HP 64791/2

70208H/70216H Emulator Terminal Interface

User's Guide

HEWLETT PACKARD

HP Part No. 64791-97009
Printed in U.S.A.
July 1994

Edition 4

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	64791-97000, August 1991
Edition 2	64791-97003, November 1991
Edition 3	64791-97006, December
Edition 4	**64791-97009, July 1994**

Using this Manual

This manual will show you how to use the following emulators with the firmware resident Terminal Interface.

- HP 64791A 70208 emulator
- HP 64792A 70216 emulator
- HP 64791B 70208H emulator
- HP 64792B 70216H emulator

For the most part, the 70208/70208H/70216/70216H emulators all operate the same way. Differences between the emulators are described where they exist. These 70208, 70208H, 70216 and 70216H emulators will be referred to as the "70216 emulator" in this manual where they are alike. In the specific instances where 70208, 70208H and 70216H emulator differs from the 70216 emulator, it will be referred as the "70208 emulator", "70208H emulator" and "70216H emulator".

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 70216 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.*

Organization

Chapter 1 **Introduction to the 70216 Emulator.** This chapter briefly introduces you to the concept of emulation and lists the basic features of the 70216 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 **70216 Emulator Specific Command Syntax**. This appendix describes the command syntax which is specific to the 70216 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.

Appendix C **Specific Error Messages.** This appendix describes the error messages which is specific to the 70216 emulator.

Notes

Contents

4 In-Circuit Emulation Topics

A 70216 Emulator Specific Command Syntax

B Using the Optional Foreground Monitor

Illustrations

Tables

1

Introduction to the 70216 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 70216 emulator is designed to replace the 70216 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.

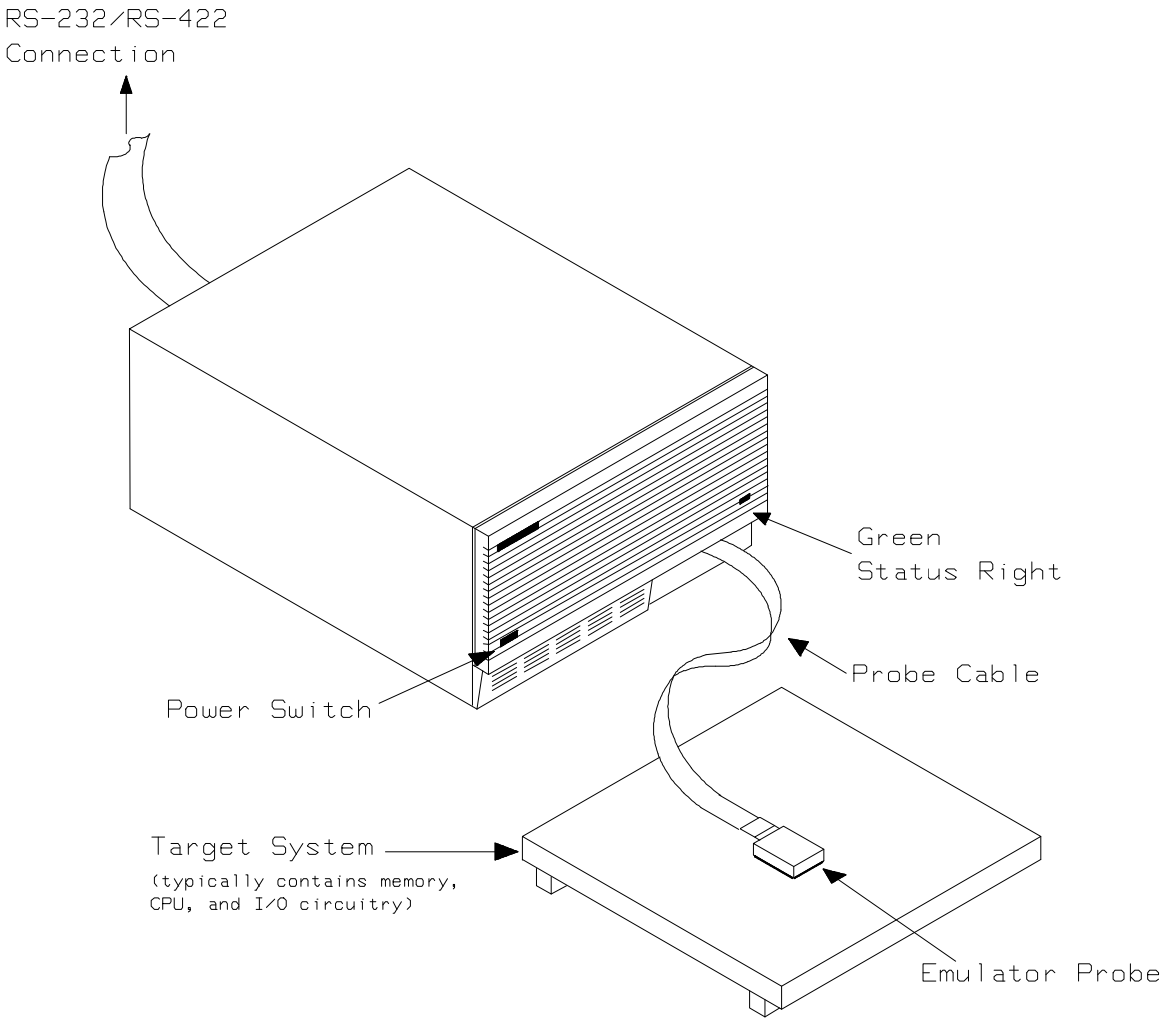


Figure 1-1. HP 64792 Emulator for uPD70216

1-2 Introduction

Features of the 70216 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 HP 64791/2 emulator supports the following packages of microprocessor.

Model No.	Microprocessor	Package
HP 64791A	uPD70208	68-pin PLCC 68-pin PGA
HP 64792A	uPD70216	68-pin PLCC 68-pin PGA
HP 64791B	uPD70208H	68-pin PLCC 68-pin PGA
HP 64792B	uPD70216H	68-pin PLCC 68-pin PGA

The HP 64791/2 emulator probe has a 68-pin PLCC connector. When you use 68-pin PGA type microprocessor, you must use with PLCC to PGA adapter; refer to the "In-Circuit Emulation Topics" chapter in this manual.

Clock Speeds

The 70208 and 70216 emulator runs with an internal clock speed of 8MHz (system clock), or with target system clocks from 2 to 10 MHz.

The 70208H and 70216H emulator runs with an internal clock speed of 16 MHz (system clock) or with target system clocks from 1 to 16 MHz.

Emulation memory

The HP 70216 emulator is used with one of the following Emulation Memory Cards.

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

When you use the HP 64729, You can only use 1M byte for emulation memory.
You can define up to 16 memory ranges (at 128 byte boundaries and at least 128 byte in length). You can characterize memory ranges as emulation RAM, emulation ROM, target system RAM, target system ROM, or as guarded memory. The emulator generates an error message when accesses are made to guarded memory locations. 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 70216 emulator is used with one of the following analyzers which allows you to trace code execution and processor activity.

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

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

Registers

You can display or modify the 70216 internal register contents.

Single-Step

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

Breakpoints

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

You can also define software breakpoints in your program. The emulator uses the BRK 3 instruction(CC hex) as software breakpoint interrupt instruction. When you define a software breakpoint, the emulator places the breakpoint interrupt instruction (CC hex) at the specified address; after the breakpoint interrupt instruction causes emulator execution to break out of your program, the emulator replaces the original opcode.

Reset Support

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

Configurable Target System Interface

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

Foreground or Background Emulation Monitor

The emulation monitor is a program that is executed by the emulation processor. It allows the emulation controller to access target system resources. For example, when you display target system memory, it is the monitor program that executes 70216 instructions which read the target memory locations and send their contents to the emulation controller.

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

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

Real-Time Operation

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

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

Easy Products Upgrades

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

Limitations, Restrictions

DMA Support

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

TC bit of DMA Status Register

While using the uPD71071 or the uPD71037 DMA mode on the 70208H emulator, or using the uPD71037 DMA mode on the 70216H emulator, when the emulator read the other than DST register, the TC bit of the DST is reset. If you know the DMA Status, you have to use the count register in the place of the TC bit.

User Interrupts

If you use the background monitor, NMI and INTP1-7 from the target system are suspended until the emulator goes into foreground operation.

Interrupts While Executing Step Command

While executing user program code in stepping in the foreground monitor, interrupts are accepted if they are enabled in the foreground monitor program. When using the background monitor the emulator will fail to step, if the interrupts are acknowledged before stepping user program code.

Evaluation chip

Hewlett-Packard makes no warranty of the problem caused by the 70208/70208H/70216/70216H 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 64792 emulator for the 70216 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:

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 HP64700 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.

```
$MODV20

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 Listing (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", an invalid command has been entered, and execution is transferred to the instructions at **Cmd_I**.

The instructions at **Cmd_A**, **Cmd_B**, and **Cmd_I** each 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** which writes the appropriate message 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.

The program emulates a primitive command interpreter.

Using the "help" Facility

The HP 64700 Series emulator's Terminal Interface provides an excellent help facility to provide you with quick information on 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

  hidden    - special use commands normally hidden from user
  sys       - system commands
  emul      - emulation commands
  trc       - analyzer trace commands
  xtrc      - external trace analysis 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 if you are not familiar with it.

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 -----
Memory address -- 32 bit (seg:off) logical or 20 bit physical address
IO address -- 16 bit address

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

--- Analyzer STATUS Field Equates ---
  fetch   - program fetch           exec     - execute instruction
  read    - read                    write    - write
  mem     - memory access           cpu      - cpu cycle
  extio   - external I/O access     intio    - internal I/O access
  haltack - halt acknowledge        holdack  - hold acknowledge
  intack  - interrupt acknowledge   refresh  - refresh cycle
  grd     - guarded memory access   rom      - rom access
  dma     - DMA memory access       casdma   - cascaded DMA cycle
  em80    - 8080 emulation mode     native   - native mode
  ds0     - ds0 use cycle           ds1      - ds1 use cycle
  ss      - ss use cycle            ps       - ps use cycle
  usr     - user cycle              mon      - monitor cycle
```

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 status of the emulator.

```
U>es
N70216--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 mapper terms, you will find that the emulation memory contents are the same.

Other Types of Initialization

There are two options to the **init** command which specify other types of initializations. The **-p** option specifies a powerup initialization, also known as a cold start initialization. The cold start initialization sequence includes the emulator, analyzer, system controller, and communications port initialization; additionally, performance verification tests are run.

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

Mapping Memory

Depending on the memory board, emulation memory consists of 128K , 512K or 1M bytes, mappable in 128 byte blocks. The monitor occupies 4K bytes, leaving 124K , 508K or 1020K bytes of emulation memory which you may use. The emulation memory system does not introduce wait states.

Note

When you use the i8087 coprocessor on your target system connected to 70216 microprocessor, the i8087 can access 70216 emulation memory on coprocessor memory read/write cycles.

In this case, you should reset the target system to connect the 70216 emulator to the i8087 coprocessor before starting emulation session.

Refer to "In-Circuit Emulation Topics" chapter for more information about accesses to emulation memory.

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 accesses to emulation memory are not allowed. Target system devices that take control of the bus, except i8087 coprocessor (for example, external DMA controllers), cannot access emulation memory.

Blocks of memory can also be characterized as guarded memory. Guarded memory accesses will generate "break to monitor" requests. Writes to ROM will generate "break to monitor" requests if the **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  : 7f000h 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. The linker load map listing will show what memory locations your program will occupy in memory. A linker load map listing for the sample program is shown below.

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

HP64873-19005 02.30 01Aug90  Copr. HP 1990
Command line: ldv20  -c cmd_rds.k -L

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

OUTPUT MODULE NAME:   cmd_rds
OUTPUT MODULE FORMAT: HP-OMF 86

MODULE SUMMARY
--------------

MODULE   SEGMENT  CLASS    START  END

cmd_rds  /users...../cmd_rds.o
         CODE     CODE     00400  00450
         COMN     COMN     00800  008FF
         DATA     DATA     00600  00635

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

SEGMENT  CLASS    GROUP    START  END    LENGTH  ALIGNMENT     COMBINE

CODE     CODE              00400  00450  00051   Paragraph     Public
DATA     DATA              00600  00635  00036   Paragraph     Public
COMN     COMN              00800  008FF  00100   Paragraph     Common
??SEG                      00000  00000  00000   Paragraph     Public
??DATA1  ??INIT            00000  00002  00003   Byte          Common


START ADDRESS:   00040:00000 -> 00400
```

From the load map listing, you can see that the sample program occupies three address range. The program area, which contains the opcodes and operands which make up the sample program, occupies locations 400 through 450 hex. The data area, which contains the ASCII values of the messages the program transfers, is occupies locations 600 through 635 hex. The destination area, which contains the command input byte and the locations of the message destination, occupies locations 800 through 8FF hex.

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

```
R>map 0..7ff erom
R>map 800..9ff eram
R>map
```

```
# remaining number of terms   : 14
# remaining emulation memory  : 7e600h bytes
map  000000..0007ff  erom    # term 1
map  000800..0009ff  eram    # term 2
map  other tram
```

When mapping memory for your target system programs, you may wish to characterize emulation memory locations containing programs and constants (locations which should not be written to) as ROM. This will prevent programs and constants from being written over accidentally, and will cause breaks when instructions 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 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). Configurations in which the emulator is connected to, and accessed from, a host computer are 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 get 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 400=0b8,0,0,8e,0d8,0b8,0,0,8e,0c0,8e,0d0,0bc,0,1
R>m -db 40f=26,0c6,6,0,0,0,90,26,0a0,0,0,3c,0,74,0f8
R>m -db 41e=3c,41,74,7,3c,42,74,0c,0eb,13,90
R>m -db 429=0b9,12,0,0be,0,0,0eb,10,90,0b9,12,0,0be,12,0,0be,7,90
R>m -db 43b=0b9,12,0,0be,24,0,0bf,1,0,0f3,0a4
R>m -db 446=32,0c0,0b9,21,0,2b,0cf,0f3,0aa,0eb,0be
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, but the same method (and a different **load** option) can be used to 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.

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.

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 400..44f

000400  -               MOV AW,#0060
000403  -               MOV DS0,AW | MOV AW,#0080
000408  -               MOV DS1,AW | MOV SS,AW | MOV SP,#010
00040f  -               MOV DS1:0000,#00
000415  -               NOP
000416  -               MOV AL,DS1:0000
00041a  -               CMP AL,#00
00041c  -               BE/Z 00416
00041e  -               CMP AL,#41
000420  -               BE/Z 00429
000422  -               CMP AL,#42
000424  -               BE/Z 00432
000426  -               BR SHORT 0043b
000428  -               NOP
000429  -               MOV CW,#0012
00042c  -               MOV IX,#0000
00042f  -               BR SHORT 00441
000431  -               NOP
000432  -               MOV CW,#0012
000435  -               MOV IX,#0012
000438  -               BR SHORT 00441
00043a  -               NOP
00043b  -               MOV CW,#0012
00043e  -               MOV IX,#0024
000441  -               MOV IY,#0001
000444  -               REP/E/Z MOVBKB
000446  -               XOR AL,AL
000448  -               MOV CW,#0021
00044b  -               SUB CW,IY
00044d  -               REP/E/Z STMB
00044f  -               BR SHORT 0040f
```

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 are trying to display have been mapped. If the memory map is not the problem, recheck the linker load map listing to verify that the absolute addresses of the program agree with the locations you are trying to display.

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 0:400
```

```
00000:00400  cmd_rds:Init     MOV AW,#0060
PC = 00000:00403
```

Note

There are a few cases in which the emulator can not step. Step command is not accepted between each of the following instructions and the next instruction. 1) Manipulation instructions for sreg: MOV sreg,reg16; MOV sreg,mem16; POP sreg. 2) Prefix instructions: PS:, SS:, DS0:, DS1:, REPC, REPNC, REP, REPE, REPZ, REPNE, REPNZ, BUSLOCK. 3) EI, RETI, DI.

Displaying Registers

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

```
M>reg *

reg ps=0000 pc=0403 psw=f002 aw=0060 bw=05d9 cw=0000 dw=0000 sp=0009 bp=0001
reg ix=0010 iy=0e01 ds0=0000 ds1=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 *Terminal Interface: User's Reference* for more information on command files.)

Combining Commands

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

```
M>s;reg

00000:00403  -                MOV DS0,AW | MOV AW,#0080
PC = 00000:00408
reg ps=0000 pc=0408 psw=f002 aw=0080 bw=05d9 cw=0000 dw=0000 sp=0009 bp=0001
reg ix=0010 iy=0e01 ds0=0060 ds1=0000 ss=0000
```

The sample above shows you that MOV DS0,AW and MOV AW,#0080H are executed by step command. Refer to the Note above.

Using Macros

Suppose you want to continue stepping through the program, 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 any other command.

```
M>st
```

```
# s ; reg
00000:00408  -                MOV DS1,AW | MOV SS,AW | MOV SP,#0100
PC = 00000:0040f
reg ps=0000 pc=040f psw=f002 aw=0080 bw=05d9 cw=0000 dw=0000 sp=0100 bp=0001
reg ix=0010 iy=0e01 ds0=0060 ds1=0080 ss=0080
```

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 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  cmd_rds:Read_Cmd MOV DS1:0000,#00
PC = 00000:00415
reg ps=0000 pc=0415 psw=f002 aw=0080 bw=05d9 cw=0000 dw=0000 sp=0100 bp=0001
reg ix=0010 iy=0e01 ds0=0060 ds1=0080 ss=0080
# s ; reg
00000:00415  -                NOP
PC = 00000:00416
reg ps=0000 pc=0416 psw=f002 aw=0080 bw=05d9 cw=0000 dw=0000 sp=0100 bp=0001
reg ix=0010 iy=0e01 ds0=0060 ds1=0080 ss=0080
# s ; reg
00000:00416  cmd_rds:Scan     MOV AL,DS1:0000
PC = 00000:0041a
reg ps=0000 pc=041a psw=f002 aw=0000 bw=05d9 cw=0000 dw=0000 sp=0100 bp=0001
reg ix=0010 iy=0e01 ds0=0060 ds1=0080 ss=0080
# s ; reg
00000:0041a  -                CMP AL,#00
PC = 00000:0041c
reg ps=0000 pc=041c psw=f046 aw=0000 bw=05d9 cw=0000 dw=0000 sp=0100 bp=0001
reg ix=0010 iy=0e01 ds0=0060 ds1=0080 ss=0080
```

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 800=41
```

To verify that 41H has been written to 800H, enter the following command.

```
M>m -db 800
000800..000800  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 800**. 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 800H, enter the following commands.

```
M>mo
mo -ab -db

M>mo -dw
M>m 800
000800..000800  0041
```

To continue the rest of program.

```
M>r
U>
```

Display the **Msg_Dest** memory locations (destination of the message, 801H) to verify that the program moved the correct ASCII bytes. At this time we want to see correct byte value, so "-db" option (display with byte) is used.

```
U>m  -db 801..820
000801..000810  43 6f 6d 6d 61 6e 64 20 41 20 65 6e 74 65 72 65
000811..000820  64 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

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 800..820="Command A entered "
pattern match at address: 000801
```

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 background monitor. While the break will occur as soon as possible, the actual stopping point may be many cycles after the break request (depend on the type of instruction being executed and whether the processor is in a hold state).

```
U>b
M>
```

Using Software Breakpoints

You can stop program execution at specific address by using **bp** (software breakpoint) command. When you define or enable a software breakpoint to a specified address, the emulator will replace the opcode with a BRK 3 instruction. When the emulator detects the breakpoint interrupt instruction (CC hex), user program breaks to the monitor, and the original opcode will be replaced at the software breakpoint address.

Since the system controller knows the locations of the defined software breakpoints, it can determine whether the breakpint interrupt instruction was generated by an enabled software breakpoint or by a single-byte interrupt instruction in your target system.

If the single-byte interrupt was generated by a software brekpoint, execution breaks to the monitor, and the brekpoint interrupt instruction

(BRK 3) is replaced by the original opcode. A subsequent run or step command will execute from this address.

If the single-byte interrupt was geneated by a BRK 3 instruction in the target system, execution still breaks to the monitor, and an "Undefined software breakpoint" message is displayed.

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

Because software brekpoints are implemented by the replacing opcodes with the brekpoint interrupt instruction (CC hex), you can not define the software breakpoints in the target ROM.

However you can copy target ROM into the emulation memory which does allow you to use software brekpoints. Once target ROM is copied into the emulation memory, software breakpoints may be used normally at the addresses in these emulation memory locations. (see the "Target ROM Debug Topics" section of the "In-Circuit Emulation" chapter.)

Note

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

Note

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

Note

Software breakpoint will be ignored, when software breakpoint and other emulation break (for example, break command (**b**), simple trigger command (**tg**), etc.) occur at the same time. Refer to *HP 64700 Emulators Terminal Interface: User's Reference* manual.

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 43aH) label.

```
M>bp 43b
M>bp
```

```
bp 00043b #enabled
```

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

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

```
!ASYNC_STAT 615! Software breakpoint: 00000:00043b
```

```
                                M>st
# s;reg
00000:0043b  -          MOV CW,#0012
PC = 00000:0043e
reg ps=0000 pc=043e psw=0090 aw=0001 bw=0000 cw=0012 dw=ff80 sp=0100 bp=90ff
reg ix=0012 iy=0021 ds0=0060 ds1=0080 ss=0080
```

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

```
                                M>bp
### BREAKPOINT FEATURE IS ENABLED ###
bp 00043b #disabled
                                M>bp -e 00043b
                                M>bp
### BREAKPOINT FEATURE IS ENABLED ###
bp 00043b #enabled
                                M>r
                                U>m 800=43
!ASYNC_STAT  615! Software breakpoint: 00043b
                                M>bp
### BREAKPOINT FEATURE IS ENABLED ###
bp 00043b #disabled
```

Predefined Trace Labels

Three trace labels are predefined in the 70216 emulator. You can view these labels by entering the **tlb** (trace label) command with no options.

```
M>tlb
```

```
#### Emulation trace labels
tlb addr 0..19
tlb data 20..35
tlb stat 36..50
```

Predefined Status Equates

Common values for the 70216 status trace signals have been predefined. You can view these predefined equates by entering the **equ** command with no options.

```
M>equ
```

```
### Equates ###
equ casdma=0xxx1xxxx1010111y
equ cpu=0xxx1xxxx00xxxxy
equ dma=0xxx1xxxx10x01xxy
equ ds0=0xxx1xx11xxxxxxxy
equ ds1=0xxx1xx00xxxxxxxy
equ em80=0xx1xxxxxxxxxxxy
equ exec=0xxx0xxxxxxxxxxy
equ extio=0xxx1xxxx00010xxy
equ fetch=0xxx1xxxx001x100y
equ grd=0xxx10xxxxxxxxxy
equ haltack=0xxx1xxxxxxx1011y
equ holdack=0xxx1xxxx11xxxxxy
equ intack=0xxx1xxxx001x000y
equ intio=0xxx1xxxx00000xxy
equ mem=0xxx1xxxxxx0x1xxy
equ mon=0x0xxxxxxxxxxxxxy
equ native=0xx0xxxxxxxxxxxy
equ ps=0xxx1xx10xxxxxxy
equ read=0xxx1xxxxxx0xx01y
equ refresh=0xxx1xxxx0100101y
equ rom=0xxx1x0xxxxxxxxxy
equ ss=0xxx1xx01xxxxxxy
equ usr=0x1xxxxxxxxxxxxxy
equ write=0xxx1xxxxxx0xx10y
```

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=800 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 using the 70216 emulator.): tg addr=801 and data=42xx

```
M>t
emulation trace started

M>r 0:400
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 800=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	70216 mnemonic,H		xbits,H	count,R	seq
0	00800	ff42	memory read	0000	---	+
1	0041a		exec	0000	0.400 uS	.
2	0041c	f874	fetch	0000	0.480 uS	.
3	0041c	BE/Z 00416		0000	0.520 uS	.
4	0041e	413c	fetch	0000	0.360 uS	.
5	0041e	CMP AL,#41		0000	0.520 uS	.
6	00420	0774	fetch	0000	0.360 uS	.
7	00420	BE/Z 00429		0000	0.520 uS	.
8	00422	423c	fetch	0000	0.360 uS	.
9	00422	CMP AL,#42		0000	0.480 uS	.
10	00424	0c74	fetch	0000	0.400 uS	.
11	00424	BE/Z 00432		0000	0.480 uS	.
12	00426	13eb	fetch	0000	0.400 uS	.
13	0009a	xxxx	refresh	0000	0.880 uS	.
14	00432	12b9	fetch	0000	0.840 uS	.
15	00432	MOV CW,#0012		0000	0.520 uS	.
16	00434	be00	fetch	0000	0.360 uS	.
17	00435	MOV IX,#0012		0000	0.640 uS	.
18	00436	0012	fetch	0000	0.240 uS	.
19	00438	07eb	fetch	0000	0.880 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	70216 mnemonic,H		xbits,H	count,R	seq
20	00438	BR SHORT 00441		0000	0.520 uS	.
21	0043a	b990	fetch	0000	0.360 uS	.
22	0009c	xxxx	refresh	0000	0.880 uS	.
23	00441	bfff	fetch	0000	0.880 uS	.
24	00441	MOV IY,#0001		0000	0.480 uS	.
25	00442	0001	fetch	0000	0.400 uS	.
26	00444	a4f3	fetch	0000	0.840 uS	.
27	00444	REP/E/Z MOVBKB		0000	0.520 uS	.
28	00445	xxxx	exec	0000	0.240 uS	.
29	00446	c032	fetch	0000	0.120 uS	.
30	00448	21b9	fetch	0000	0.880 uS	.
31	00612	ff43	memory read	0000	0.880 uS	.
32	00801	43ff	memory write	0000	0.880 uS	.
33	00613	6fff	memory read	0000	0.880 uS	.
34	00802	ff6f	memory write	0000	0.880 uS	.
35	00614	ff6d	memory read	0000	0.880 uS	.
36	00803	6dff	memory write	0000	0.840 uS	.
37	00615	6dff	memory read	0000	0.880 uS	.
38	00804	ff6d	memory write	0000	0.880 uS	.
39	00616	ff61	memory read	0000	0.880 uS	.

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 500=400..44f
U>r 0:500
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  70216 mnemonic,H                      xbits,H    count,R  seq
-----   ------  ---------------------------------  -------  ---------  ---
    0   00516     a026  fetch                        0000         ---   +
    1   00516   MOV     AL,DS1:0000                  0000      0.520 uS   .
    2   00517     xxxx  exec                         0000      0.240 uS   .
    3   00518     0000  fetch                        0000      0.120 uS   .
    4   0051a     003c  fetch                        0000      0.880 uS   .
    5   00800     ff00  memory read                  0000      0.880 uS   .
    6   0051a   CMP     AL,#00                       0000      0.360 uS   .
    7   0051c     f874  fetch                        0000      0.480 uS   .
    8   0051c   BE/Z    00516                        0000      0.520 uS   .
    9   0051e     413c  fetch                        0000      0.360 uS   .
   10   00114     xxxx  refresh                      0000      0.880 uS   .
   11   00516     a026  fetch                        0000      0.880 uS   .
   12   00516   MOV     AL,DS1:0000                  0000      0.520 uS   .
   13   00517     xxxx  exec                         0000      0.240 uS   .
   14   00518     0000  fetch                        0000      0.120 uS   .
   15   0051a     003c  fetch                        0000      0.880 uS   .
   16   00800     ff00  memory read                  0000      0.880 uS   .
   17   0051a   CMP     AL,#00                       0000      0.360 uS   .
   18   0051c     f874  fetch                        0000      0.520 uS   .
   19   0051c   BE/Z    00516                        0000      0.480 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 "ADDRESS" section in the "70216 Emulator Specific Command Syntax" appendix).

```
U>r 400
R>cov -a 400..450
# coverage list  -  list of address ranges accessed
0000400..000041f

percentage of memory accessed:  % 39.5
```

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 800=41
U>cov -a 400..450
# coverage list  -  list of address ranges accessed
0000400..0000423
0000429..0000433
0000441..0000450

percentage of memory accessed:  % 77.8
U>m 800=42
U>cov -a 400..450
# coverage list  -  list of address ranges accessed
0000400..000043b
0000441..0000450

percentage of memory accessed:  % 92.6
U>m 800=43
U>cov -a 400..450
# coverage list  -  list of address ranges accessed
0000400..0000450

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>
```

3

Emulation Topics

Introduction

Many of the topics described in this chapter involve the commands which are unique to the 70216 emulator such as the **cf** command which allows you to specify emulator configuration.
A reference-type description of the 70216 emulator configuration items can be found in the "70216 Emulator Specific Command Syntax" appendix.

This chapter will:

- Describe how to run in real-time 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 specify a run from reset.

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.

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.

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 4K bytes of memory. Use the **cf** command to select the foreground monitor.

```
R>cf mon=fg..2000
```

2000 defines an hexadecimal address (on a 4K 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 should be 4k boundary and in between 1000H and 0FE000H. 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 4K 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.

Note 

The foreground monitor provided with the 70216 emulator should not be located at a base address 0 or 0ff000 hex; because the 70216 microprocessor's vector table is located.

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.

Selecting Accept Or Ignore Target System Reset

The 70216 emulator can respond or ignore target system reset while running in user program or waiting for target system reset (refer to "**cf rst**" configuration setting in "70216 Emulator specific Command Syntax" appendix).

While running in background monitor, the 70216 emulator ignores target system reset completely independent on this setting.

You can ignore reset from target system completely by specifying "**cf rst=dis**". In this configuration emulator ignore any reset from target system. 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 (0FFFF0 hex) as same manner as actual microprocessor after reset is inactivated

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 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 *Concept of Emulation and Analysis* manual and the "Getting Started" chapter of this manual.

Installing the Emulator Probe into a Target System

The 70216 emulator probe has a 68-pin PLCC connector;
The 70216 emulator is shipped with a pin protector over the target system probe. This guard is designed to prevent impact damage to the pins and should be left in place while you are not using the emulator.

Caution

DAMAGE TO THE EMULATOR CIRCUITRY MAY RESULT IF THESE PRECAUTIONS ARE NOT OBSERVED. The following precautions should be taken while using the 70216 emulator.

Power Down Target System. Turn off power to the user target system and to the 70216 emulator before inserting the user plug to avoid circuit damage resulting from voltage transients or mis-insertion of the user plug.

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

Protect Against Static Discharge. The 70216 emulator contains devices which are susceptible to damage by static discharge. Therefore, operators should take precautionary measures before handling the user plug to avoid emulator damage.

Protect Target System CMOS Components. If your target system includes any CMOS components, turn on the target system first, then turn on the 70216 emulator; when powering down, turn off the emulator first, then turn off power to the target system.

Auxiliary Output Line

One auxiliary output lines, "**TARGET BUFFER DISABLE**" is provided with the 70216 emulator.

Caution

DAMAGE TO THE EMULATOR PROBE WILL RESULT IF THE AUXILIARY OUTPUT LINES ARE INCORRECTLY INSTALLED.

When installing the auxiliary output line into the end of the emulator probe cable, make sure that the ground pin on the auxiliary output line (labeled with white dots) is matched with the ground receptacles in the end of the emulator probe cable.

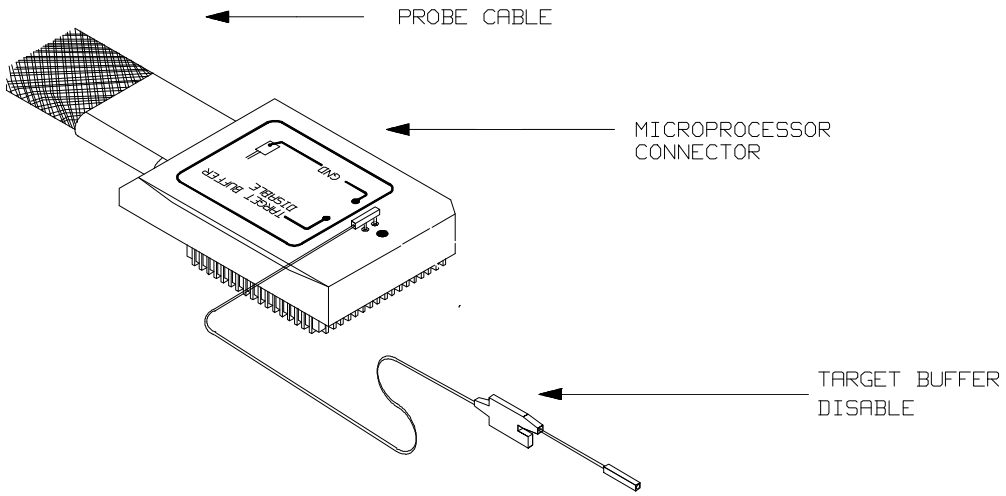
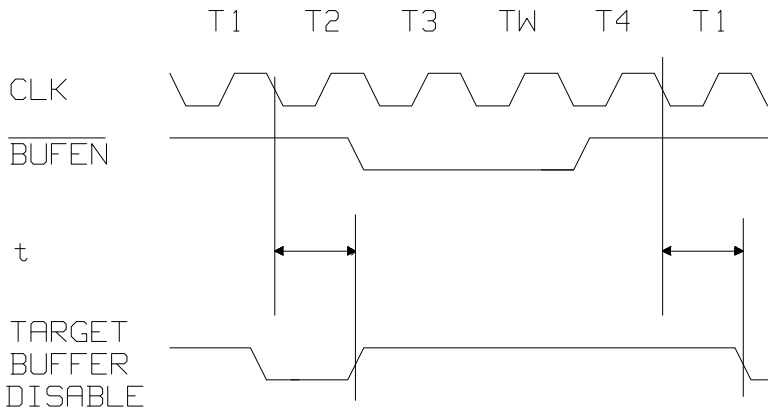


Figure 4-1. Auxiliary Output Lines

TARGET BUFFER DISABLE ---This active-high output is used when the co-processor memory accesses to emulation memory will be operated. This output is used to tristate (in other words, select the high Z output) any target system devices on the 70216 data bus. Target system devices should be tristated because co-processor memory reads from emulation memory will cause data to be output on the user probe.

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



The time ´t´ is

30 nsec MAX. (70208/70216/
70208H/70216H Emulator)

Installing into a PLCC Type Socket

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

- Remove the 70216 microprocessor (PLCC type) from the target system socket. Note the location of pin 1 on the microprocessor and on the target system socket.
- Store the microprocessor in a protected environment (such as antistatic form).
- Install the microprocessor connector into the target system microprocessor socket.

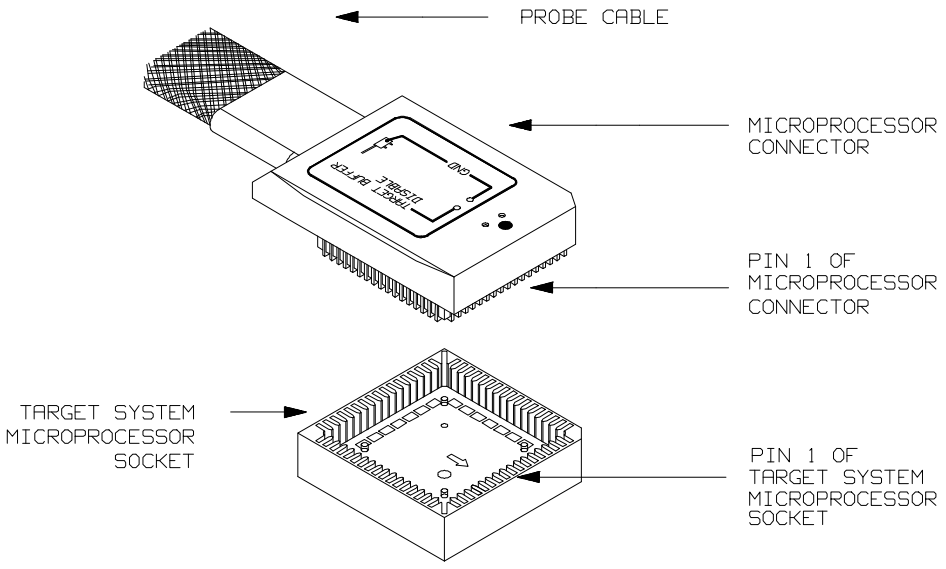


Figure 4-2. Installing into a PLCC type socket

Installing into a PGA Type Socket

You can use an ITT CANNON "LCS-68-12" PLCC connector to plug into the target system socket of an PGA type. You may use this socket with the pin protector to connect the microprocessor connector to the target system. To connect the microprocessor connector to the target system, proceeded with the following instructions.

- Remove the 70216 microprocessor (PGA type) from the target system socket. Note the location of pin A1 on the microprocessor and on the target system socket.
- Store the microprocessor in a protected environment (such as antistatic form).
- Place the microprocessor connector with a PLCC-to-PGA socket and a pin protector (see figure 4-3), attached to the end of the probe cable, into the target system microprocessor socket.

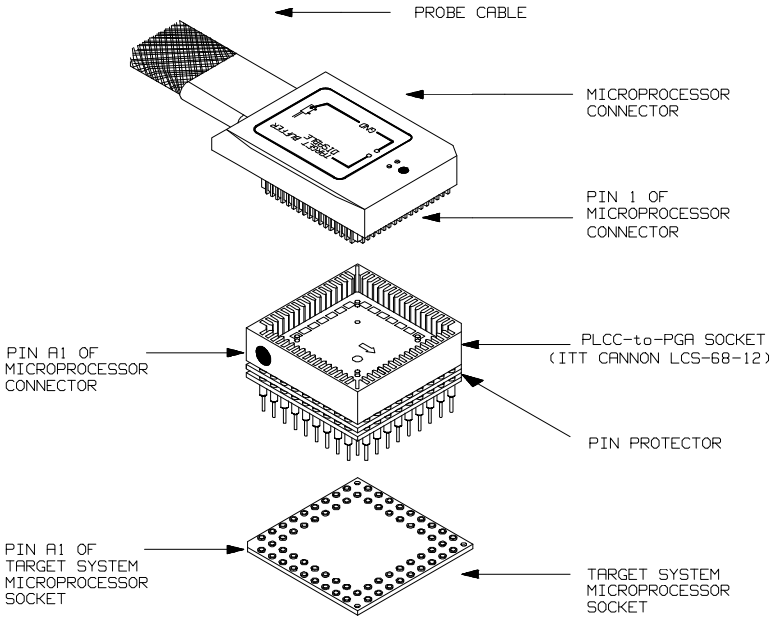


Figure 4-3. Installing into a PGA type socket

Execution Topics

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

Specifying the Emulator Clock Source

The default 70208 and 70216 emulator configuration selects the internal 8 MHz (system clock speed) clock as the emulator clock source. The default 70208H and 70216H emulator configuration selects the internal 16 MHz (system clock speed) clock as the emulator clock source. You should configure the 70208/70208H and 70216/70216H 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
```

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
```

When the ready relationship is locked to the target system, emulation memory accesses honor ready signals from the target system (wait states are inserted if requested).

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

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

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

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
```

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 (refer to the "ADDRESS" section the "70216 Emulator Specific Command Syntax" appendix).

```
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.

Electrical Characteristics (70208/70216)

The AC characteristics of the 70208 and 70216 emulator are listed in the following table

Table 4-2 70208/70216 AC Electrical Specifications

Characteristic	Symbol	uPD70208 uPD70216 10MHz		HP 64791A/2A			Unit
				Worst Case		Typical	
		Min	Max	Min	Max		
External Clock Cycle Times	t_{CYX}	50	250	50	250		ns
External Clock High Level Width	t_{XXH}	19		19			ns
External Clock Low Level Width	t_{XXL}	19		19			ns
External Clock Rise Time	t_{KR}		5		5		ns
External Clock Fall Time	t_{KF}		5		5		ns
CLKOUT Cycle Time	t_{CYK}	100	500	100	500		ns
CLKOUT High Level Width	t_{KKH}	45		45			ns
CLKOUT Low Level Width	t_{KKL}	45		45			ns
CLKOUT Rise Time	t_{KR}		5		5		ns
CLKOUT Fall Time	t_{KF}		5		5		ns
CLKOUT Delay Time (External Clock)	t_{DXK}		40		80		ns
Input Rise Time (Except External Clock)	t_{IR}		15		15		ns
Input Fall Time (Except External Clock)	t_{IF}		10		10		ns

Table 4-2 70208/70216 AC Electrical Specification(Cont'd)

Output Rise Time (Except CLKOUT)	t_{OR}		15		15		ns
Output Fall Time (Except CLKOUT)	t_{OF}		10		10		ns
$\overline{\text{RESET}}$ Setup Time (CLKOUT)	t_{SRESK}	20			61.5		ns
$\overline{\text{RESET}}$ Hold Time (CLKOUT)	t_{HKRES}	25			66.5		ns
RESETOUT Output Delay (CLKOUT)	t_{DKRES}	5	50	0	55.5		ns
READY Inactive Setup Time (CLKOUT)	t_{SRYLK}	15		36.5			ns
READY Inactive Hold Time (CLKOUT)	t_{HKRYL}	20		41.5			ns
READY Active Setup Time (CLKOUT)	t_{SRYHK}	15		36.5			ns
READY Active Hold Time (CLKOUT)	t_{HKRYH}	20		41.5			ns
NMI Setup Time (CLKOUT)	t_{SNMIK}	15		46.5			ns
POLL Setup Time (CLKOUT)	t_{SPOKL}	20		48.5			ns
Data Setup Time (CLKOUT)	t_{SDK}	15		38			ns
Data Hold Time (CLKOUT)	t_{HKD}	10		33			ns
CLKOUT To Address Delay Time	t_{DKA}	10	50	5	55.5		ns
CLKOUT To Address Hold Time	t_{HKA}	10		5			ns
CLKOUT Low To PS Delay Time	t_{DKP}	10	50	5	55.5		ns
CLKOUT High To PS Float Delay Time	t_{FKP}	10	50	5	55.5		ns
Address Setup Time (ASTB)	t_{SAST}	25		23			ns
CLKOUT Low To Address Float Delay Time	t_{FKA}	10	50	10	55.5		ns

Table 4-2 70208/216 AC Electrical Specification(Cont'd)

CLKOUT Low To ASTB High Delay Time	t_{DKSTH}		40		48.5		ns
CLKOUT High To ASTB Low Delay Time	t_{DKSTL}		45		53.5		ns
ASTB High Level Width	t_{STST}	35		35			ns
ASTB Low To Address Hold Time	t_{HSTA}	25		25			ns
CLKOUT To CONTROL1 Delay Time(*1)	t_{DKCT1}	10	60	8.5	68.5		ns
CLKOUT To CONTROL2 Delay Time(*2)	t_{DKCT2}	10	55	8.5	63.5		ns
Address Float To $\overline{RD}$ Low Delay Time	t_{DAFRL}	0		0			ns
CLKOUT Low To $\overline{RD}$ Low Delay Time	t_{DKRL}	10	65	10	73.5		ns
CLKOUT Low To $\overline{RD}$ High Delay Time	t_{DKRH}	10	60	10	68.5		ns
$\overline{RD}$ High To Address Delay Time	t_{DRHA}	60		51.5			ns
$\overline{RD}$ Low Level Width	t_{RR}	160		160			ns
$\overline{BUFEN}$ High To BUF$\overline{R}$/W Delay Time	t_{DBECT}	25		38.5			ns
CLKOUT Low To Data Delay Time	t_{DKD}	10	60	5	65		ns
CLKOUT Low To Data Float Delay Time	t_{FKD}	10	60	5	65		ns
$\overline{WR}$ Low Level Width	t_{WW}	160		160			ns
$\overline{WR}$ High To $\overline{BUFEN}$ High OR BUF$\overline{R}$/W Low	t_{DWCT}	25		25			ns
CLKOUT High To BS Low Delay Time	t_{DKBL}	10	60	10.5	68.5		ns

Table 4-2 70208/216 AC Electrical Specification(Cont'd)

CLKOUT Low To BS High Delay Time	t_{DKBH}	10	60	10.5	68.5		ns
HLDRQ Setup Time (CLKOUT)	t_{SHQK}	20		53			ns
CLKOUT Low TO HLDAK Delay Time	t_{DKHA}	10	70	5	75		ns
CLKOUT High To $\overline{\text{DMAAK}}$ Delay Time	t_{DKHDA}	10	60	5	65		ns
CLKOUT Low To $\overline{\text{DMAAK}}$ Delay Time	t_{DKLDA}	10	90	5	75		ns
$\overline{\text{WR}}$ Low Level Width (DMA Extended Write)	t_{WW1}	160		160			ns
$\overline{\text{WR}}$ Low Level Width (DMA Normal Write)	t_{WW2}	60		60			ns
$\overline{\text{RD}}$ Low, $\overline{\text{WR}}$ Low Delay Time (DMAAK)	t_{DDARW}	15		16.5			ns
$\overline{\text{DMAAK}}$ Hihg Delay Time ($\overline{\text{RD}}$)	t_{DRHDAH}	15		16.5			ns
$\overline{\text{RD}}$ High Delay Time ($\overline{\text{WR}}$)	t_{DWHRH}	5		5			ns
$\overline{\text{TC}}$ Output Delay Time (CLKOUT)	t_{DKTCL}		60		60		ns
$\overline{\text{TC}}$ OFF Delay Time (CLKOUT)	t_{DKTCF}		60		60		ns
$\overline{\text{TC}}$ Low Level Width	t_{TCTCL}	75		75			ns
$\overline{\text{TC}}$ Pull Up Delay Time (CLKOUT)	t_{DKTCH}		135		135		ns
$\overline{\text{END}}$ Setup Time (CLKOUT)	t_{SEDK}	35		51.5			ns
$\overline{\text{END}}$ Low Level Width	t_{EDEDL}	100		100			ns
DMARQ Setup Time (CLKOUT)	t_{SDQF}	35		72.5			ns
INTPn Low Level Width	t_{IPIPL}	100		100			ns

Table 4-2 70208/70216 AC Electrical Specification(Cont'd)

RxD Setup Time (SCU Internal Clock)	t_{SRX}	1000		1000			ns
RxD Hold Time (SCU Internal Clock)	t_{HRX}	1000		1000			ns
CLKOUT Low To $\overline{\text{SRDY}}$ Delay Time	t_{DKSR}		150		155		ns
TOUT1 Low To TxD Delay Time	t_{DTX}		500		505		ns
TCTL2 Setup Time (CLKOUT)	t_{SGK}	50		50			ns
TCTL2 Setup Time (TCLK)	t_{SGTK}	50		50			ns
TCTL2 Hold Time (CLKOUT)	t_{HKG}	100		100			ns
TCTL2 Hold Time (TCLK)	t_{HTKG}	50		50			ns
TCTL2 High Level Width	t_{GGH}	50		50			ns
TCTL2 Low Level Width	t_{GGL}	50		50			ns
TOUT Output Delay Time (CLKOUT)	t_{DKTO}		200		205		ns
TOUT Output Delay Time (TCLK)	t_{DTKTO}		150		155		ns
TOUT Output Delay Time (TCTL2)	t_{DGTO}		120		125		ns
TCLK Rise Time	t_{TKR}		25		25		ns
TCLK Fall Time	t_{TKF}		25		25		ns
TCLK High Level Width	t_{TKTKH}	50		50			ns
TCLK Low Level Width	t_{TKTKL}	50		50			ns
TCLK Cycle Time	t_{CYTK}	124	DC	124	DC		ns
Access Rate	t_{AI}	150		150			ns

Table 4-2 70208/70216 AC Electrical Specification(Cont'd)

$\overline{\text{REFRQ}}$High Delay Time ($\overline{\text{MRD}}$)	t_{DRQHRH}	15		16.5			ns

*1 $\overline{\text{MWR}}$, $\overline{\text{IOWR}}$ during DMA cycle

*2 $\overline{\text{BUFEN}}$, BUF$\overline{\text{R}}$/W, $\overline{\text{INTAK}}$, $\overline{\text{REFRQ}}$, and $\overline{\text{MWR}}$, $\overline{\text{IOWR}}$ during CPU cycle

Electrical Characteristics (70208H/70216H)

The AC characteristics of the 70208H and 70216H emulator are listed in the following table

Table 4-2 70208H/70216H AC Electrical Specifications

Characteristic	Symbol	uPD70208Hu PD70216H 16MHz		HP 64791B/2B			Unit
				Worst Case		Typical	
		Min	Max	Min	Max		
External Clock Cycle Times	t_{CYX}	31.5	DC	31.25	500		ns
External Clock High Level Width	t_{XXH}	12		12			ns
External Clock Low Level Width	t_{XXL}	12		12			ns
External Clock Rise Time	t_{KR}		5		5		ns
External Clock Fall Time	t_{KF}		5		5		ns
CLKOUT Cycle Time	t_{CYK}	62.5	DC	62.5	1000		ns
CLKOUT High Level Width	t_{KKH}	26.25		26.25			ns
CLKOUT Low Level Width	t_{KKL}	26.25		26.25			ns
CLKOUT Rise Time	t_{KR}		5		5		ns
CLKOUT Fall Time	t_{KF}		5		5		ns
CLKOUT Delay Time (External Clock)	t_{DXK}		20		60		ns
Input Rise Time (Except External Clock)	t_{IR}		15		15		ns
Input Fall Time (Except External Clock)	t_{IF}		10		10		ns

Table 4-2 70208H/70216H AC Electrical Specification(Cont'd)

Output Rise Time (Except CLKOUT)	t_{OR}		15		15		ns
Output Fall Time (Except CLKOUT)	t_{OF}		10		10		ns
$\overline{\text{RESET}}$ Setup Time (CLKOUT)	t_{SRESK}	20			61.5		ns
$\overline{\text{RESET}}$ Hold Time (CLKOUT)	t_{HKRES}	15			56.5		ns
RESETOUT Output Delay (CLKOUT)	t_{DKRES}	5	30	0	35		ns
READY Inactive Setup Time (CLKOUT)	t_{SRYLK}	7		28.5			ns
READY Inactive Hold Time (CLKOUT)	t_{HKRYL}	15		36.5			ns
READY Active Setup Time (CLKOUT)	t_{SRYHK}	7		28.5			ns
READY Active Hold Time (CLKOUT)	t_{HKRYH}	15		36.5			ns
NMI Setup Time (CLKOUT)	t_{SNMIK}	15		46.5			ns
POLL Setup Time (CLKOUT)	t_{SPOLK}	20		48.5			ns
Data Setup Time (CLKOUT)	t_{SDK}	7		30			ns
Data Hold Time (CLKOUT)	t_{HKD}	5		28			ns
CLKOUT To Address Delay Time	t_{DKA}	5	25	0	30.5		ns
CLKOUT To Address Hold Time	t_{HKA}	10		5			ns
CLKOUT Low To PS Delay Time	t_{DKP}	5	30	0	35.5		ns
CLKOUT High To PS Float Delay Time	t_{FKP}	5	30	0	35.5		ns
Address Setup Time (ASTB)	t_{SAST}	16.25		14.25			ns
CLKOUT Low To Address Float Delay Time	t_{FKA}	10	30	5	35.5		ns

Table 4-2 70208H/216H AC Electrical Specification(Cont'd)

CLKOUT Low To ASTB High Delay Time	t_{DKSTH}		25		33.5		ns
CLKOUT High To ASTB Low Delay Time	t_{DKSTL}		30		38.5		ns
ASTB High Level Width	t_{STST}	16.25		16.25			ns
ASTB Low To Address Hold Time	t_{HSTA}	16.25		16.25			ns
CLKOUT To CONTROL1 Delay Time(*1)	t_{DKCT1}	5	40	3.5	48.5		ns
CLKOUT To CONTROL2 Delay Time(*2)	t_{DKCT2}	5	35	3.5	43.5		ns
Address Float To $\overline{RD}$ Low Delay Time	t_{DAFRL}	0		0			ns
CLKOUT Low To $\overline{RD}$ Low Delay Time	t_{DKRL}	5	40	5	48.5		ns
CLKOUT Low To $\overline{RD}$ High Delay Time	t_{DKRH}	5	35	5	43.5		ns
$\overline{RD}$ High To Address Delay Time	t_{DRHA}	52.5		44			ns
$\overline{RD}$ Low Level Width	t_{RR}	105		105			ns
$\overline{BUFEN}$ High To BUF$\overline{R}$/W Delay Time	t_{DBECT}	16.25		29.75			ns
CLKOUT Low To Data Delay Time	t_{DKD}	5	30	0	35		ns
CLKOUT Low To Data Float Delay Time	t_{FKD}	5	30	0	35		ns
$\overline{WR}$ Low Level Width	t_{WW}	105		105			ns
$\overline{WR}$ High To $\overline{BUFEN}$ High OR BUF$\overline{R}$/W Low	t_{DWCT}	16.25		16.25			ns
CLKOUT High To BS Low Delay Time	t_{DKBL}	5	30	5.5	38.5		ns

Table 4-2 70208H/216H AC Electrical Specification(Cont'd)

CLKOUT Low To BS High Delay Time	t_{DKBH}	5	30	5.5	38.5		ns
HLDRQ Setup Time (CLKOUT)	t_{SHQK}	7		40			ns
CLKOUT Low TO HLDAK Delay Time	t_{DKHA}	5	40	0	45		ns
CLKOUT High To $\overline{\text{DMAAK}}$ Delay Time	t_{DKHDA}	5	35	0	40		ns
CLKOUT Low To $\overline{\text{DMAAK}}$ Delay Time	t_{DKLDA}	5	55	0	40		ns
$\overline{\text{WR}}$ Low Level Width (DMA Extended Write)	t_{WW1}	105		105			ns
$\overline{\text{WR}}$ Low Level Width (DMA Normal Write)	t_{WW2}	47.5		47.5			ns
$\overline{\text{RD}}$ Low, $\overline{\text{WR}}$ Low Delay Time (DMAAK)	t_{DDARW}	11.25		12.75			ns
$\overline{\text{DMAAK}}$ Hihg Delay Time ($\overline{\text{RD}}$)	t_{DRHDAH}	11.25		12.75			ns
$\overline{\text{RD}}$ High Delay Time ($\overline{\text{WR}}$)	t_{DWHRH}	5		5			ns
$\overline{\text{TC}}$ Output Delay Time (CLKOUT)	t_{DKTCL}		35		35		ns
$\overline{\text{TC}}$ OFF Delay Time (CLKOUT)	t_{DKTCF}		35		35		ns
$\overline{\text{TC}}$ Low Level Width	t_{TCTCL}	52.5		52.5			ns
$\overline{\text{TC}}$ Pull Up Delay Time (CLKOUT)	t_{DKTCH}		83.75		83.75		ns
$\overline{\text{END}}$ Setup Time (CLKOUT)	t_{SEDK}	20		36.5			ns
$\overline{\text{END}}$ Low Level Width	t_{EDEDL}	50		50			ns
DMARQ Setup Time (CLKOUT)	t_{SDQF}	15		52.5			ns
INTPn Low Level Width	t_{IPIPL}	80		80			ns

Table 4-2 70208H/70216H AC Electrical Specification(Cont'd)

RxD Setup Time (SCU Internal Clock)	t_{SRX}	500		500			ns
RxD Hold Time (SCU Internal Clock)	t_{HRX}	500		500			ns
CLKOUT Low To $\overline{\text{SRDY}}$ Delay Time	t_{DKSR}		100		105		ns
TOUT1 Low To TxD Delay Time	t_{DTX}		200		205		ns
TCTL2 Setup Time (CLKOUT)	t_{SGK}	40		40			ns
TCTL2 Setup Time (TCLK)	t_{SGTK}	40		40			ns
TCTL2 Hold Time (CLKOUT)	t_{HKG}	80		80			ns
TCTL2 Hold Time (TCLK)	t_{HTKG}	40		40			ns
TCTL2 High Level Width	t_{GGH}	40		40			ns
TCTL2 Low Level Width	t_{GGL}	40		40			ns
TOUT Output Delay Time (CLKOUT)	t_{DKTO}		150		155		ns
TOUT Output Delay Time (TCLK)	t_{DTKTO}		100		105		ns
TOUT Output Delay Time (TCTL2)	t_{DGTO}		90		95		ns
TCLK Rise Time	t_{TKR}		25		25		ns
TCLK Fall Time	t_{TKF}		25		25		ns
TCLK High Level Width	t_{TKTKH}	30		30			ns
TCLK Low Level Width	t_{TKTKL}	30		30			ns
TCLK Cycle Time	t_{CYTK}	62.5	DC	62.5	DC		ns
Access Rate	t_{AI}	105		105			ns

Table 4-2 70208H/70216H AC Electrical Specification(Cont'd)

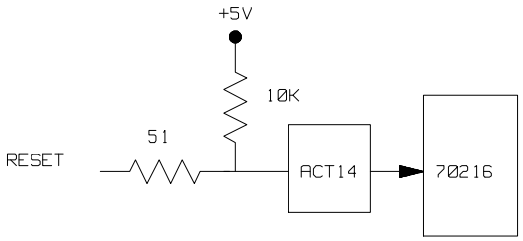
$\overline{\text{REFRQ}}$High Delay Time ($\overline{\text{MRD}}$)	t_{DRQHRH}	16.25		17.75			ns

*1 $\overline{\text{MWR}}$, $\overline{\text{IOWR}}$ during DMA cycle

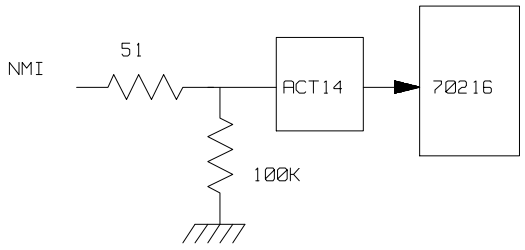
*2 $\overline{\text{BUFEN}}$, BUF$\overline{\text{R}}$/W, $\overline{\text{INTAK}}$, $\overline{\text{REFRQ}}$, and $\overline{\text{MWR}}$, $\overline{\text{IOWR}}$ during CPU cycle

Target System Interface

$\overline{\text{RESET}}$ This singal is connected to 70216 through ACT14, 51ohm and 10K ohm pull-up register.

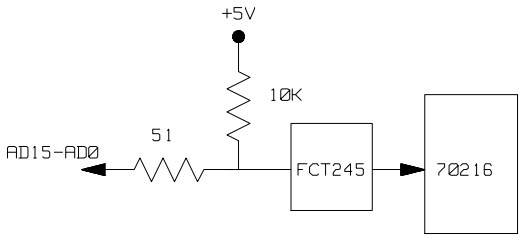


NMI This singal is connected to 70216 through ACT14, 51 ohm and 100K ohm pull-down register.



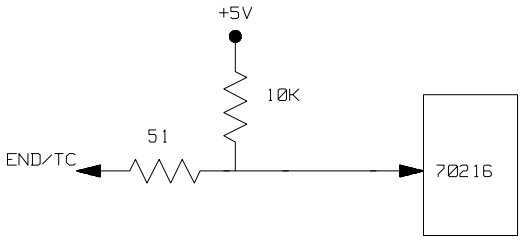
AD15-AD0

These singals are connected to 70216 through FCT245, 51 ohm and 10K ohm pull-up register.



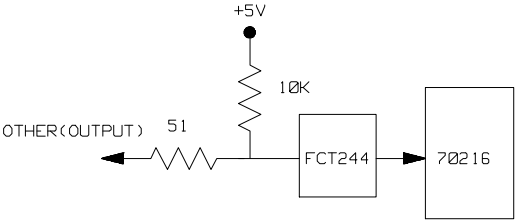
$\overline{\text{END}}/\overline{\text{TC}}$

This singal is connected to 70216 through 51 ohm and 10K ohm pull-up register.



OTHER(OUTPUT)

These singals are connected to 70216 through FCT244, 51 ohm and 10K ohm pull-up registers.



Notes

A

70216 Emulator Specific Command Syntax

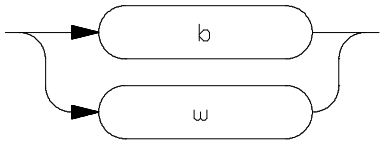
The following pages contain descriptions of command syntax specific to the 70216 emulator. The following syntax items are included (several items are part of other command syntax):

- <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. When the emulator read or write odd number of byte data, the emulator will read or write the last byte data using byte cycle. At an odd address, the emulator will access target memory using byte cycles.

Defaults In the 70208 and 70208H,the **<ACCESS_MODE>** is **b** at power up initialization.In the 70216 and 70216H,the **<ACCESS_MODE>** is **w**

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.

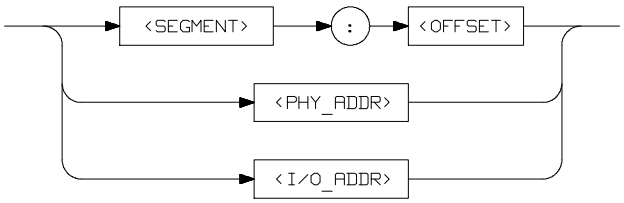
Related Commands **mo** (specify display and access modes)

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 or as physical address (though a physical address in run or step command is converted to logical address by the emulation system).

Parameters

**** This expression (0-0FFFF hex) is the segment portion of the logical address. The value specified is placed in the 70216 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 70216 PC register before running or stepping.

<PHY_ADDR> This expression (0-0FFFFF hex) is a physical address in the 70216 address range. In run and step commands, 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).

<I/O_ADDR> This expression (0-0FFFF hex) with no function code is a 70216 I/O address. This expression should be used in I/O command.

Defaults

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

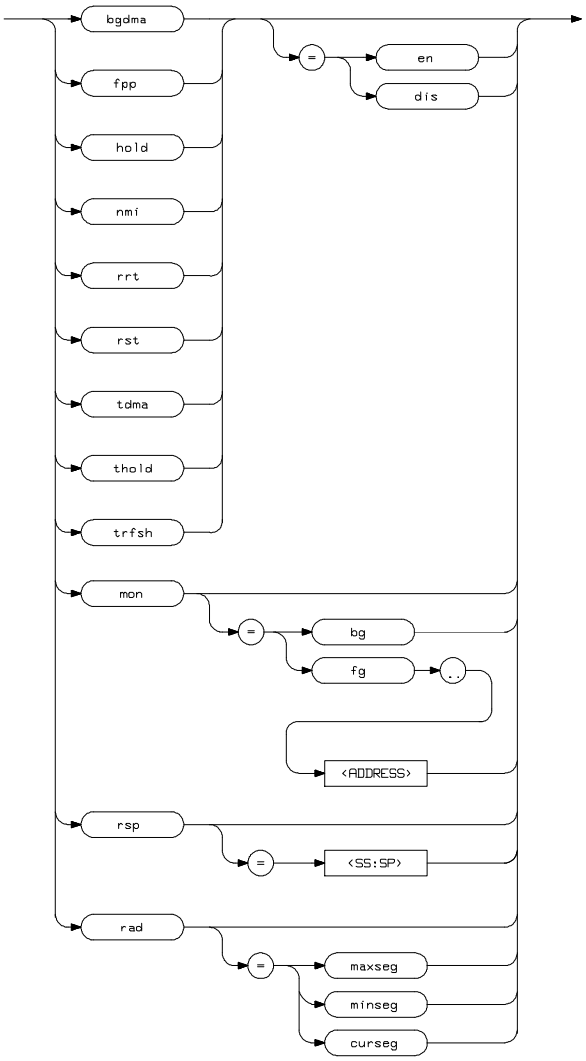
Related Commands

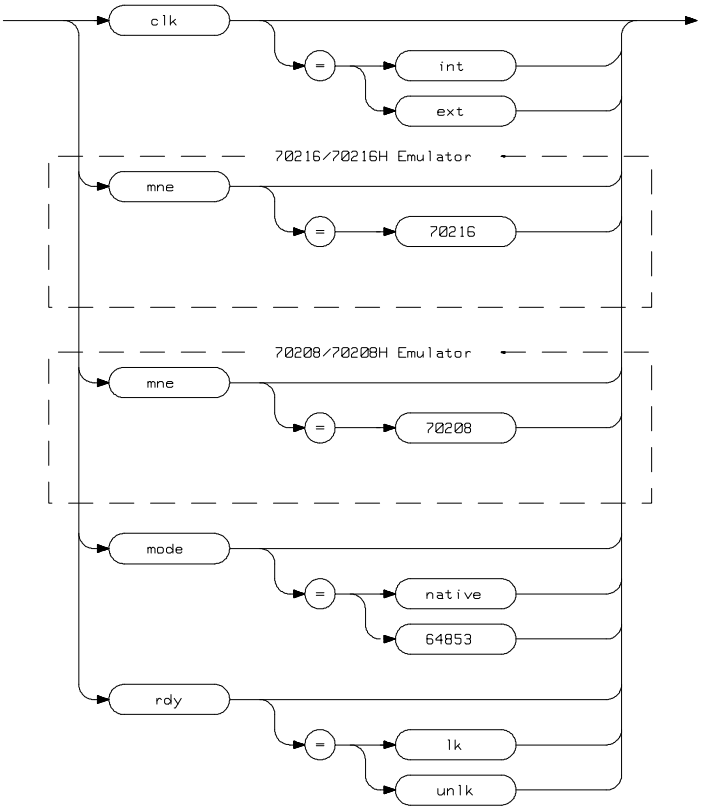
<CONFIG_ITEMS> (70216 specific items specified with the **cf** command)

CONFIG_ITEMS

Summary 70208/70216 emulator configuration items.

Syntax





Function The **<CONFIG_ITEMS>** are the 70216 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

bgdma Enabling internal DMA during background operation. This configuration allows you to specify whether or not the 70216 emulation processor's internal DMA is allowed while in background.

Setting **bgdma** equal to **en** specifies that the internal DMA is allowed while in background.

Setting **bgdma** equal to **dis** specifies that the internal DMA is not allowed while in background.

Note

If you use the background monitor and set bgdma equal to dis, DMA request from internal DMA controller is ignored during background operation. When the emulator goes into foreground operation, the emulator accepts DMA request.

---

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 70208 and 70216 Emulator, the internal clock speed is 8 MHz (system clock).
In the 70208H and 70216H Emulator, the internal clock speed is 16 MHz (system clock).

The 70208 and 70216 emulator will operate at external clock speed from 4 to 20 MHz (entered clock).
The 70208H and 70216H emulator will operate at external clock speed from 2 to 32 MHz (entered clock).

fpp

This configuration allows you to use to FPP(Floating Point co-Processor) and to specify whether the emulator will drive the target system bus during ANY memory cycle.

Setting **fpp** equal to **en** specifies that the emulator will drive target system,during ANY emulation memory cycle.

If the target system does not contain a floating point co-processor, you should answer "**dis**".

When "en" is selected, a special hardware mode which allows the emulator to support a floating point co-processor is enabled. When a floating point co-processor is present, it must monitor all address and data that the emulation processor inputs and outputs.
Because of this, it is necessary to enable data bus drivers to the target system for all emulation memory read cycles.

This is normally done only on write cycles, and is not done on read cycles to avoid bus contention problems between the emulator and the target system. When this mode is enabled, the USER output from the pod should be used to disable user buffers that would normally to turned on when the emulator is reading from emulation memory. Also you should select "cf hold=en" for target hold signal input.

hold

Respond to target hold . This configuration 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.

mne This configuration item is reserved. Change of this item has no effect.

mode Select the mode of dis-assembler that are used by the emulator to display assembler syntax.

Setting **mode** equal to **native** specifies that the emulator will display dis-assembler with AxLS(HP64873) assembler syntax.

Setting **mode** equal to **64853** specifies that the emulator will display dis-assembler with OLS(HP64853) assembler syntax.

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. (Breaks to the monitor still put the emulator into the background mode, but the monitor program returns to foreground before performing any functions.)

Note

You should use the 20 bits physical address or segment:offset address expression to locate the foreground monitor on a 4K byte boundary. The start address of the foreground monitor should not be located at a base address 0 or 0ff000 hex;because the 70216 microprocessor's vector table is located. Refer to the "Using the Optional Foreground Monitor" appendix in this manual.

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**. When **nmi** is set to **dis**, the emulator ignores user NMI input.

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 ignored until the monitor is finished.

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.

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, or display/modify target system memory or I/O) are not allowed.

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.

Note 

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

---

rst

The 70216 emulator can respond or ignore target system reset while running in user program or waiting for target system reset.

While running in background monitor, the 70216 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 as 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 any 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 should use **"cf rst=en"** configuration setting.

---

tdma

Trace Internal DMA cycles. This question allows you to specify whether or not the analyzer trace the 70216 emulation processor's internal DMA cycles.

Setting **tdma** equal to **en** specifies that the analyzer will trace the 70216 internal DMA cycles.

Setting **tdma** equal to **dis** specifies that the analyzer will not trace the 70216 internal DMA cycles.

thold Trace hold acknowledge cycles. This question allows you to specify whether or not the analyzer trace the 70216 emulation processor's hold acknowledge cycles.

Setting **thold** equal to **en** specifies that the analyzer will trace the 70216 hold acknowledge cycles.

Setting **thold** equal to **dis** specifies that the analyzer will not trace the 70216 hold acknowledge cycles.

trfsh Trace refresh cycles. This question allows you to specify whether or not the analyzer trace the 70216 emulation processor's refresh cycles.

Setting **trfsh** equal to **en** specifies that the analyzer will trace the 70216 refresh cycles.

Setting **trfsh** equal to **dis** specifies that the analyzer will not trace the 70216 refresh cycles.

Defaults

The default values of the 70216 and 70216H emulator configuration items are listed below.

```
cf bgdma=dis
cf clk=int
cf fpp=dis
cf hold=en
cf mne=70216
cf mode=native
cf mon=bg
cf nmi=en
cf rad=minseg
cf rdy=lk
cf rrt=dis
cf rsp=0000:0009
cf rst=en
cf tdma=en
cf thold=en
cf trfsh=en
mo -aw -dw
```

The default values of the 70208 and 70208H emulator configuration items are listed below.

```
cf bgdma=dis
cf clk=int
cf fpp=dis
cf hold=en
cf mne=70208
cf mode=native
cf mon=bg
cf nmi=en
cf rad=minseg
cf rdy=lk
cf rrt=dis
cf rsp=0000:0009
cf rst=en
cf tdma=en
cf thold=en
cf trfsh=en
mo -ab -db
```

Related Commands

help

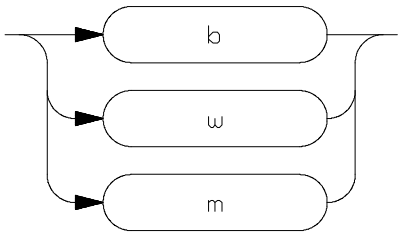
You can get an on line help information for particular configuration items by typing:

```
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.
m	**Mnemonic**. Memory is displayed in mnemonic format; that is, the contents of memory locations are inverse-assembled into mnemonics and operand. 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 70208 and 70208H Emulator, the **<ACCESS_MODE>** and **<DISPLAY_MODE>** are **b**. In the 70216 and 70216H Emulator, the **<ACCESS_MODE>** and **<DISPLAY_MODE>** is **w** at power up initialization.

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 NAMES and CLASSES

The following register names and classes are used with the display/modify registers commands in 70216 emulator.

BASIC(*) class

Register name	Description
aw, bw	BASIC registers.
cw, dw	
bp, ix, iy	
ds0, ds1, ss	
sp, pc, ps, psw	

SIO class (70208/70216 Emulator)

(System I/O registers)

Register name	Description
opcn	On-chip peripheral connection register
opsel	On-chip peripheral selection register
opha	On-chip peripheral high address register
dula	DMAU low address register
iula	ICU low address register
tula	TCU low address register
sula	SCU low address register
wcy1	Programmable wait, cycle 1 register
wcy2	Programmable wait, cycle 2 register
wmb	Programmable wait, memory boundary register
rfc	Refresh control register
tcks	Timer clock selection register

SIO class (70208H/70216H Emulator)

(System I/O registers)

Register name	Description
opcn	On-chip peripheral connection register
opsel	On-chip peripheral selection register
opha	On-chip peripheral high address register
dula	DMAU low address register
iula	ICU low address register
tula	TCU low address register
sula	SCU low address register
sctl	System control register
wcy1	Programmable wait, cycle 1 register
wcy2	Programmable wait, cycle 2 register
wmb	Programmable wait, memory boundary register
rfc	Refresh control register
sbcr	Stand-by control register
tcks	Timer clock selection register
exwb	Extended wait block selection register
wsmb	Wait submemory block selection register
wiob	Wait I/O block selection register
wcy3	Programmable wait, cycle 3 register
brc	Boud rate counter
badr	Bank address register
bsel	Bank select register

ICU class (Interrupt Control Unit registers)

Register name	Description	
imkw	Interrupt mask word register	
irq	Interrupt request register	(Read only)
iis	Interrupt in-service register	(Read only)
ipol	Interrupt polling register	(Read only)
ipfw	Interrupt priority and finish word register (Write only)	
imdw	Interrupt mode word register	(Write only)
iiw1	Interrupt initialize word 1 register	(Write only)
iiw2	Interrupt initialize word 2 register	(Write only)
iiw3	Interrupt initialize word 3 register	(Write only)
iiw4	Interrupt initialize word 4 register	(Write only)

Caution

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

TCU class (Timer Control Unit registers)

Register name	Description	
tct0	Timer/counter 0 register	
tst0	Timer status 0 register	(Read only)
tct1	Timer/counter 1 register	
tst1	Timer status 1 register	(Read only)
tct2	Timer/counter 2 register	
tst2	Timer status 2 register	(Read only)
tmd	Timer/counter mode register	(Write only)

SCU class

(Serial Control Unit registers)

Register name	Description	
srb	Serial receive data buffer	(Read only)
sst	Serial status register	(Read only)
stb	Serial transmit data buffer	(Write only)
scm	Serial command register	(Write only)
smd	Serial mode register	(Write only)
simk	Serial interrupt mask register	(Write only)

DMA71 class

(DMA Control Unit registers (for uPD71071 mode)

Register name	Description	
dicm	DMA initialize register	(Write only)
dch	DMA channel register	
dbc/dcc0	DMA base/current count register channel 0	
dbc/dcc1	DMA base/current count register channel 1	
dbc/dcc2	DMA base/current count register channel 2	
dbc/dcc3	DMA base/current count register channel 3	
dba/dca0	DMA base/current address register channel 0	
dba/dca1	DMA base/current address register channel 1	
dba/dca2	DMA base/current address register channel 2	
dba/dca3	DMA base/current address register channel 3	
dmd0	DMA mode control register channel 0	
dmd1	DMA mode control register channel 1	
dmd2	DMA mode control register channel 2	
dmd3	DMA mode control register channel 3	
ddc	DMA device control register	
dst	DMA status register	(Read only)
dmk	DMA mask register	

DMA37 class (70208H/70216H Emulator only)

(DMA Control Unit register (for uPD71037 mode)

Register name	Description
cmd	DMA read status/write command register
bank0	DMA bank register channel 0
bank1	DMA bank register channel 1
bank2	DMA bank register channel 2
bank3	DMA bank register channel 3
adr0	DMA current address register channel 0
adr1	DMA current address register channel 1
adr2	DMA current address register channel 2
adr3	DMA current address register channel 3
cnt0	DMA current count register channel 0
cnt1	DMA current count register channel 1
cnt2	DMA current count register channel 2
cnt3	DMA current count register channel 3
sfrq	Software DMA write request register channel (Write only)
smsk	DMA write single mask register channel (Write only)
mode	DMA write mode register channel (Write only)
clbp	DMA clear byte pointer F/F (Write only)
init	DMA initialize register (Write only)
cmsk	DMA clear mask register (Write only)
amsk	DMA write all mask register bit (Write only)

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 **FMV50.S**, **FMV50HL.S**, **FMV40.S** and **FMV40HL.S** are for the HP 64853 Cross Assembler/Linker.

Note

Use the appropriate monitor; "FMV50.S" for the 70216, "FMV50HL.S" for the 70216H, "FMV40.S" for the 70208 and "FMV40HL.S" for the 70208H emulator. "FMV50.S" foreground monitor program is used in this example. If your emulator is for the other emulator, read this appendix by replacing "FMV50.S" with the appropriate monitor.

Comparison of Foreground and Background Monitors

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

Background Monitors

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

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

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

Foreground Monitors

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

You can tailor the foreground monitor to meet your needs, such as servicing target system interrupts. However, the foreground monitor does use part of the processor's address space, which may cause problems in some target systems. You must also properly configure the emulator to use a foreground monitor (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 400 hex with its data at 600 hex and its common at 800 hex.

Modify EQU Statement

To use the monitor, you must modify the EQU statement near the top of the monitor listing 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 4k byte boundary.

Note

You should not load the foreground monitor at the base address 0 or 0ff000 hex; because the 70216 microprocessor's vector table is located.

Assemble and Link the Monitor

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

```
$ asm -o FMV50.S > FMV50.LIS <RETURN>
$ lnk <RETURN>
object files  FMV50.R  <RETURN>
library files  <RETURN>
Load addresses: PROG,DATA,COMN  0,0,0
<RETURN>
more files (y or n)  n  <RETURN>
absolute file name  FMV50.X  <RETURN>
```

If you haven't already assembled and linked the sample program, do that now. Refer to the "Getting Started" chapter for instructions on assembling and linking the sample program.

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 4k 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  : 1f000h bytes
map  0001000..0001fff  eram 16       # term 1
map  other tram
```

Notice that a 4k byte block from 1000 through 1fff hex was mapped.

Now, you need to map memory space for the sample program. Let's map the memory from 0 through 5ff hex to emulation ROM and map the memory from 600 through 9ff hex to emulation RAM in this example.

```
R> map 0..7ff erom
R> map 800..9ff eram
```

Load the Program Code

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 FMV50.X"
```

##############

Load the Sample Program

Assuming the sample program has been assembled and linked as shown in Chapter 2, 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
```

Before we forget, let's initialize the stack pointer by breaking the emulator out of reset:

```
R> cf rsp=0:0a00
R> b
```

Now you can run the sample program with the following command:

```
M> r 400
```

Single Step and Foreground Monitors

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

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, set the foreground/background configuration option to **cf mon=bg**.

Specific Error Messages

The following pages document the error messages which are specific to the 70208/70208H and 70216/70216H emulator. The cause of the error is described, as well as the action you must take to remedy the situation.

Message 140 : Second term is smaller than first term

Cause

This error occurs when you attempt to enter the address range with the first term is smaller than the second term.

Action

Enter the address range with the first term is not smaller than the second term.

Message 141 : Range terms must be the same type

Cause

This error occurs when you attempt to enter the address range with the address expression of the first term is different from the second term.

Action

Use the same address expression to enter the address range. Refer to "ADDRESS" section of the appendix A of this manual for more information.

Message 142 : Range terms must be in the same segment

Cause

This error occurs when you attempt to enter the address range with :<OFFSET> address expression and the value of first term is different from the value of second term.

Action

Use the same value to enter the address range with :<OFFSET> address expression.

Message 143 : Physical address can not be used

Cause

This error occurs when you attempt to enter the address with physical address expression in the emulation commands which physical address expression cannot be used in.

Action

Use the other address expression which can be used in the emulation commands. Refer the "ADDRESS" section of the Appendix A in this manual.

Message 144 : I/O address range overflow

Cause

This error occurs when you attempt to enter the I/O address which is over I/O address range (0-0FFFF hex) after executing the I/O command.

Action

Use the I/O address which is not over the I/O address range after executing the I/O command.

Message 145 : Stack pointer not initialized

Cause

This error occurs when you attempt to execute program without setting stack pointer in foreground moniter.

Action

Enter the stack pointer address. Refer the "CONFIG_ITEMS" section of the Appendix A in this manual.

Message 146 : Reset by force of internal 32MHz clock

Cause

This error occurs when you attempt to reset 70208H/70216H Emulator with very slow external clock or with external DC clock.

Action

In this case,the 70208H/70216H emulator was reset by force of internal 32MHz clock. Use the internal clock, this error does not occurs.

Message 147 : Unable to modify breakpoint while running

Cause

This error occurs when you attempt to modify the software breakpoint while the emulator is running user code.

Action

Enter the command **b** to break into the monitor and modify the breakpoint.

Message 148: Access size greater than display size, access=%c, display=%c

Cause

This error occurs when you attempt to display or modify I/O contents by the byte in the word access mode.

Action

Change the access mode to byte.

Message 150 : DMA controller is 71071 mode

Cause

This error occurs when you attempt to access the DMA37 class registers (refer to "REGISTER NAMES and CLASSES" section in Appendix A) and the 70216H internal DMA Control Unit is uPD71071 mode.

Action

Change the mode of 70216H internal DMA Control Unit to uPD71037 mode.(70208H/70216H Emulator only)

Message 151 : DMA controller is 71037 mode (70208H/70216H Emulator only)

Cause

This error occurs when you attempt to access the DMA71 class registers (refer to "REGISTER NAMES and CLASSES" section in Appendix A) and the 70208H or 70216H internal DMA Control Unit is uPD71037 mode.

Action

Change the mode of 70208H or 70216H internal DMA Control Unit to uPD71071 mode.

Message 152 : Device not enable

Cause

This error occurs when you attempt to access registers in the 70216 internal peripheral (ICU, TCU, SCU, and DMAU) and the internal peripheral is disabled.

Action

Enable the internal peripheral to modify the OPSEL register (on-chip peripheral selection register) with using "reg" command (refer to "REGISTER NAMES and CLASSES" section in Appendix A).

Notes

Index