describe how to do other things which do not fall into the categories mentioned above: how to access internal RAM/SFR.

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.

Execution Topics

The descriptions in this section are of emulation tasks which involve program execution in general.

Address Expression

The 64768 emulator allow you to use address forms as the following.

- physical address form: (e.g.,0ffff0)
- function code address form: (e.g.,0ef@iram)
- logical address form: (e.g.,0fff0:00ff)
- extended logical address form: (e.g.,0fff0::00ff)

You must specify addresses in function code address form(the **iram** information), when you access Internal RAM space(addresses from 000@iram to 1ff@iram).

If you specify the address in 1M bytes memory space, you can specify addresses in either physical address form(00000 to 0fffff) or logical address form.

If you specify the address in 16M bytes memory space, you can specify addresses in either physical address form (000000 to 0ffffff) or extended logical address form.

The address expression is restricted by emulation commands as shown below.

Table 3-1 Address Expression Matrix

Command group	Terminal command	PHY_ADDR	IRAM_ADDR @iram	: <OFFSET>	<XSEGMENT>:: <XOFFSET>
Memory commands	cp,dump, m,ser	OK	OK	OK	OK
	cim	OK	OK (*1)	OK	OK
	cov	OK	ERROR	OK	OK
Run commands (*2)	r,xr,s,ss (rad=maxseg)	OK	ERROR	OK	ERROR
	r,xr,s,ss (rad=minseg)	OK	ERROR	OK	ERROR
	r,xr,s,ss (rad=curseg)	(0-0FFFFH)	ERROR	OK	ERROR
I/O command	io	OK (0-0FFFFH)	ERROR	ERROR	ERROR
Map command	map	OK	ERROR	ERROR	ERROR
Breakpoints command	bp	OK (0-0FFFFFH)	ERROR	OK	ERROR
Symbols command	sym	OK	OK	OK	OK

*1 : The emulator ignores a function code.

*2 : Refer to the "cf rad" command in "CONFIG_ITEMS" section of "Emulation Specific Command Syntax"

Note

When you specify address range, you can not specify end address and start address in different address form respectively. But following form are permitted.

- physical..physical@iram(physical@iram..physical)
  physical address is recognized as function code address.
- segment:offset..physical(physical..segment:offset)
  physical address is recognized as offset, and segment is same value.
- xsegment::xoffset..physical(physical..xsegment::xoffset)
  physical address is recognized as xoffset, and xsegment is same value.

Default Physical to Logical Run Address Conversion

The run and step commands allows you to enter addresses in ether logical address form(segment:offset) or physical address form.

When a physical address(non-segment) is entered with either a run or step command, the emulator must convert it to a logical(segment:offset) address. By default, a physical run address is converted such that the low 16 bits of the address become the offset value. The physical address is right-sifted 4 bits and ANDed with 0F000H to yield segment.Use the **cf**(configuration) command with **rad**(run address default conversion) configuration item to specify how the low 4 bits of physical address become the offset. The physical address is right-sifted 4 bits to yield the segment value.

R> **cf rad=minseg**

#phys_addr =
((phys_addr >4) & 0xf000h):(phys_addr & 0xffff)

R> **cf rad=maxseg**

#phys_addr =
(phys_addr >4):(phys_addr & 0xf)

To configure so that the physical address(0h-0ffffh) becomes the offset value, enter following command. The value of current program segment(PS) becomes the segment value.

R> **cf rad=curseg**

#phys_addr=(current segment):(entered value)

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

Restricting the Emulator to Real-Time Runs

By default, the emulator is not restricted to real-time runs. However, you may wish to restrict runs to real-time to prevent accidental breaks that might cause target system problems. Use the **cf** (configuration) command to enable the **rrt** configuration item.

R> **cf rrt=en**

When runs are restricted to real-time and the emulator is running user code, the system refuses all commands that cause a break except **rst** (reset), **r** (run), **s** (step), and **b** (break to monitor).

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

The following commands are not allowed when runs are restricted to real-time:

- **reg** (register display/modification).
- **m** (memory display/modification) commands that access target system memory.
- **io** (I/O display/modification).

The following command will disable the restriction to real-time runs and allow the system to accept commands normally.

R> **cf rrt=dis**

Setting Up to Break on an Analyzer Trigger

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

Specify the Signal Driven when Trigger is Found

Use the **tgout** (trigger output) command to specify which signal is driven when the analyzer triggers. Either the "trig1" or the "trig2" signal can be driven on the trigger.

```
R> tgout trig1
```

Enable the Break Condition

Enable the "trig1" break condition.

```
R> bc -e trig1
```

After you specify the trigger to drive "trig1" and enable the "trig1" break condition, set up the trace, issue the **t** (trace) command, and run the program.

Making Coordinated Measurements

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

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

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

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

Analyzer Topics

Analyzer Status Qualifiers

The follwing are the analyzer status labels which may be used in the "**tg**" and "**tsto**" analyzer commands.

Qualifier	Status bits	Description
bg	0xxxxxxxxxxxxxxx0xy	background
cpu	0x0000xx01xxx1xxxxy	cpu cycle
dma	0xx0000xx11xxx1xxxxy	DMA memory access
exec	0xxxx01xxxxxxxxxxxxy	execute instruction
fetch	0xx0000xx10xxx1xxxxy	program fetch
fg	0xxxxxxxxxxxxxxx1xy	foreground
grd	0xxxx00xxxxxxx1x0xxy	guarded memory access
halt	0xx11xxxxxxxxx1xxxxy	halt
hold	0xxxxxxxxxxxxx0xxxxy	hold acknowledge
int	0xxxx10xxxxxxxxxxxxy	interrupt acknowledge
io	0xx0000xx01x001xxxxy	I/O access
mem	0x10000xxxxxxx1xxxxy	memory access
memio	0xx00000111xxx1xxxxy	memory to io
memsfr	0xx00000011xxx1xxxxy	memory to sfr
ms	01x0000xx01xxx1xxxxy	macro service
read	0xx0000xxx10xx1xxxxy	read
refresh	0xx0000xx001xx1xxxxy	refresh cycle
sfr	0xx0000xx01x111xxxxy	sfr access
stop	0xx10xxxxxxxxx1xxxxy	stop
write	0xx0000xxx11xx1xxxxy	write
wrrom	0xxxx00xxxxxxx10xxxy	write to rom

Specifying Data for Trigger Condition or Store Condition

The analyzer captures the data of the 70433 microprocessor. When you specify a data in the analyzer trigger condition or store condition, the ways of analyzer data specifications differ according to the data size.

To trigger the analyzer when the 70744 microprocessor accesses the word data 1234H at address 1000H in 8bit data bus size. the data bus activity of the cycles will be as follows.

```
Sequencer level Address bus  Data bus
    1   1000H        xx34
    2   1001H        xx12
```

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

M> **tif 1 addr=1000 and data=0xx34**

To specify the condition of sequencer level 2, enter:

M> **tif 2 addr=1001 and data=0xx12**

To restart sequencer when any states except for "exec" status are generated between sequencer term 1 and 2.

M> **telif stat!=exec**

Monitor Option Topics

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 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.

Background Monitor

When the emulator is powered up or initialized, the background monitor is selected by default.

Foreground monitor

The default emulator configuration selects the background monitor. You can change the emulator configuration to select the foreground monitor. When you select the foreground monitor, processor address space is taken up. The foreground monitor takes up 2K bytes of memory. Use the **cf** command to select the foreground monitor.

R> **cf mon=fg..1000**

1000 defines an hexadecimal address (on a 2K byte boundary) where the monitor will be located. (Note: this will not load the monitor, it only specifies its location.) The start address of the foreground monitor

must be 2k boundary and between 800H and 0FF000H. The foreground monitor must then be loaded into emulation memory. A memory mapper term is automatically created when you execute the **cf mon=fg** command to reserve 2K bytes of memory space for the monitor. The memory map is reset any time **cf mon=bg** is entered. It is only reset when the **cf mon=bg** command is entered if the emulator is not already configured to use the background monitor. Refer to the "Using the Optional Foreground Monitor" appendix.

Note

The foreground monitor provided with 64768 emulator should not be located at a base address 0 or 0ff800 hex; because the 70433 microprocessor's vector table or SFR are located respectively.

Note

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

Other Topics

This section describes how other emulation tasks, which did not fit into the previous groupings, are performed.

Accessing Internal RAM/SFR

When you access internal RAM ,you can use the **m** command with function code and the **reg** command. If you display/modify register in the current register bank, you must use **reg** command instead of the **m** command. Otherwise you will destroy the monitor program.

When you access SFR(Special Function Registers), you must use the reg command with their register name instead of the m command. You can access SFR regardless of memory mapping.

Notes

4

In-Circuit Emulation Topics

Introduction

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

This chapter will:

- Describe the issues concerning the installation of the emulator probe into target systems.
- Show you how to install the emulator probe.
- Describe how to set up the emulator to use a target system clock, how to execute program from target reset and how to allow DMA accesses to emulation memory. These topics are related to program execution in general.
- Describe how to use software breakpoints with ROMed code, how to perform coverage testing on ROMed code, and how to test patches to ROMed code. These topics relate to the debugging of target system ROM.
- Describe some of restrictions and considerations.

Prerequisites

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

Installing the Emulator Probe into a Target System

The 64768 emulator probe has a 132-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 static discharge. The emulator probe contains devices that are susceptible to damage by static discharge. Therefore, precautionary measures should be taken before handling the microprocessor connector attached to the end of the probe cable to avoid damaging the internal components of the probe by static electricity.

Caution

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

Caution

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

Caution

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

Pin Protector

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

Conductive Pin Guard

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

Caution

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

Installing into a PGA Type Socket

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

- Remove the 70433 microprocessor (PGA type) from the target system socket. Note the location of pin A1 on the microprocessor and on the target system socket.
- Store the microprocessor in a protected environment (such as antistatic form).
- Install the microprocessor connector into the target system microprocessor socket.

Caution

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

Installing into a QFP Type Socket

To connect the 64768 emulator microprocessor connector to the NEC EV-9200GD-120 socket on the target system, use the NEC EV-9501GD-120 adapter.

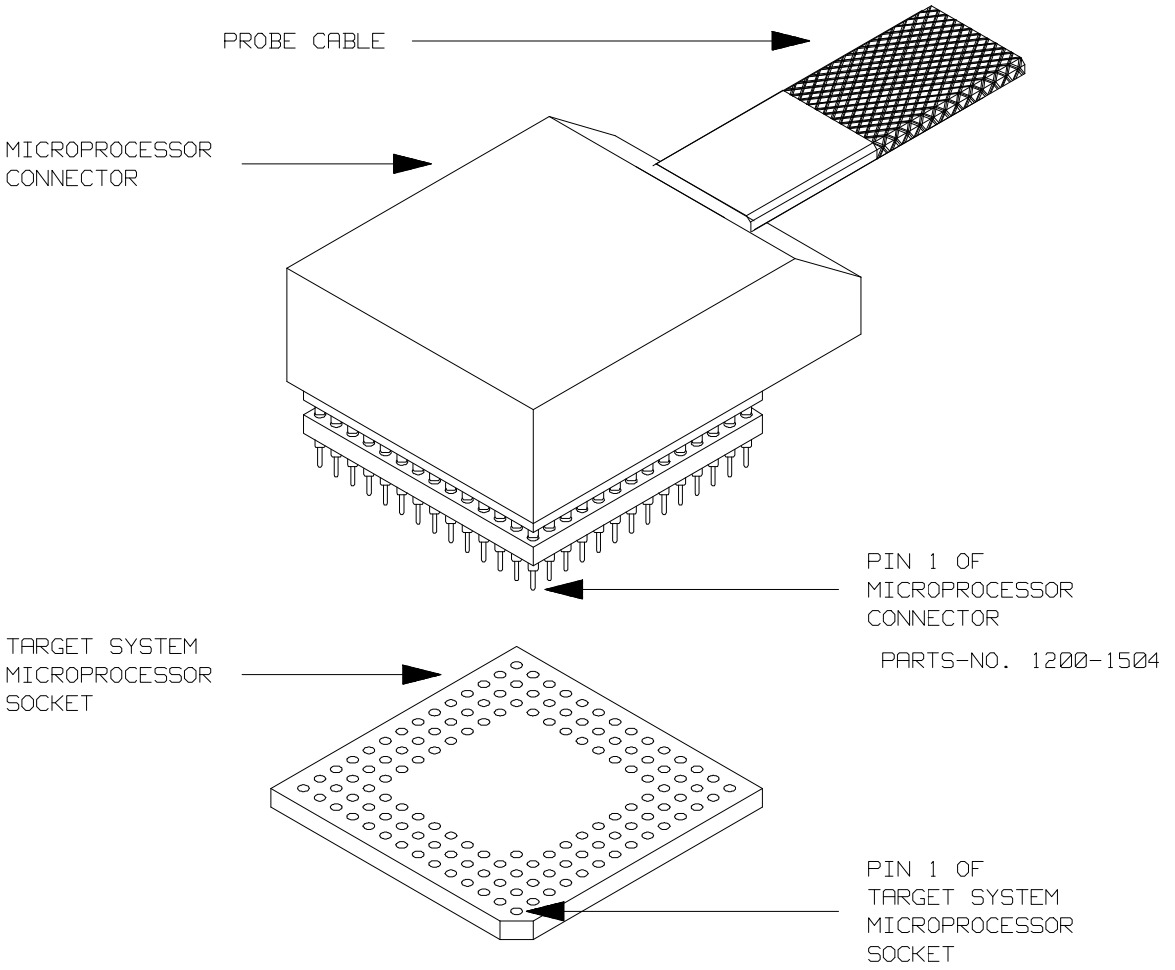


Figure 4-1 Installing into a 70433 PGA type socket

Execution Topics

The descriptions in this section are of emulation tasks which involve program execution in general.

Specifying the Emulator Clock Source

The default 64768 emulator configuration selects the internal 12.5MHz (system clock speed) clock as the emulator clock source. You should configure the 64768 emulator to select an external target system clock source for the "in-circuit" emulation. Use the **cf** (configuration) command and the **clk** configuration item to specify that the emulator use a target system clock.

```
R> cf clk=ext
```

To reconfigure the emulator to use its internal clock, enter the following command.

```
R> cf clk=int
```

DMA Cycles

Cycles from the emulation processor's internal DMA controller is allowed to access emulation memory. If DMA controller is programmed to transfer with **BURST** or **DEMAND RELEASE** mode, the emulator can not break in monitor until transfers are completed.With other DMA modes, the emulator break into the monitor(background) upon request, however, DMA transfers are continued. DMA cycles are sent to the analyzer.

Note

Target system DMA controller access to emulation memory is not allowed. Only internal DMA controller can access emulation memory.

Run from Target System Reset

You can use "**r rst**" command to execute program from target system reset. If you use background monitor, you will see **T**> system prompt when you enter "r rst". In this status, the emulator accept target system reset. Then program stars if reset signal from target system is released.

If you use foreground monitor, reset signal from target system is always accepted.

Note

In the "Awaiting target reset" status(T>), you can not break into the monitor. If you enter "r rst" in out-of-circuit or in the configuration that emulator does not accepted target system reset(cf rst=dis), you must reset the emulator.

The 64768 emulator supports power on reset. If you want program to be executed by power on reset, execute the following process.

1) Enter "rst"

2) Turn OFF your target system

3) Enter "r rst"

4) Turn On your target system

Note

When you enter "r rst", you will see c> system prompt if you use external clock(cf clk=ext). This status is the same as "Awaiting target reset" status.

Emulator Probe Signal Topics

The descriptions in this section are of emulation tasks which involve emulator probe signals while in background or while accessing emulation memory.

Allowing the Target System to Insert Wait States

High-speed emulation memory provides no-wait-state operation. However, the emulator may optionally respond to the target system ready lines while emulation memory is being accessed. Use the **cf** (configuration) command with the **rdy** configuration item to cause emulation memory accesses to honor target system ready signals.

```
R> cf rdy=lk
```

To reconfigure so that emulation memory accesses do not honor target system ready signals, enter the following command.

```
R> cf rdy=unlk
```

Accepting the DMA Request Signals from Target System

Even if the emulator is running in the background monitor, DMARQ0, DMARQ1 signals from target system can be accepted.

Note

Frequent DMA requests will slow down the monitor operation. It may cause a failure.

Target ROM Debug Topics

The descriptions in this section are of emulation tasks which involve debugging target ROM. The tasks described below are made possible by the **cim** (copy target system memory image) command.

The **cim** command allows you to read the contents of target memory into the corresponding emulation memory locations. Moving target ROM contents into emulation memory is the key which allows you to perform the tasks described below. For example, if target ROM exists at locations 400H through 0A38H, you can copy target ROM into emulation memory with the following commands.

```
R> map 400..0bff erom
R>cim 400..0a38
```

Using Software Breakpoints with ROMed Code

You cannot define software breakpoints in target ROM memory. However, you can copy target ROM into emulation memory which does allow you to use software breakpoints.

Once target ROM is copied into emulation memory, software breakpoints may be used normally at addresses in these emulation memory locations.

```
R> bc -e bp
```

```
R> bp 440
```

Coverage Testing ROMed Code

Coverage testing (as described in the "Getting Started" chapter) can only be performed on emulation memory. However, if you wish to perform coverage tests on code in target system ROM, you can copy target ROM into emulation memory and perform the coverage tests on your ROMed code.

Once target ROM is copied into emulation memory, coverage testing may be done normally at addresses in these emulation memory locations.

```
U> cov -a 400..0a38
```

Modifying ROMed Code

Suppose that, while debugging your target system, you begin to suspect a bug in some target ROM code. You might want to fix or "patch" this code before programming new ROMs. This can also be done by copying target system ROM into emulation memory with the **cim** (copy target memory image) command. Once the contents of target ROM are copied into emulation memory, you can modify emulation memory to "patch" your suspected code.

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.
$\overline{\text{ASTB}}$	Same as foreground.
$\overline{\text{DEX}}$	Same as foreground
$\overline{\text{WRL}}$,$\overline{\text{WRH}}$	Always high level, except accessing target memory, I/O by background monitor.
$\overline{\text{RD}}$	Same as foreground except for emulation memory write. When accessing emulation memory, low.
Other	Same as foreground

Electrical Characteristics

The AC characteristics of the HP 64768A/B emulators are listed in the following table

Table 4-1 AC Electrical Specifications

Characteristic	Symbol	uPD70433 12.5MHz		HP 64768A			Unit
				Worst Case		Typical (*1)	
		Min	Max	Min	Max		
CLKOUT Low to Address Valid	t_{DKA}	5	30	1	39	16.8	ns
CLKOUT High to Address Invalid	t_{HKA}	5		1		16.8	ns
CLKOUT High to Address-Data Bus High Impedance	t_{FKT}		40		43	8.3	ns
Address Valid to $\overline{\text{ASTB}}$ Asserted	t_{SAST}	15		5		36.0	ns
$\overline{\text{ASTB}}$ Asserted to Address Invalid	t_{HSTA}	25		17		30.4	ns
CLKOUT Low to $\overline{\text{ASTB}}$ Negated	t_{DKSTL}	0	25	-4	32	7.6	ns
CLKOUT Low to $\overline{\text{ASTB}}$ Negated	t_{DKSTH}	0	25	-4	32	6.0	ns
$\overline{\text{ASTB}}$ Width High	t_{WSTH}	65		60		79.6	ns
CLKOUT High to RD Asserted(Read)	t_{DKRL}	0	25	-1	42	26.4	ns
CLKOUT High to RD Negated(Read)	t_{DKRH}	0	25	-1	40	15.6	ns
$\overline{\text{RD}}$ Width Low (Read)	t_{WRL}	105		100		112.4	ns
Address-Data Bus High Impedance to $\overline{\text{RD}}$ Asserted(Read)	t_{FARL}	0		-21		18.1	ns
$\overline{\text{RD}}$ Negated to Address Valid(Read)	t_{DRA}	40		4		30.4	ns

Table 4-1 AC Electrical Specification(Cont'd)

Characteristic	Symbol	uPD70433 12.5MHz		HP 64768A			Unit
				Worst Case		Typical (*1)	
		Min	Max	Min	Max		
CLKOUT Low to $\overline{DEX}$ Asserted,Negated	t_{DKDX}	0	30	-4	39	-	ns
CLKOUT Low to $\overline{DEX}$ Asserted,Negated	t_{HKDX}	0		-4		-	ns
CLKOUT Low to Data-In Valid	t_{SDK}	15		33		21.0	ns
CLKOUT Low to Data-In Invalid	t_{HKDR}	0				0	ns
CLKOUT Low to $\overline{WR}$ Asserted(Write)	t_{DKWL}	0	25	-1	42	18.4	ns
CLKOUT Low to $\overline{WR}$ Negated(Write)	t_{DKWH}	0	25	-1	40	14.4	ns
$\overline{WR}$ Width Low (Write)	t_{WWL}	65		60		72.8	ns
CLKOUT High to Data-out Valid(Write)	t_{DKD}	3	30	-1	39	14.4	ns
CLKOUT Low to Data-out Invalid(Write)	t_{HKDW}	0		-4		14.8	ns
$\overline{WR}$ Negated to $\overline{ASTB}$ Negated(Write)	t_{DWSTH}	0		-15		-7.6	ns
CLKOUT High to $\overline{RAS}$ Asserted	t_{DKRAL}	0	25	-4	34	-	ns
CLKOUT High to $\overline{RAS}$ Negated	t_{DKRAH}	0	25	-4	29	-	ns
$\overline{RAS}$ Width High	t_{WRAH}	65		63		-	ns
$\overline{WR}$ Asserted to $\overline{RAS}$ Negated	t_{SWRA}	30		10		-	ns

Table 4-1 AC Electrical Specification(Cont'd)

Characteristic	Symbol	uPD70433 12.5MHz		HP 64768A			Unit
				Worst Case		Typical (*1)	
		Min	Max	Min	Max		
READY Asserted Input Setup Time	t_{SRYHK}	18		41		30.0	ns
READY Asserted Input Hold Time	t_{HKRYL}	15				15	ns
READY Negated Input Setup Time	t_{SRYLK}	18		41		30.0	ns
READY Negated input Hold Time	t_{HKRYH}	15				15	ns
$\overline{\text{RESET}}$ Width Low (Stop/PWR on RST)	t_{WRSL1}	30				30	ms
$\overline{\text{RESET}}$ Width Low (System Reset)	t_{WRSL2}	5				5	us
NMI Width High	t_{WNIH}	5				5	us
NMI Width Low	t_{WNIL}	5				5	us
$\overline{\text{POLL}}$ Setup Time	t_{SPLK}	30		43		-	ns
HLDREQ Setup Time	t_{SHQK}	30		70		-	ns
CLKOUT Low to $\overline{\text{HLDACWR}}$ Asserted	t_{DKHA}	0	30	-4	39	-	ns
$\overline{\text{HLDAK}}$ Asserted to Bus Control (*2) High Impedance	t_{FCHA}	0		-3		-	ns
$\overline{\text{HLDAK}}$ Negated to Bus Control (*2) Driven	t_{DHAC}	40		42		-	ns
HLDREQ Negated to $\overline{\text{HLDAC}}$ Negated	t_{DHQHA}		280		326	-	ns

Table 4-1 AC Electrical Specification(Cont'd)

Characteristic	Symbol	uPD70433 12.5MHz		HP 64768A			Unit
				Worst Case		Typical (*1)	
		Min	Max	Min	Max		
HLDREQ Negated to Bus control (*2) Driven	t_{DHQC}	90		87		-	ns
HLDREQ Width Low	t_{WHQL}	160		155		-	ns
$\overline{\text{HLDAK}}$ Width Low	t_{WHAL}	240		235		-	ns
CLKOUT Low to $\overline{\text{BUSLOCK}}$ Asserted	t_{DKBL}	2	25	-2	34	-	ns
$\overline{\text{RD}}$,$\overline{\text{IORD}}$ Negated to $\overline{\text{ASTB}}$ Negated	t_{DRST}	0		-15		-	ns
$\overline{\text{WR}}$,$\overline{\text{IOWR}}$ Negated to $\overline{\text{RD}}$,$\overline{\text{IORD}}$ Negated	t_{DWR}	0		-12		-	ns
CLKOUT High to $\overline{\text{DMAAKm}}$ Asserted	t_{DKDA}	0	30	-4	39	-	ns
$\overline{\text{DMAAKm}}$ Width Low	t_{WDAL}	230		220		-	ns
CLKOUT High to $\overline{\text{TCEm}}$ Asserted	t_{DKTE}	0	30	-4	39	-	ns
$\overline{\text{TCEm}}$ Width Low	t_{WTCL}	70		60		-	ns
$\overline{\text{WDTOUT}}$ Width low	t_{WWTL}	2550		-		2550	ns

*1 Typical outputs measured with 50pF load

*2 $\overline{\text{ASTB}}$,$\overline{\text{RD}}$,$\overline{\text{WRH}}$,$\overline{\text{WRL}}$,$\overline{\text{DEX}}$,$\overline{\text{RAS}}$,$\overline{\text{BUSLOCK}}$,$\overline{\text{IORD}}$,$\overline{\text{IOWR}}$,AD0-AD15,A16-A23

Table 4-2 AC Electrical Specifications

Characteristic	Symbol	uPD70433 16MHz		HP 64768B			Unit
				Worst Case		Typical (*1)	
		Min	Max	Min	Max		
CLKOUT Low to Address Valid	t_{DKA}	5	27	1	36	16.4	ns
CLKOUT High to Address Invalid	t_{HKA}	0		-4		16.4	ns
CLKOUT High to Address-Data Bus High Impedance	t_{FKT}		36		39	8.3	ns
Address Valid to $\overline{\text{ASTB}}$ Asserted	t_{SAST}	6		-4		36.0	ns
$\overline{\text{ASTB}}$ Asserted to Address Invalid	t_{HSTA}	16		8		19.8	ns
CLKOUT Low to $\overline{\text{ASTB}}$ Negated	t_{DKSTL}	0	22	-4	29	12.8	ns
CLKOUT Low to $\overline{\text{ASTB}}$ Negated	t_{DKSTH}	0	22	-4	29	9.6	ns
$\overline{\text{ASTB}}$ Width High	t_{WSTH}	47		42		62.0	ns
CLKOUT High to RD Asserted(Read)	t_{DKRL}	0	22	-1	39	26.8	ns
CLKOUT High to RD Negated(Read)	t_{DKRH}	0	22	-1	37	15.2	ns
$\overline{\text{RD}}$ Width Low (Read)	t_{WRL}	78		73		84.0	ns
Address-Data Bus High Impedance to $\overline{\text{RD}}$ Asserted(Read)	t_{FARL}	0		-21		18.5	ns
$\overline{\text{RD}}$ Negated to Address Valid(Read)	t_{DRA}	31		-2		36.4	ns

Table 4-2 AC Electrical Specification(Cont'd)

Characteristic	Symbol	uPD70433 16MHz		HP 64768B			Unit
				Worst Case		Typical (*1)	
		Min	Max	Min	Max		
CLKOUT Low to $\overline{DEX}$ Asserted,Negated	t_{DKDX}	0	27	-4	36	-	ns
CLKOUT Low to $\overline{DEX}$ Asserted,Negated	t_{HKDX}	0		-4		-	ns
CLKOUT Low to Data-In Valid	t_{SDK}	11		32		-	ns
CLKOUT Low to Data-In Invalid	t_{HKDR}	0		0		-	ns
CLKOUT Low to $\overline{WR}$ Asserted(Write)	t_{DKWL}	0	22	-1	39	18.8	ns
CLKOUT Low to $\overline{WR}$ Negated(Write)	t_{DKWH}	0	22	-1	37	12.8	ns
$\overline{WR}$ Width Low (Write)	t_{WWL}	50		45		51.2	ns
CLKOUT High to Data-out Valid(Write)	t_{DKD}	3	27	-1	36	26.8	ns
CLKOUT Low to Data-out Invalid(Write)	t_{HKDW}	0		-4		10.8	ns
$\overline{WR}$ Negated to $\overline{ASTB}$ Negated(Write)	t_{DWSTH}	0		-15		-7.2	ns
CLKOUT High to $\overline{RAS}$ Asserted	t_{DKRAL}	0	22	-4	31	-	ns
CLKOUT High to $\overline{RAS}$ Negated	t_{DKRAH}	0	22	-4	26	-	ns
$\overline{RAS}$ Width High	t_{WRAH}	47		45		-	ns

Table 4-2 AC Electrical Specification(Cont'd)

Characteristic	Symbol	uPD70433 16MHz		HP 64768B			Unit
				Worst Case		Typical (*1)	
		Min	Max	Min	Max		
READY Asserted Input Setup Time	t_{SRYHK}	18		41		30.0	ns
READY Asserted Input Hold Time	t_{HKRYL}	12		12		-	ns
READY Negated Input Setup Time	t_{SRYLK}	18		41		30.0	ns
READY Negated input Hold Time	t_{HKRYH}	12		12		-	ns
$\overline{\text{RESET}}$ Width Low (Stop/PWR on RST)	t_{WRSL1}	30		30		-	ms
$\overline{\text{RESET}}$ Width Low (System Reset)	t_{WRSL2}	1		1		-	us
NMI Width High	t_{WNIH}	5		5		-	us
NMI Width Low	t_{WNIL}	5		5		-	us
$\overline{\text{POLL}}$ Setup Time	t_{SPLK}	25		38		-	ns
HLDREQ Setup Time	t_{SHQK}	25		65		-	ns
CLKOUT Low to $\overline{\text{HLDACWR}}$ Asserted	t_{DKHA}	0	27	-4	36	-	ns
$\overline{\text{HLDAK}}$ Asserted to Bus Control (*2) High Impedance	t_{FCHA}	0		-3		-	ns
$\overline{\text{HLDAK}}$ Negated to Bus Control (*2) Driven	t_{DHAC}	39		41		-	ns
HLDREQ Negated to $\overline{\text{HLDAC}}$ Negated	t_{DHQHA}		252		298	-	ns

Table 4-2 AC Electrical Specification(Cont'd)

Characteristic	Symbol	uPD70433 16MHz		HP 64768B			Unit
				Worst Case		Typical (*1)	
		Min	Max	Min	Max		
HLDREQ Negated to Bus control (*2) Driven	t_{DHQC}	76		73		-	ns
HLDREQ Width Low	t_{WHQL}	124		119		-	ns
$\overline{\text{HLDAK}}$ Width Low	t_{WHAL}	176		171		-	ns
CLKOUT Low to $\overline{\text{BUSLOCK}}$ Asserted	t_{DKBL}	0	27	-4	36	-	ns
$\overline{\text{RD}}$,$\overline{\text{IORD}}$ Negated to $\overline{\text{ASTB}}$ Negated	t_{DRSTH}	0		-15		-	ns
$\overline{\text{WR}}$,$\overline{\text{IOWR}}$ Negated to $\overline{\text{RD}}$,$\overline{\text{IORD}}$ Negated	t_{DWRH}	0		-15		-	ns
CLKOUT High to $\overline{\text{DMAAKm}}$ Asserted	t_{DKDA}	0	27	-4	36	-	ns
$\overline{\text{DMAAKm}}$ Width Low	t_{WDAL}	176		166		-	ns
CLKOUT High to $\overline{\text{TCEm}}$ Asserted	t_{DKTE}	0	27	-4	36	-	ns
$\overline{\text{TCEm}}$ Width Low	t_{WTCL}	52		42		-	ns
$\overline{\text{WDTOUT}}$ Width low	t_{WWTL}	1974		-		-	ns

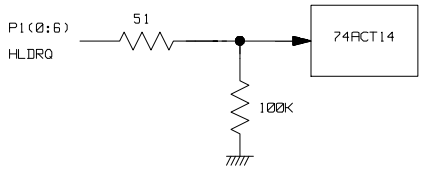
*1 Typical outputs measured with 50pF load

*2 $\overline{\text{ASTB}}$,$\overline{\text{RD}}$,$\overline{\text{WRH}}$,$\overline{\text{WRL}}$,$\overline{\text{DEX}}$,$\overline{\text{RAS}}$,$\overline{\text{BUSLOCK}}$,$\overline{\text{IORD}}$,$\overline{\text{IOWR}}$,AD0-AD15,A16-A23

Target System Interface

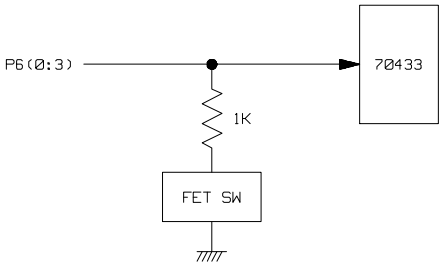
P1(0:6)
HLDRQ

These signals are connected to 74ACT14 through 51 ohm series register and 100K ohm pull-down register.



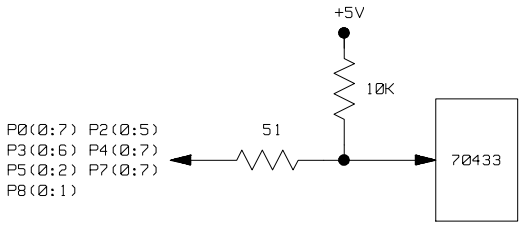
P6(0:3)

These signals are connected to 64768 emulation processor and FET Switch through 1K ohm register.

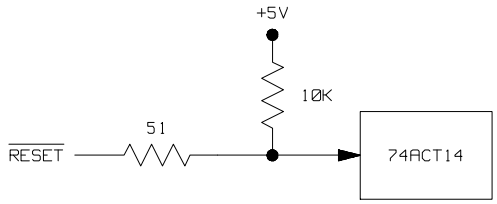


P0(0:7) P2(0:5)
P3(0:6) P4(0:7)
P5(0:2) P7(0:7)
P8(0:1)

These signals are connected to 64768 emulation processor through 51 ohm register and 10K ohm pull-up register.



$\overline{\text{RESET}}$ This signal is connected to 74ACT14 through 51 ohm register and 10K ohm pull-up register.



Other signals These signals are connected to 74FCT245 or 74FCT244 through 51 ohm register and 10K ohm pull-up register.

A

64768 Emulator Specific Command Syntax

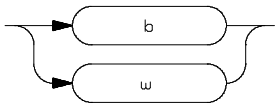
The following pages contain descriptions of command syntax specific to the HP 64768 emulator. The following syntax items are included (several items are part of other command syntax):

- <ACCESS_MODE>. May be specified in the **mo** (display and access mode), **m** (memory), and **io** (I/O port) commands. The access mode is used when the **m** or **io** commands modify target memory or I/O locations.
- <ADDRESS>. May be specified in emulation commands which allow addresses to be entered.
- <CONFIG_ITEMS>. May be specified in the **cf** (emulator configuration) and **help cf** commands.
- <DISPLAY_MODE>. May be specified in the **mo** (display and access mode), **m** (memory), **io** (I/O port), and **ser** (search memory for data) commands. The display mode is used when memory locations are displayed or modified.
- <REG_NAME> and <REG_CLASS>. May be specified in the **reg** (register) command.

ACCESS_MODE

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

Syntax



Function The **<ACCESS_MODE>** 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.

Parameters

b	**Byte.** Selecting the byte access mode specifies that the emulator will access target memory using upper and lower byte cycles (one byte at a time).
w	**Word.** Selecting the word access mode specifies that the emulator will access target memory using word cycles (one word at a time) at an even address. At an odd address, the emulator will access target memory using byte cycles.

Defaults In the HP 64768, the **<ACCESS_MODE>** is **b** at power up initialization. Access mode specifications are saved; that is, when a command changes the access mode, the new access mode becomes the current default.

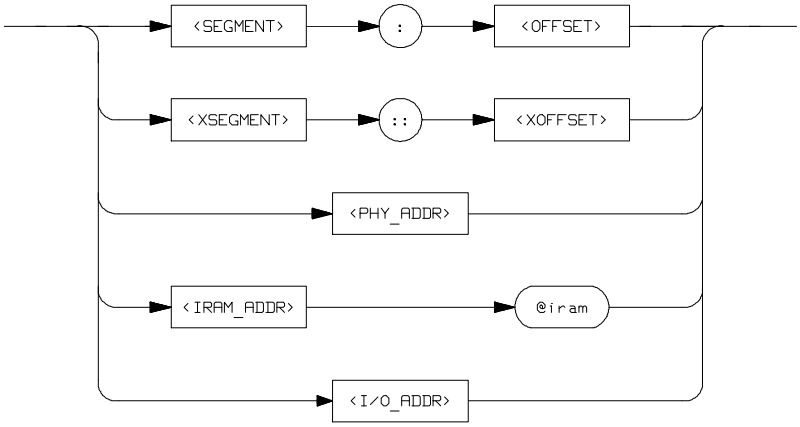
Related Commands **mo** (specify display and access modes)

ADDRESS

Address Syntax

Address specifications used in emulation commands.

Syntax



Function

The **<ADDRESS>** parameter used in emulation commands may be specified as a logical address, extended logical address, physical address(though a physical address in run or step command is converted to logical address by the emulation system), function code address.

Parameters

**** This expression (0-0FFFF hex) is the segment portion of the logical address. The value specified is placed in the 70433 PS register before running or stepping.

<OFFSET> This expression (0-0FFFF hex) is the offset portion of the logical address. The value specified is placed in the 70433 PC register before running or stepping.

<XSEGMENT> This expression (0-0FFFF hex) is the segment portion of the extended logical address.

<XOFFSET> This expression (0-0FFFF hex) is the offset portion of the extended logical address.

<PHY_ADDR> This expression (0-0FFFFFF hex) is a physical address in the 70433 address range. In run and step commands, the only expression (0-0FFFFF hex) is permitted, and the emulation system converts this physical address to a logical address as specified by the **rad** (run address default) configuration item (see the **<CONFIG_ITEM>** description).

<IRAM_ADDR> This expression (0-1FF hex) with function code is a address in the 70433 internal RAM address range. This expression should be used in memory command.

<I/O_ADDR> This expression (0-0FFFF hex) with no function code is a address in the 70433 I/O address range. This expression should be used in I/O command.

Defaults

If no number base is specified, values entered are interpreted as hexadecimal numbers.

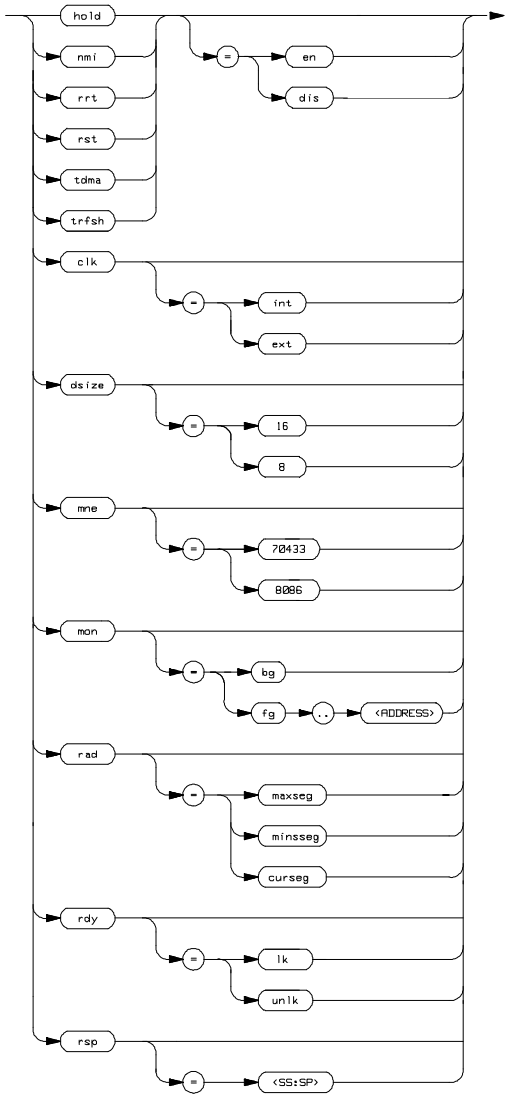
Related Commands

<CONFIG_ITEMS> (70433 specific items specified with the **cf** command)

CONFIG_ITEMS

Summary HP 64768 emulator configuration items.

Syntax



Function

The **<CONFIG_ITEMS>** are the HP 64768 specific configuration items which can be displayed/modified using the **cf** (emulator configuration) command. If the "=" portion of the syntax is not used, the current value of the configuration item is displayed.

Parameters

hold Respond to Target Hold . This configuration item allows you to specify whether or not the emulator accepts hold signal generated by the target system.

Setting **hold** equal to **dis** specifies that the emulator ignores hold signal from target system completely.

Setting **hold** equal to **en** specifies that the emulator accepts hold signal. When the hold is accepted, the emulator will respond as actual microprocessor.

nmi Enable/disable user NMI. This configuration item allows you to specify whether user NMI is accepted or ignored by the emulator.

To accept user NMI, set **nmi** equal to **en**. To ignore user NMI, set **nmi** to **dis**.

Note

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

rrt Restrict to Real-Time Runs. This configuration item allows you to 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.

To restrict execution to real-time, set **rrt** equal to **en**. To allow breaks to the monitor during program

execution, set **rrt** equal to **dis**. When runs are restricted to real-time, commands which access target system resources (display registers, display/modify target system memory or I/O) are not allowed.

rst Respond to Target Reset. This configuration item allows you to specify whether or not the emulator respond target system reset while running in user program or waiting for target system reset.

While running in background monitor, the HP 64768 emulator ignores target system reset completely independent on this setting.

Specifying "**cf rst=en**", 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 in the same manner as actual microprocessor after reset is inactivated.

You can ignore reset from target system completely by specifying "**cf rst=dis**". In this configuration emulator ignore reset from target system.

Note

When you use the **r rst** (run from reset) command in-circuit to run form processor reset after the target reset input, you must use "**cf rst=en**" configuration setting.

tdma

Trace Internal DMA cycles. This configuration item allows you to specify whether or not the analyzer trace the HP 64768 emulation processor's internal DMA cycles.

Setting **tdma** equal to **en** specifies that the analyzer will trace the HP 64768 internal DMA cycles.

Setting **tdma** equal to **dis** specifies that the analyzer will not trace the HP 64768 internal DMA cycles.

clk

Clock Source. This configuration item allows you to specify whether the emulator clock source is to be internal (**int**, provided by the emulator) or external (**ext**, provided by the target system).

In the HP 64768A/B emulators, the internal clock speed is 12.5 MHz (system clock).

The HP 64768A emulator will operate at external clock speed from 4 to 25 MHz (entered clock).

The HP 64768B emulator will operate at external clock speed from 4 to 32 MHz (entered clock).

The HP 64768 emulator is reset often after specifying this configuration item.

dsize

Data Bus size. This configuration item allows you to specify whether the data bus size is to be 8(**8**,data bus size is 8 bits) or 16(**16**,data bus size is 16 bits).

The HP 64768 emulator is reset state after specifying this configuration item.

Note

The HP 64768 emulator operates in accordance with this configuration instead of D8/16 signal from target system. D8/16 signal from target system is ignored.

mne Type of Mnemonic. This configuration item allows you to specify the type of mnemonic that are used by display memory and display trace command.

Setting **mne** equal to **70433** specifies that emulator will display memory in uPD70433 mnemonic.

Setting **mne** equal to **8086** specifies that emulator will display memory in iAPX86/10(8086) mnemonic.

Note

The instruction that is not included iAPX86/10 mnemonic is displayed with uPD70433 mnemonic, even if you specify **mne=8086**.

mon Monitor Options. This configuration item is used to select the type of monitor to be used by the emulator.

If **bg** (background monitor) is selected, all monitor functions are performed in background.
If **fg** (foreground monitor) is selected, all monitor functions are performed in foreground. You should use the 20 bits physical address expression to locate the foreground monitor on a 2K byte boundary

The HP 64768 emulator is reset after specifying this configuration item.

Note

The start address of the foreground monitor should not be located at a base address 0 or 0ff800 hex;because the 70433 microprocessor's vector table or SFR are located respectively. Refer to the "Using the Optional Foreground Monitor" appendix in this manual.

rad Physical to Logical Run Address Conversion. This configuration item allows you to specify the default method in which the emulation system will convert physical addresses specified in run and step commands to logical addresses.

Setting **rad** equal to **maxseg** specifies that the low nibble of the physical address become the offset value; the high four nibbles become the segment value.

Setting **rad** equal to **minseg** specifies that the low four nibbles of the physical address become the offset value; the high nibble and three hex zeros will become the segment value.

Setting **rad** equal to **curseg** specifies that the value which is entered in a run or step command will become the offset value.

rdy Allow Target Ready Signals to Insert Wait States. This configuration item allows you to specify whether the emulator should honor target system ready signals on accesses to emulation memory.

Setting **rdy** equal to **lk** specifies that target ready signals be honored on emulation memory accesses. Setting **rdy** equal to **unlk** specifies that target ready signals be ignored on emulation memory accesses.

rsp Specify the Stack Location. This configuration item allows you to specify the stack location value ; (SS:SP) after the emulation reset. The stack segment (SS) and stack pointer (SP) will be set on entrance to the emulation monitor initiated RESET state.
You should use the logical address expression to locate the stack area.

Note

When you are using the foreground monitor, this address should be defined in an emulation memory or a target system RAM area.

trfsh Trace Refresh cycles. This configuration item allows you to specify whether or not the analyzer trace the HP 64768 emulation processor's refresh cycles.

Setting **trfsh** equal to **en** specifies that the analyzer will trace refresh cycles.

Setting **trfsh** equal to **dis** specifies that the analyzer will not trace refresh cycles.

Defaults

The default values of HP 64768 emulator configuration items are listed below.

```
cf clk=int
cf dsize=16
cf hold=en
cf mne=70433
cf mon=bg
cf nmi=en
cf rad=minseg
cf rdy=lk
cf rrt=dis
cf rsp=0000:8000
cf rst=en
cf tdma=en
cf thold=en
cf trfsh=en
mo -aw -dw
```

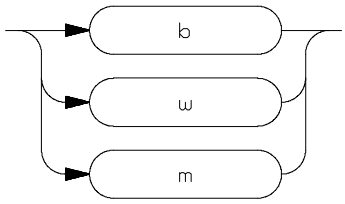
Related Commands You can get an on line help information for particular configuration items by typing:

R> **help cf <CONFIG_ITEM>**

DISPLAY_MODE

Summary Specify the memory display format or the size of memory locations to be modified.

Syntax



Function The **<DISPLAY_MODE>** specifies the format of the memory display or the size of the memory which gets changed when memory is modified.

Parameters

b	**Byte.** Memory is displayed in a byte format, and when memory locations are modified, bytes are changed.
w	**Word.** Memory is displayed in a word format, and when memory locations are modified, words are changed.
d	**Double-word.** Memory is displayed in a double-word format, and when memory locations are modified, double-words are changed.
m	**Mnemonic.** Memory is displayed in mnemonic format; that is, the contents of memory locations are inverse-assembled into mnemonics and operands. When memory locations are modified, the last non-mnemonic display mode specification is used.

You cannot specify this display mode in the **ser** (search memory for data) command.

Defaults

At powerup or after init,in the HP 64768 Emulator, the **<ACCESS_MODE>** and **<DISPLAY_MODE>** are **b**.

Display mode specifications are saved; that is, when a command changes the display mode, the new display mode becomes the current default.

Related Commands

mo (specify access and display modes)

m (memory display/modify)

io (I/O display/modify)

ser (search memory for data)

REGISTER CLASS and NAME

Summary 70433 register designator. All available register class names and register names are listed below.

<REG_CLASS>

<REG_NAME>	Description
***(All basic registers)**	
aw, bw	BASIC registers.
cw, dw	
bp, ix, iy	
ds0, ds1,	
ds2, ds3	
ss, sp	
pc, ps, psw	

port (Port registers)

p0	Port 0	
p1	Port 1	(Read Only)
p2	Port 2	
p3	Port 3	
p4	Port 4	
p5	Port 5	
p6	Port 6	(Read Only)
p7	Port 7	
p8	Port 8	
pm0	Port 0 mode	
pm2	Port 2 mode	
pm3	Port 3 mode	
pm4	Port 4 mode	
pm5	Port 5 mode	
pm7	Port 7 mode	
pm8	Port 8 mode	
pmc2	Port 2 mode control	
pmc3	Port 3 mode control	
pmc4	Port 4 mode control	
pmc5	Port 5 mode control	
pmc7	Port 7 mode control	
pmc8	Port 8 mode control	
prdc	Port read control	

rop (Real-time Output port registers)

rtpc	Real-time output port control
rtpd	Real-time output port delay display
p7l	Port 7 buffer(Low)
p7h	Port 7 buffer(high)
rtp	Real-time output port

tim (Timer registers)

tm0	Timer 0	
tm1	Timer 1	
tm2	Timer 2	
tm3	Timer 3	
ct00	Timer capture 00	
ct01	Timer capture 01	
ct10	Timer capture 10	
cm00	Timer compare 00	
cm01	Timer compare 01	
cm10	Timer compare 10	
cm11	Timer compare 11	
cm20	Timer compare 20	
cm21	Timer compare 21	
cm22	Timer compare 22	
cm23	Timer compare 23	
cm30	Timer compare 30	
cm31	Timer compare 31	
tmc	Timer control	
toc	Timer output control	
stc	Software timer counter	(Read Only)
stmc	Software timer counter compare	

pwmu (PWM uint registers)

pwm	PWM
pwmc	PWM control

dma (DMA registers)

dmam0	DMA mode 0
dmam1	DMA mode 1
dmac0	DMA control 0
dmac1	DMA control 1
tc0	Terminal counter 0
tc1	Terminal counter 1
tcm0	Terminal counter modulo 0
tcm1	Terminal counter modulo 1
mar0	DMA memory address 0
mar1	DMA memory address 1
udc0	DMA up/down counter 0
udc1	DMA up/down counter 1
dcm0	DMA compare 0
dcm1	DMA compare 1
dptc0	DMA read/write pointer 0
dptc1	DMA read/write pointer 1
dmas	DMA status

pi (Parallel I/F registers)

pab	Parallel interface buffer
pac0	Parallel interface control 0
pac1	Parallel interface control 1
pas	Parallel interface status
pai1	Parallel interface acknowledge interval 1 (Write Only)
pai2	Parallel interface acknowledge interval 2 (Write Only)

ad (Analog-Digital conversion registers)

adm	A/D convertor mode	
adcr0	A/D conversion result 0	(Read Only)
adcr1	A/D conversion result 1	(Read Only)
adcr2	A/D conversion result 2	(Read Only)
adcr3	A/D conversion result 3	(Read Only)

uart (UART registers)

asp	Protocol select	
uartm0	UART mode 0	
uartm1	UART mode 1	
uarts0	UART status 0	
uarts1	UART status 1	
rxb0	Receive buffer 0	(Read Only)
rxb1	Receive buffer 1	(Read Only)
txb0	UART transfer buffer 0	(Write Only)
tx b1	UART transfer buffer 1	(Write Only)
prs0	Prescaler 0	
prs1	Prescaler 1	
rxbrg0	Receive baud rate generator 0	
rxbrg1	Receive baud rate generator 1	
txbrg0	Transfer baud rate generator 0	
txbrg1	Transfer baud rate generator 1	

csi (Clocked serial I/F registers)

asp	Protocol select	
csim0	Clocked serial interface mode 0	
csim1	Clocked serial interface mode 1	
sbic0	SBI control 0	
rxb0	Receive buffer 0	
rxb1	Receive buffer 1	
sio0	Clocked serial I/O shift 0	(Write Only)
sio1	Clocked serial I/O shift 1	(Write Only)
prs0	Receive baud rate generator 0	
prs1	Receive baud rate generator 1	
txbrg0	Transfer baud rate generator 0	
txbrg1	Transfer baud rate generator 1	

proc (Processor status registers)

stbc	Standby control
prc	Processor control
pwc0	Programmable wait control 0
pwc1	Programmable wait control 1
rfm	Refresh mode
mbc	Memory block control
wdm	Watchdog timer mode

intc (Interrupt control registers)

imc	Interrupt mode control	
mk0	Interrupt mask flag 0	
mk1	Interrupt mask flag 1	
ic09	Interrupt request control 09	
ic10	Interrupt request control 10	
ic11	Interrupt request control 11	
ic12	Interrupt request control 12	
ic13	Interrupt request control 13	
ic14	Interrupt request control 14	
ic16	Interrupt request control 16	
ic17	Interrupt request control 17	
ic18	Interrupt request control 18	
ic19	Interrupt request control 19	
ic20	Interrupt request control 20	
ic21	Interrupt request control 21	
ic22	Interrupt request control 22	
ic23	Interrupt request control 23	
ic24	Interrupt request control 24	
ic25	Interrupt request control 25	
ic26	Interrupt request control 26	
ic27	Interrupt request control 27	
ic28	Interrupt request control 28	
ic29	Interrupt request control 29	
ic30	Interrupt request control 30	
ic31	Interrupt request control 31	
ic32	Interrupt request control 32	
ic36	Interrupt request control 36	
ic37	Interrupt request control 37	
i spr	In-service priority	(Read Only)
intm	External interrupt mode	

bank<N> (register bank)

ps_<N>	ps of register bank <N>
pc_<N>	pc of register bank <N>
psw_<N>	psw of register bank <N>
aw_<N>	aw of register bank <N>
bw_<N>	bw of register bank <N>
cw_<N>	cw of register bank <N>
dw_<N>	dw of register bank <N>
sp_<N>	sp of register bank <N>
bp_<N>	bp of register bank <N>
ix_<N>	ix of register bank <N>
iy_<N>	iy of register bank <N>
ds0_<N >	ds0 of register bank <N>
ds1_<N>	ds1 of register bank <N>
ds2_<N>	ds2 of register bank <N>
vpc_<N>	vpc of register bank <N>
ss_<N>	ss of register bank <N>

Function

The **<REG_CLASS>** names may be used in the **reg**(register) command to display a class of 70433 registers.

The **<REG_NAME>** names may be used with the **reg** command to either display or modify the contents of 70433 registers.

Refer to your 70433 use's manual for complete details on the use of the 70433 registers.

Related Commands

reg (register display/modify)

B

Using the Optional Foreground Monitor

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

The monitor programs named **fmon70433.s** is to be assembled and linked into target program by the HP 64873 Cross Assembler/Linker.

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. 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, while 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 applications that rely on the microprocessor for real-time, non-intrusive support. Also, the background monitor code resides in emulator firmware and can't be modified to handle special conditions.

Foreground Monitors

A foreground monitor may be required for more interrupt intensive applications. A foreground monitor is a block of code that runs in the same memory space as your program. You link this monitor into your code so that when control is passed to monitor program, the emulator can still service real-time events, such as interrupts or watchdog timers. 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 "Emulation topics" chapter and the examples in this appendix).

An Example Using the Foreground Monitor

In the following example, we will illustrate how to link a foreground monitor. By using the emulation analyzer, we will also show how the emulator switches from state to state using a foreground monitor.

For this example, We will locate the monitor at 1000 hex; the sample program will be located at 800 hex with its data at 600 hex and its common at 400 hex.

Modify EQU Statement

To use the monitor, you must modify the EQU statement near the top of the monitor some code to point to the segment start address where the monitor will be loaded. In this example, the monitor will be located at 1000 hex, so the modified EQU statement looks like this:

```
MONSEGMENT      EQU     0100H
```

Notice that the EQU statement is indented from the left margin; if it is not indented, the assembler will attempt to interpret the EQU as a label and will generate an error when processing the address portion of the statement. You can load the monitor at any base address on a 2k byte boundary.

Note

You should not load the foreground monitor at the base address 0 or 0ff800 hex, because the 70433 microprocessor's vector table and SFR are located respectively.

Assemble and Link the Monitor

You can assemble and link the foreground monitor program with the following commands by using the HP 64853 Cross Assembler/Linker:

$ **asv20 -f optimize fmon70433** <RETURN>
$ **ldv20 -o fmon70433.X fmon70433.o** <RETURN>

Initialize the Emulator

To initialize the emulator to a known state for this example, type:

M> **init -p**

Configure the Emulator

You need to tell the emulator that you will be using a foreground monitor and allocate the memory space for the monitor. This is all done with one configuration command. To locate the monitor on a 2k boundary starting at 1000 hex, type:

R> **cf mon=fg..001000**

To see the new memory mapper term allocated for the foreground monitor, type:

R> **map**

```
# remaining number of terms   : 15
# remaining emulation memory  : 1f800h bytes
map 0001000..00017ff  eram        # term 1
map  other tram
```

Notice that a 2k byte block from 1000 through 17ff hex was mapped.

Now, you need to map memory space for the sample program. Let's map the memory from 0 through 4ff hex to emulation RAM and map the memory from 600 through 9ff hex to emulation ROM.

R> **map 0..4ff eram**
R> **map 600..9ff erom**

Load the Foreground Monitor

Now it's time to load the sample program and monitor. In the example shown, we're loading the program from a host with the emulator in Transparent Configuration. If you're using the standalone configuration with a data terminal, you will need to enter the data using the **m** command. (You can get the data from your assembly listings.) Load the program by typing:

R> **load -hbs "transfer -tb fmon70433.X"**

```
###############
```

Load the Sample Program

Assuming the sample program has been assembled and linked as shown in "Getting Start" chapter, you can load the sample program by typing:

R> **load -hbs "transfer -tb cmd_rds.X"**

```
#####
```

Disable Tracing Refresh Cycle

If you wish to disable the analyzer from tracing refresh cycles, you can use the **cf trfsh** command ; the refresh cycles are not detected by the analyzer. Type:

M> **cf trfsh=dis**

Set Analyzer Master Clock Qualifiers

We want to view the transitions made between the different emulator states; reset to break, break to run, run to break. 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. We can do this by modifying the emulation analyzer's master clock qualifier to include tracing of background code. To see the initial clock qualifier, type:

M> **tck**

```
tck -r L -u -s S
```

Modify this as follows:

M> **tck -r L -ub -s S**

Now, reset the processor so we can make the first measurement from a known state:

M> **rst**

Reset to Break

We want to see the monitor's transition from the reset state to running in the foreground monitor. Since the foreground monitor occupies the address range from 1000 through 17ff hex, we can simply trigger on any access to that range:

R> **tg addr=1000..17ff**

We also want see the states leading up to the transition between reset and foreground monitor execution. We can position the trigger so that there are 20 states **before** the trigger as follows:

R> **tp -b 20**

Start the measurement:

R> **t**

```
Emulation trace started
```

Now, break the emulator into the monitor:

R> **b**

Display 20 disassembled states of the trace from the top the trace:

M> **tl -td 20**

```
  Line   addr,H  70433 mnemonic,H                      count,R  seq
 -----   ------  --------------------------------  ---------  ---
    -7   0ffff2    0007  fetch                 BGM        ---   .
    -6   000020    0000  write mem             BGM    1.920 uS   .
    -5   000022    ffff  write mem             BGM    1.920 uS   .
    -4   000024    f002  write mem             BGM    1.920 uS   .
    -3   000008    0200  read mem              BGM    3.840 uS   .
    -2   0ffff4    fe01  fetch                 BGM    1.920 uS   .
    -1   00000a    0100  read mem              BGM    1.920 uS   .
     0   001200    a32e  fetch                 BGM    2.880 uS   +
     1   001202    002c  fetch                 BGM    3.840 uS   .
     2   001200  MOV PS:WORD PTR 002c,AW              0.320 uS   .
     3   001204    892e  fetch                 BGM    1.600 uS   .
     4   00102c    f080  write mem             BGM    1.920 uS   .
     5   001206    2e2e  fetch                 BGM    1.920 uS   .
     6   001204  MOV PS:WORD PTR 002e,BP              0.320 uS   .
     7   001208    2e00  fetch                 BGM    3.520 uS   .
     8   00120a    20a1  fetch                 BGM    1.920 uS   .
     9   001209  MOV AW,PS:WORD PTR 0020              0.640 uS   .
    10   00102e    00fa  write mem             BGM    1.280 uS   .
    11   00120c    2e00  fetch                 BGM    1.920 uS   .
    12   001020    0000  read mem              BGM    3.840 uS   .
```

At line -6, the processor began executing code; it executed in the background monitor. To see the transition from background execution to foreground monitor program execution, type:

M> **tl 45..65**

```
 Line   addr,H  70433 mnemonic,H                       count,R  seq
-----   ------  ---------------------------------  ---------  ---
   45   001230    00a1  fetch                 BGM    1.920 uS   .
   46   00122f  MOV AW,PS:WORD PTR 0000              0.640 uS   .
   47   001022    0100  write mem             BGM    1.280 uS   .
   48   001232    0f00  fetch                 BGM    3.840 uS   .
   49   001234    0427  fetch                 BGM    1.920 uS   .
   50   001000    0000  read mem              BGM    1.920 uS   .
   51   001233  illegal opcode, data = 0f 27         0.320 uS   .
   52   001236    0000  fetch                 BGM    1.600 uS   .
   53   000024    f002  read mem              BGM    1.920 uS   .
   54   000022    0100  read mem              BGM    3.840 uS   .
   55   001238    0400  fetch                 BGM    1.920 uS   .
   56   000020    0332  read mem              BGM    1.920 uS   .
   57   001332    f62e  fetch                        2.880 uS   .
   58   001334    1906  fetch                        3.840 uS   .
   59   001332  TEST PS:BYTE PTR 0019,01             0.320 uS   .
   60   001336    0100  fetch                        1.600 uS   .
   61   001338    c62e  fetch                        1.920 uS   .
   62   00133a    1b06  fetch                        1.920 uS   .
   63   001019    01xx  read mem                     1.920 uS   .
   64   001338  MOV PS:BYTE PTR 001b,02              0.320 uS   .
   65   00133c    0200  fetch                        3.520 uS   .
```

The foreground monitor start at states 57.

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. Type:

M> **tg addr=800**

We will leave the trigger position where it was for the last measurement(20 states are retained before the trigger position). Start the measurement:

M> **t**

```
Emulation trace started
```

Now, run the sample program:

M> **r 800**

Display trace states from -15 to +5 in inverse-assembled form as follows:

U> **tl -d -15..5**

```
 Line   addr,H  70433 mnemonic,H                      count,R  seq
-----   ------  ---------------------------------  ---------  ---
  -15   00151b  MOV DS0,WORD PTR 0046                0.320 uS   .
  -14   00151e    2e00  fetch                       1.600 uS   .
  -13   001520    06c6  fetch                       1.920 uS   .
  -12   001046    0060  read mem                   1.920 uS   .
  -11   00151f  MOV PS:BYTE PTR 0016,80             0.320 uS   .
  -10   001522    0016  fetch                       3.520 uS   .
   -9   001524    cf80  fetch                       1.920 uS   .
   -8   001526    0e8b  fetch                       1.920 uS   .
   -7   001525  RETI                                0.320 uS   .
   -6   001016    xx80  write mem                   1.600 uS   .
   -5   0004fa    0800  read mem                    3.840 uS   .
   -4   001528    0014  fetch                       1.920 uS   .
   -3   0004fc    0000  read mem                    1.920 uS   .
   -2   00152a    3e8e  fetch                       1.920 uS   .
   -1   0004fe    f046  read mem                    1.920 uS   .
    0   000800    60b8  fetch                       3.840 uS   +
    1   000800  MOV AW,0060                         0.320 uS   .
    2   000802    8e00  fetch                       1.600 uS   .
    3   000804    b8d8  fetch                       1.920 uS   .
    4   000803  MOV DS0,AW                          0.320 uS   .
    5   000806    0040  fetch                       1.600 uS   .
```

At state -7 in the trace listing, the processor executed the **RETI** instruction to transfer execution to the user program at state 0.

Note

As you can see in the trace list, user stack pointer is used when context is changed from foreground monitor to user program, or from user program to foreground monitor program. If you are configuring the emulator to background monitor, the user stack is not used.

User Program Run to Break

You can trace the execution from the user program run to the foreground monitor due to a break condition by setting as follows:

U> **tg stat=bg**

Start the measurement:

U> **t**

```
Emulation trace started
```

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

U> **b**

Now, display trace states from -10 to +10 in disassembled form as follows:

M> **tl -10..10**

```
  Line   addr,H  70433 mnemonic,H                        count,R  seq
 -----   ------  ----------------------------------  ---------  ---
   -10   00081b  BE/Z 00815                           0.320 uS   .
    -9   00081e    7441  fetch                       1.600 uS   .
    -8   000815    26xx  fetch                       1.920 uS   .
    -7   000816    00a0  fetch                       1.920 uS   .
    -6   000815  MOV AL,DS1:BYTE PTR 0000            0.640 uS   .
    -5   000818    3c00  fetch                       3.200 uS   .
    -4   00081a    7400  fetch                       1.920 uS   .
    -3   000400    xx00  read mem                    1.920 uS   .
    -2   000815    xxxx  intack                      0.320 uS   .
    -1   00081c    3cf8  fetch                       1.600 uS   .
     0   000020    0819  write mem             BGM   3.840 uS   +
     1   000022    0000  write mem             BGM   1.920 uS   .
     2   000024    f046  write mem             BGM   1.920 uS   .
     3   000008    0200  read mem              BGM   1.920 uS   .
     4   00000a    0100  read mem              BGM   3.840 uS   .
     5   001200    a32e  fetch                 BGM   2.880 uS   .
     6   001202    002c  fetch                 BGM   1.920 uS   .
     7   001200  MOV PS:WORD PTR 002c,AW            0.320 uS   .
     8   001204    892e  fetch                 BGM   1.600 uS   .
     9   00102c    0000  write mem             BGM   3.840 uS   .
    10   001206    2e2e  fetch                 BGM   1.920 uS   .
```

At state 0 of the trace list, the processor entered the background monitor to make the transition. And actual foreground monitor program start at after several background monitor execution. To see the starting point of foreground monitor, type:

M> **tl 55..75**

```
 Line   addr,H  70433 mnemonic,H                        count,R  seq
-----   ------  ---------------------------------  ---------  ---
   55   001000    0000  read mem               BGM    1.920 uS   .
   56   001233  illegal opcode, data = 0f 27          0.320 uS   .
   57   000024    f046  read mem               BGM    3.520 uS   .
   58   001236    0000  fetch                  BGM    1.920 uS   .
   59   000022    0100  read mem               BGM    1.920 uS   .
   60   001238    0400  fetch                  BGM    1.920 uS   .
   61   000020    0332  read mem               BGM    1.920 uS   .
   62   001332    f62e  fetch                         3.840 uS   .
   63   001334    1906  fetch                         1.920 uS   .
   64   001332  TEST PS:BYTE PTR 0019,01              0.320 uS   .
   65   001336    0100  fetch                         1.600 uS   .
   66   001338    c62e  fetch                         3.840 uS   .
   67   001019    00xx  read mem                      2.560 uS   .
   68   00133a    1b06  fetch                         1.920 uS   .
   69   001338  MOV PS:BYTE PTR 001b,02               0.320 uS   .
   70   00133c    0200  fetch                         1.600 uS   .
   71   00133e    4475  fetch                         3.840 uS   .
   72   00133e  BNE/Z 01384                           0.960 uS   .
   73   00101b    02xx  write mem                     1.600 uS   .
   74   001340    a12e  fetch                         1.920 uS   .
   75   001342    0026  fetch                         1.920 uS   .
```

At state 62, the foreground monitor program starts.

Single Step and Foreground Monitors

To use the "step" command to step through processor instructions with the foreground monitor listed in this chapter, you **must** modify the processor's interrupt vector table. The entry that you **must** modify is the "BRK flag" interrupt vector, located at 4H thru 7H. The "BRK flag" interrupt vector must point to the "SINGLE_STEP_ENTRY" in the foreground monitor. The address of the "SINGLE_STEP_ENTRY" is 300H plus the beginning of the foreground monitor.

Software Breakpoint and Foreground Monitors

To use the software breakpoint with the foreground monitor listed in this chapter, you must modify the processor's interrupt vector table. The entry that you must modify is the "BRK 3"interrupt vector, located at 0CH thru 0FH. The PC portion of the "BRK 3" interrupt vector must be 1234H and the PS portion must be 5678H.

Limitations of Foreground Monitors

Synchronized measurements

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

Instruction Using BRK flag

If user program includes instruction using the BRK flag(in PSW register), you can not use foreground monitor because the foreground monitor uses the BRK flag in step command.

Stepping

You can not use step command in the following instruction.

HALT/STOP
POP PSW
BRK 3/BRK imm8/BRKV
CHKIND
FPO
TSKSW/BRKCS
RETRBI

Break from Halt/Stop state

When the processor is in halt or stop state, the program counter(PC) indicates the next address of HALT or STOP instruction. If you use commands which require temporary break(display/modify register,or display/modify target system memory or I/O), the program will run from the address that PC indicates.

Notes

Index