HP 64146

7700 Series Emulator Terminal Interface

User's Guide

HP Part No. 64146-97003
Printed in U.S.A.
February 1994

Edition 2

Notice

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

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

UNIX is a registered trademark of AT&T.

MELPS is a registered trademark of Mitsubishi Electric Corporation.

Torx is a registered trademark of Camcar Division of Textron, Inc.

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 64146-97000, July 1992

Edition 2 64146-97003, February 1994

Using This Manual

This manual is designed to give you an introduction to the HP 64146A/B 7700 Series Emulator. This manual will also help define how these emulators differ from other HP 64700 Emulators.

This manual will:

- give you an introduction to using the emulator
- explore various ways of applying the emulator to accomplish your tasks
- show you emulator commands which are specific to the 7700 Series Emulator

This manual will not:

- tell you how to use each and every emulator/analyzer command (refer to the *User's Reference* manual)

For the most part, the HP 64146A and HP 64146B emulators all operate the same way. Differences of between the emulators are described where they exist. Both the HP 64146A and HP 64146B emulators will be referred to as the "HP 64146A/B 7700 Series emulator" or "7700 Series emulator". In the specific instances where HP 64146B emulator differs from HP 64146A emulator, it will be described as "HP 64146B emulator".

Organization

Chapter 1 An introduction to the HP 64146A/B 7700 Series emulator features and how they can help you in developing new hardware and software.

Chapter 2 A brief introduction to using the HP 64146A/B 7700 Series Emulator. You will load and execute a short program, and make some measurements using the emulation analyzer.

Chapter 3 Configuring the emulator to adapt it to your specific measurement needs.

Chapter 4 How to plug the emulator probe into a target system.

Appendix A Using a foreground monitor program; advantages and disadvantages.

Appendix B 7700 Series Emulator Specific Command Syntax

Contents

Illustrations

Tables

1

Introduction to the 7700 Series Emulator

Introduction

The topics in this chapter include:

- Purpose of the 7700 Series Emulator
- Features of the 7700 Series Emulator

Purpose of the 7700 Series Emulator

The HP 64146A/B 7700 Series Emulator is designed to replace the MELPS 7700 Series microprocessor in your target system so you can control operation of the processor in your application hardware (usually referred to as the *target system*). The emulator performs just like the MELPS 7700 Series microprocessor, but is a device that allows you to control the MELPS 7700 Series directly. These features allow you to easily debug software before any hardware is available, and ease the task of integrating hardware and software.

Note

In this manual, MELPS 7700 Series is referred to as 7700 Series.

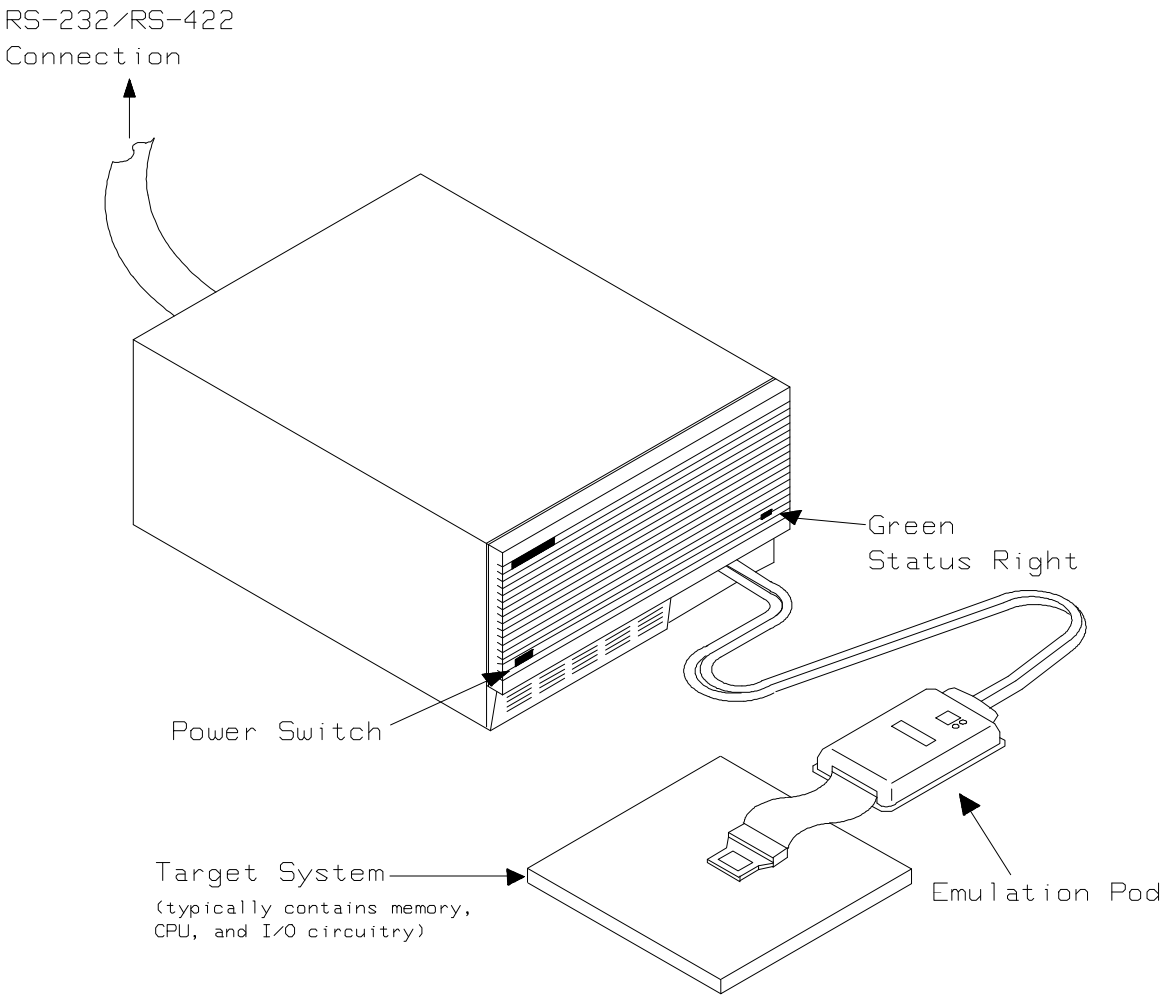


Figure 1-1. HP 64146 Emulator for MELPS 7700 Series

1-2 Introduction

Supported Microprocessors

A list of the supported 7700 Series microprocessors is shown in Table 1-1. You need to purchase appropriate emulation pod and emulation processor.

Processor		Clock	Emulation Processor	Emulation Pod
M37700/1	M2-xxxFP/SP	8	M37700SAFP	M37700T-HPD
	M2AxxxFP/SP	16		
	SFP/SP	8		
	SAFP/SP	16		
M37700/1	M4-xxxFP/SP	8	M37700S4AFP	
	M4AxxxFP/SP	16		
	S4FP/SP	8		
	S4AFP/SP	16		
M37702/3	M2-xxxFP/SP	8	M37702S1AFP	M37702T-HPD
	M2AxxxFP/SP	16		
	S1FP/SP	8		
	S1AFP/SP	16		
M37702/3	M4-xxxFP/SP	8	M37702S4AFP	
	M4AxxxFP/SP	16		
	S4FP/SP	8		
	S4AFP/SP	16		
M37702	M6LxxxFP	8	M37702S1BFP	M37702TL-HPD HP 641466-61002 (64146B)
M37702/3	M2BxxxFP/SP	25	M37702S1BFP	M37702TB-HPD
	S1BFP/SP	25		HP 64146-61001 (64146A)
M37702/3	M4BxxxFP/SP	25	M37702S4BFP	HP 64146-61002
	S4BFP/SP	25		(64146B)
	M6BxxxFP	25		
M37704/5	M2-xxxFP/SP	8	M37704S1AFP	M37704T-HPD
	M2AxxxFP/SP	16		
	S1FP/SP	8		
	S1AFP/SP	16		
M37704	M3BxxxFP	25	M37704M4BFP	M37704TB-HPD
	M3BxxxFP	25		
M37710	M4BxxxFP	25	M37710M4BFP	M37710TL-HPD
	S4BFP	25		
M37720	S1FP	8	M37720S1AFP	M37720T-HPD
	S1AFP	16		
M37730	S2FP/SP	8	M37730S2AFP	M37730T-HPD
	S2AFP/SP	16		

Table 1-1. Supported Microprocessors

M37732	S4FP/SP	8	M37732S4AFP	M37732T-HPD
	S4AFP/SP	16		
M37780	STJ/FP	16	M37780STJ	M37780T-HPD
M37781	M4TxxxJ/FP	16	M37781M4TJ	M37781T-HPD
	E4TxxxJ/FP	16		
M37795	SJ	8	M37795SJ	M37795T-HPD
	STJ	8		
M37796	E4-xxxJ	8	M37796E4J	
	E4TxxxJ	8		

Table 1-1. Supported Microprocessors (Cont'd)

The HP 64146A emulator is provided with the following items.

- HP 64146-61001 emulation pod with M37702S1BFP emulation processor
- Adaptor for M37703 processor

The HP 64146B emulator is provided with the following items.

- HP 64146-61002 emulation pod with M37702S1BFP emulation processor
- Adaptor for M37703 processor

As you can see from Table 1-1, the HP 64146A/B emulator can emulate M37702/3M2 and M37702/3S1 processor by default. These emulation pods can be used with clock up to 25 MHz. Also, HP 64146B emulator can emulate M37702 M6L processor using default emulation pod, HP 64146-61002.

To emulate other processors of 7700 Series, you need to purchase appropriate emulation pod and/or emulation processor.

The HP 64146A/B #001 emulator is provided with no emulation pod. You need to purchase appropriate emulation pod and emulation processor listed in Table 1-1.

To purchase emulation pod or emulation processor, contact the address listed in the manual provided with your emulation pod.

The list of supported microprocessors in Table 1-1 is not necessarily complete. To determine if your microprocessor is supported or not, contact Hewlett-Packard.

Features of the 7700 Series Emulator

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

Clock Speed

The HP 64146-61001 and HP 64146-61002 emulation pod generate internal clock of 1 MHz. These emulation pods can be used with target system clock up to 25 MHz.

The emulator can run with no wait state up to 25 MHz. When clock is faster than 16 MHz, you can use the emulator with one of the following methods.

- Insert one wait state by the RDY signal. The emulator can be configured to generate the RDY signal. Also, the emulator accepts RDY signal from the target system.
- Use the high speed access mode of the emulator. The emulator can run with no wait state. However, there is a limitation in the mapping of the emulation memory in this mode. Refer to Chapter 4 of this manual for more detail.

Emulation memory

The HP 64146A/B 7700 Series emulator is used with one of the following Emulation Memory Cards.

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

The emulation memory can be configured into 256 byte blocks. A maximum of 16 ranges can be configured as emulation RAM (eram), emulation ROM (erom), target system RAM (tram), target system ROM (trom), or guarded memory (grd). The HP 64146A/B 7700

Series emulator will attempt to break to the emulation monitor upon accessing guarded memory; additionally, you can configure the emulator to break to the emulation monitor upon performing a write to ROM (which will stop a runaway program).

Analysis

The HP 64146A/B 7700 Series emulator is used with one of the following analyzers which allows you to trace code execution and processor activity.

- HP 64704 80-channel Emulation Bus Analyzer
- HP 64703 64-channel Emulation Bus Analyzer and 16-channel State/Timing Analyzer
- HP 64794A/C/D 80-channel 8K/64K/256K 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.

Foreground or Background Emulation Monitor

When you power up the emulator, or when you initialize it, the background monitor is used by default. You can also configure the emulator to use a foreground monitor. Before the background and foreground monitors are described, you should understand the function of the emulation monitor program.

The Function of the Monitor Program

The monitor program is the interface between the emulation system controller and the target system. The emulation system controller uses its own microprocessor to accept and execute emulation, system, and analysis commands. The monitor program is executed by the emulation processor.

The monitor program makes possible emulation commands which access target system resources. (The only way to access target system resource is through the emulation processor.) For example, when you enter a command to modify target system memory, it is the execution of monitor program instructions that cause the new values to be written to target system memory.

The Background Monitor

On emulator power-up, or after initialization, the emulator uses the background monitor program. The background monitor does not occupy processor address space.

The Foreground Monitor

You can configure the emulator to use a foreground monitor program. When a foreground monitor is selected it executes in the foreground emulator mode. The foreground monitor occupies processor memory space and executes as if it were part of your program.

Register Display and Modification

You can display or modify the 7700 Series internal register contents. This includes the ability to modify the program counter (PC) and the program bank register (PG) values so you can control where the emulator starts a program run.

Single-Step

When you are using the background monitor, you can direct the emulation processor to execute a single instruction or a specified number of instructions.

Breakpoints

You can set the emulator/analyzer interaction so the emulator will break to the monitor program when the analyzer finds a specific state or states, allowing you to perform post-mortem analysis of the program execution. You can also set software breakpoints in your program. This feature is realized by inserting BRK instructions into user program. Refer to the "Using Software Breakpoints" section of "Getting Started" chapter for more information.

Real Time Operation

Real-time signifies continuous execution of your program at full rated processor speed without interference from the emulator. (Such interference occurs when the emulator needs to break to the monitor to perform an action you requested, such as displaying target system memory.) Emulator features performed in real time include: running and analyzer tracing. Emulator features not performed in real time include: display or modify of target system memory; load/dump of target memory, display or modification of registers, and single step.

Coverage Measurements

Coverage memory is provided for the processor's external program memory space. This memory allows you to perform coverage measurements on programs in emulation memory.

Reset Support

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

Watch Dog Timer

You can configure the emulator to disable the watch dog timer.

Limitations, Restrictions

Access to Internal RAM

Modifying internal RAM or SFR suspends user program execution.

Trace Internal RAM

Read data from the internal RAM or SFR is not traced correctly by the emulation analyzer.

Note

Write data is also not traced correctly, when the following conditions are met:

- The emulator is used with the M37795 emulation pod.
- The processor is operating in the memory expansion or microprocessor mode with 8 bit external bus.

DMA Support

Direct memory access to emulation memory is not allowed.

Watch Dog Timer in Background

Watch dog timer suspends count down while the emulator is running in background monitor.

Step Command with Foreground Monitor

Step command is not available when the emulator is used with a foreground monitor.

Step Command and Interrupts

When an interrupt occurs while the emulator is running in monitor, the emulator fails to do the first step operation. The emulator will display the mnemonic of the instruction which should be stepped, but the instruction is not actually executed. The second step operation will step the first instruction of the interrupt routine.

Emulation Commands in Stop/Wait Mode

When the 7700 microprocessor is in the stop or wait mode, emulation commands which access memory or registers will fail. You need to break the emulator into the monitor to use these commands. Once you break the emulator into the monitor, the stop or wait mode will be released.

Stack Address

In some versions of 7700 microprocessor, the stack can be located in Bank FF. However, the HP 64146A/B 7700 Series emulator doesn't support the feature. The stack must be located in Bank 0.

2

Getting Started

Introduction

This chapter will lead you through a basic, step by step tutorial designed to familiarize you with the use of the HP 64700 emulator for the 7700 Series microprocessor. When you have completed this chapter, you will be able to perform these tasks:

- Set up an emulation configuration for out of circuit emulation use
- Transfer a small program into emulation memory
- Use run/stop controls to control operation of your program
- Use memory manipulation features to alter the program's operation
- Use analyzer commands to view the real time execution of your program

Before You Begin

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

1. Completed Hardware installation

   Complete hardware installation of the HP 64700 emulator in configuration you intend to use for your work:

   – Standalone configuration
   – Transparent configuration
   – Remote configuration

   References: *The HP 64700 Series Installation/Service* manual

2. Connected the emulation pod to the emulator

   Connect the emulation pod to the emulator as shown in Figure 2-1.

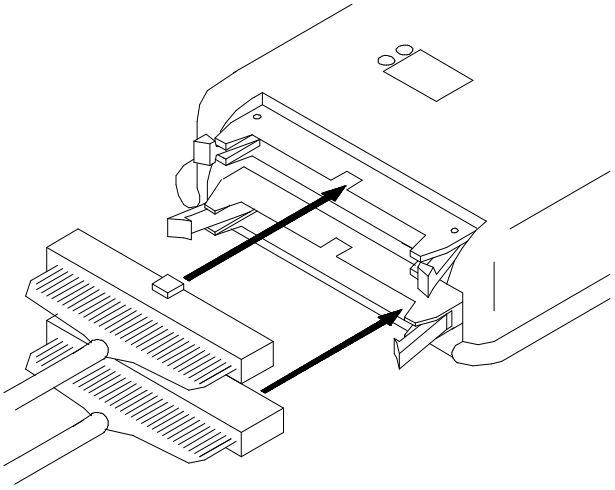


Figure 2-1. Connecting the Emulation Pod

Caution

Turn off power of the emulator before inserting the cable to the emulation pod to avoid circuit damage.

If you have properly completed steps above, you should be able to hit <RETURN> (or <ENTER> on some keyboards) and get a prompt on your terminal screen:

If you do not see any prompt, 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 "COMMAND_READER" used in this chapter is shown Figure 2-2. The program emulates a primitive command interpreter.

Note

This sample program is written for Mitsubishi RASM77 Assembler.

Data Declarations

INPUT_POINTER and OUTPUT_POINTER define the address locations of an input area and an output area to be used by the program. MESSAGE_A, MESSAGE_B and INVALID_INPUT are the messages used by the program to respond to various command inputs.

Initialization

First, the INIT routine set up the stack pointer. Next, locations of the input area is cleared by the CLEAR_INPUT routines.

READ_INPUT

This routine continuously reads the byte at location 100 hex until it is something other than a null character (00 hexadecimal); when this occurs, the CLEAR_OUTPUT routine clears the output area, and the PROCESS_COMM routine is executed.

```
 1                                              .DP             0
 2                                              .DT             0
 3
 4                                              .SECTION        BUFFER
 5                                              .ORG            100H
 6 (000100)      1H BYTE     INPUT_POINTER:     .BLKB           1
 7                                              .ORG            200H
 8 (000200)      1H BYTE     OUTPUT_POINTER:    .BLKB           1
 9
10                                              .SECTION        TABLE
11                                              .ORG            0D000H
12  00D000  544849532049     MESSAGE_A:         .BYTE           'THIS IS MESSAGE A'
    00D006  53204D455353
    00D00C  4147452041
13  00D011  544849532049     MESSAGE_B:         .BYTE           'THIS IS MESSAGE B'
    00D017  53204D455353
    00D01D  4147452042
14  00D022  494E56414C49     INVALID_INPUT:     .BYTE           'INVALID COMMAND'
    00D028  4420434F4D4D
    00D02E  414E44
15
16                                              .SECTION        SAMPPROG
17                                              .DATA           8
18                                              .INDEX          16
19                                              .ORG            0C000H
20  00C000  A27F02           INIT:              LDX             #27FH
21  00C003  9A                                  TXS
22  00C004  F8                                  SEM
23  00C005  42A900           CLEAR_INPUT:       LDA             B,#00H
24  00C008  428D0001      L                     STA             B,DT:INPUT_POINTER
25  00C00C  AD0001        L  READ_INPUT:        LDA             A,DT:INPUT_POINTER
26  00C00F  C900                                CMP             A,#00H
27  00C011  F0F9          L                     BEQ             READ_INPUT
28
29                                              .INDEX          8
30  00C013  E210                                SEP             X
31  00C015  A200             CLEAR_OUTPUT:      LDX             #00H
32  00C017  A020                                LDY             #20H
33  00C019  429D0002      L  CLEAR_LOOP:        STA             B,DT:OUTPUT_POINTER,X
34  00C01D  E8                                  INX
35  00C01E  88                                  DEY
36  00C01F  D0F8          L                     BNE             CLEAR_LOOP
37
38                                              .INDEX          16
39  00C021  C210                                CLP             X
40  00C023  C941             PROCESS_COMM:      CMP             A,#41H
41  00C025  F006          L                     BEQ             COMMAND_A
42  00C027  C942                                CMP             A,#42H
43  00C029  F009          L                     BEQ             COMMAND_B
44  00C02B  800E          L                     BRA             UNRECOGNIZED
45  00C02D  A911             COMMAND_A:         LDA             A,#11H
46  00C02F  A200D0        L                     LDX             #MESSAGE_A
47  00C032  800C          L                     BRA             OUTPUT
48  00C034  A911             COMMAND_B:         LDA             A,#11H
49  00C036  A211D0        L                     LDX             #MESSAGE_B
50  00C039  8005          L                     BRA             OUTPUT
```

Figure 2-2. Sample Program Listing

2-4 Getting Started

```
51  00C03B  A90F            UNRECOGNIZED:   LDA             A,#0FH
52  00C03D  A222D0      L                   LDX             #INVALID_INPUT
53  00C040  A00002      L   OUTPUT:         LDY             #OUTPUT_POINTER
54  00C043  540000                          MVN             0,0
55  00C046  80BD        L                   BRA             CLEAR_INPUT
56
57                                          .END
```

Figure 2-2. Sample Program Listing (Cont'd)

PROCESS_COMM

Compares the input byte (now something other than a null) to the possible command bytes of "A" (ASCII 41 hex) and "B" (ASCII 42 hex), then jumps to the appropriate set up routine for the command message. If the input byte does not match either of these values, a branch to a set up routine for an error message is executed.

COMMAND_A, COMMAND_B, UNRECOGNIZED

These routines set up the proper parameters for writing the output message: the number of bytes in the message is moved to accumulator A and the base address of the message in the data area is moved to index register X.

OUTPUT

First, the base address of the output area is moved to index register Y. Finally, the proper message is written to the output area by the MVN instruction. When done, OUTPUT jumps back to CLEAR_INPUT and the command monitoring process begins again.

Using the various features of the emulator, we will show you how to load this program into emulation memory, execute it, monitor the program's operation with the analyzer, and simulate entry of different commands utilizing the memory access commands provided by the HP 64700 command set.

Using the Help Facility

If you need a quick reference to the Terminal Interface syntax, you can use the built-in **help** facilities. For example, to display the top level **help** menu, type:

```
R> help
```

```
help  - display help information

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

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

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

You can type the **?** symbol instead of typing **help**. For example, if you want a list of commands in the **emul** command group, type:

```
R> ? emul
```

```
emul - emulation commands
------------------------------------------------------------------------------
b......break to monitor  dump...dump memory       r......run user code
bc.....break condition   es.....emulation status  reg....registers
bp.....breakpoints       io.....input/output      rst....reset
cf.....configuration     load...load memory       rx.....run at CMB execute
cim....copy target image m......memory            s......step
cmb....CMB interaction   map....memory mapper     ser....search memory
cov....coverage          mo.....modes
cp.....copy memory       mx.....mx
```

To display help information for any command, just type **help** (or **?**) and the command name. For example:

```
R> help load
```

```
load - download absolute file into processor memory space

  load -i          - download intel hex format
  load -m          - download motorola S-record format
  load -t          - download extended tek hex format
  load -S          - download symbol file
  load -h          - download hp format (requires transfer protocol)
  load -a          - reserved for internal hp use
  load -e          - write only to emulation memory
  load -u          - write only to target memory
  load -o          - data received from the non-command source port
  load -s          - send a character string out the other port
  load -b          - data sent in binary (valid with -h option)
  load -x          - data sent in hex ascii (valid with -h option)
  load -q          - quiet mode
  load -p          - record ACK/NAK protocol (valid with -imt options)
  load -c <file>   - data is received from the 64000. file name format is:
                     <filename>:<userid>:absolute
```

Initialize the Emulator to a Known State

To initialize the emulator to a known state for this tutorial:

Note

It is especially important that you perform the following step if the emulator is being operated in a standalone mode controlled by only a data terminal. The only program entry available in this mode is through memory modification; consequently, if the emulator is reinitialized, emulation memory will be cleared and a great deal of tedious work could be lost.

1. Verify that no one else is using the emulator or will have need of configuration items programmed into the emulator.

2. Initialize the emulator by typing the command:

```
R> init -p
```

Set Up the Proper Emulation Configuration

Set Up Emulation Conditions

To set the emulator's configuration values to the proper state for this tutorial, do this:

1. Type:

R> **cf**

You should see the following configuration items displayed:

```
cf chip=7702M2
cf isfr=0..07f
cf iram=080..27f
cf irom=0c000..0ffff
cf ipmr=05e
cf mode=single
cf mon=bg
cf clk=int
cf int=en
cf rdy=dis
cf rush=dis
cf wdog=dis
cf rsp=27f
cf rrt=dis
cf dmdt=dis
cf tdma=dis
cf trfsh=dis
cf thold=dis
```

Note

The individual configuration items won't be explained in this example; refer to Chapter 4 of this manual and the *User's Reference* manual for details.

2. You need to select chip you emulate. You can select chip with the following command:

R> **cf chip=<chip_name>**

You must enter appropriate <chip_name> to your processor. Valid <chip_name> are listed in Table 2-1. The default <chip_name> 7702M2 is applied to M37702M2 and M37703M2 processors.

<chip_name>	Processor
7700M2	M37700M2-xxxFP M2AxxxFP M37701M2-xxxSP M2AxxxSP
7700M4	M37700M4-xxxFP M4AxxxFP M37701M4-xxxSP M4AxxxSP
7700S	M37700SFP SAFP M37701SSP SASP
7700S4	M37700S4FP S4AFP M37701S4SP S4ASP
7702M2	M37702M2-xxxFP M2AxxxFP M2BxxxFP M37703M2-xxxSP M2AxxxSP M2BxxxSP
7702M4	M37702M4-xxxFP M4AxxxFP M4BxxxFP M37703M4-xxxSP M4AxxxSP M4BxxxSP
7702M6	M37702M6BxxxFP M6LxxxFP
7702S1	M37702S1FP S1AFP S1BFP M37703S1SP S1ASP S1BSP
7702S4	M37702S4FP S4AFP S4BFP M37703S4SP S4ASP S4BSP
7704M2	M37704M2-xxxFP M2AxxxFP M37705M2-xxxSP M2AxxxSP
7704M3	M37704M3BxxxFP
7704M4	M37704M4BxxxFP
7704S1	M37704S1FP S1AFP M37705S1SP S1ASP
7710M4	M37710M4BxxxFP
7710S4	M37710S4BFP
7720S1	M37720S1FP S1AFP
7730S2	M37730S2FP S2AFP S2SP S2ASP
7732S4	M37732S4FP S4AFP
7780S	M37780STJ STFP
7781M4	M37781M4TxxxJ M4TxxxFP
7781E4	M37781E4TxxxJ E4TxxxFP
7795S	M37795SJ STJ
7796E4	M37796E4-xxxJ E4TxxxJ E4TxxxFP

Table 2-1. <chip_name> for cf chip Command

3. Now, set up reset value for the stack pointer.

R> **cf rsp=27f**

Note

Without a stack pointer, the emulator is unable to make the transition to the run state, step, or perform many other emulation functions.

4. Let's go ahead and set up the proper break conditions. Type:

R> **bc**

You will see:

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

For each break condition that does not match the one listed, use one of the following commands:

To enable break conditions that are currently disabled, type:

R> **bc -e <breakpoint type>**

To disable break conditions that are currently enabled, type:

R> **bc -d <breakpoint type>**

For example, if typing **bc** gives the following list of break conditions:

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

(items in **bold** indicate improper values for this example)

Type the following commands to set the break conditions correctly for this example:

```
R> bc -d rom
```

(this enables the write to ROM break)

```
R> bc -d trig1 trig2
```

(this disables break on triggers from the analyzer)

5. To avoid problems later while modifying and displaying memory locations, type:

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

This sets the access and display mode for memory operation byte.

Map Memory

The emulation memory can be configured as you desire. You can define emulation memory as emulation RAM, emulation ROM, target RAM, target ROM or guarded memory.

We will use the default mapping for this sample program. To see the default mapping, type:

```
R> map
```

You will see similar display to the following:

```
# remaining number of terms   : 15
# remaining emulation memory  : 1b800h bytes
map 000c000..000ffff     erom    # term 1
map   other tram
```

Note

You don't have to map the internal RAM area, since the emulator uses the internal RAM of emulation processor to perform emulation. If your processor has no internal RAM, map address 100 hex through 2ff hex with the following command for this tutorial.

```
R> map 100..2ff eram
```

Transfer Code into Emulation Memory

From a Terminal in Standalone Configuration

To transfer code into emulation memory from a data terminal running in standalone mode, you must use the modify memory commands. This is necessary because you have no host computer transfer facilities to automatically download the code for you (as if you would if you were using the transparent configuration or the remote configuration.) To minimize the effects of typing errors, you will modify only one row of memory at a time in this example. Do the following:

Enter the data information for the program by typing the following commands:

```
R> m 0d000..0d00f=54,48,49,53,20,49,53,20,4d,45,53,53,41,47,45,20
R> m 0d010..0d01f=41,54,48,49,53,20,49,53,20,4d,45,53,53,41,47,45
R> m 0d020..0d02f=20,42,49,4e,56,41,4c,49,44,20,43,4f,4d,4d,41,4e
R> m 0d030=44
```

You could also type the following line instead:

```
R> m 0d000="THIS IS MESSAGE ATHIS IS MESSAGE BINVALID COMMAND"
```

You should now verify that the data area of the program is correct by typing:

```
R> m 0d000..0d030
```

You should see:

```
000d000..000d00f     54 48 49 53 20 49 53 20 4d 45 53 53 41 47 45 20
000d010..000d01f     41 54 48 49 53 20 49 53 20 4d 45 53 53 41 47 45
000d020..000d02f     20 42 49 4e 56 41 4c 49 44 20 43 4f 4d 4d 41 4e
000d030..000d030     44
```

If this is not correct, you can correct the errors by re-entering only the modify memory commands for the particular rows of memory that are wrong.

For example,if row d000..d00f shows these values:

```
000d000..000d00f     54 48 49 53 20 20 49 53 20 4d 45 53 53 41 47 45
```

you can correct this row of memory by typing:

```
R> m 0d000..0d00f=54,48,49,53,20,49,53,20,4d,45,53,53,41,47,45,20
```

Or, you might need to modify only one location, as in the instance where address d00f equals 22 hex rather than 20 hex. Type:

```
R> m 0d00f=20
```

Enter the program information by typing the following commands:

```
R> m 0c000..0c00f=0a2,7f,02,9a,0f8,42,0a9,00,42,8d,00,01,0ad,00,01,0c9
R> m 0c010..0c01f=00,0f0,0f9,0e2,10,0a2,00,0a0,20,42,09d,00,02,0e8,88,0d0
R> m 0c020..0c02f=0f8,0c2,10,0c9,41,0f0,06,0c9,42,0f0,09,80,0e,0a9,11,0a2
R> m 0c030..0c03f=00,0d0,80,0c,0a9,11,0a2,11,0d0,80,05,0a9,0f,0a2,22,0d0
R> m 0c040..0c047=0a0,00,02,54,00,00,80,0bd
```

(note the hex letters must be preceded by a digit)

You should now verify that the program area is correct by typing:

```
R> m 0c000..0c047
```

You should see:

```
000c000..000c00f  a2 7f 02 9a f8 42 a9 00 42 8d 00 01 ad 00 01 c9
000c010..000c01f  00 f0 f9 e2 10 a2 00 a0 20 42 9d 00 02 e8 88 d0
000c020..000c02f  f8 c2 10 c9 41 f0 06 c9 42 f0 09 80 0e a9 11 a2
000c030..000c03f  00 d0 80 0c a9 11 a2 11 d0 80 05 a9 0f a2 22 d0
000c040..000c047  a0 00 02 54 00 00 80 bd
```

If this is not correct, you can correct the errors by re-entering only the modify memory commands for the particular rows of memory that are wrong.

Transparent Configuration

If your emulator is connected between a terminal and a host computer, you can down load programs into memory using the **load** command with the **-o** (from other port) option.

First, you must establish communications with your host computer through the transparent mode link provided in the HP 64700. Type:

```
R> xp -e
```

If you then press <RETURN> a few times, you should see:

```
login:
login:
login:
```

This is the login prompt for an HP-UX host system. (Your prompt may differ depending on how your system manager has configured your system.) Log in to your host system.

Disable the transparent mode so that your terminal will talk directly to the emulator. Type:

```
$ <ESC>g xp -d
```

The "<ESC>g" sequence temporarily toggles the transparent mode so that the emulator will accept commands; "xp -d" then fully disables the transparent mode.

The succeeding procedure is different depending on the format you are going to download.

HP Absolute

If you have a Softkey Interface, a file format converter is provided with it. The converter can convert MELPS 7700 Hex format files into HP Absolute files. (Refer to Softkey Interface User's Guide for more details.) Downloading the HP Absolute requires the **transfer** protocol. The example below assumes that the transfer utility has been installed on the HP 9000/300 host computer.

```
R> load -hbo <RETURN>
transfer -rtb cmd_rds.X <ESC>g
##########
R>
```

Intel Hex format

The example which follow shows how to download the Intel hexadecimal files.

```
R> load -io  <RETURN>
cat cmd_rds.hex<ESC>g
##########
Data records received = 00009
R>
```

Note

The emulator can load the MELPS 7700 Hex format files with **load -i** command when the program is located address fffff hex or less.

At this point you should examine a portion of memory to verify that your code was loaded correctly.

Type:

```
R> m 0c000..0c047
```

You should see:

```
000c000..000c00f  a2 7f 02 9a f8 42 a9 00 42 8d 00 01 ad 00 01 c9
000c010..000c01f  00 f0 f9 e2 10 a2 00 a0 20 42 9d 00 02 e8 88 d0
000c020..000c02f  f8 c2 10 c9 41 f0 06 c9 42 f0 06 80 0e a9 11 a2
000c030..000c03f  00 d0 80 0c a9 11 a2 11 d0 80 05 a9 0f a2 22 d0
000c040..000c047  a0 00 02 54 00 00 80 bd
```

Looking at Your Code

Now that you have loaded your code into emulation memory, you can display it in mnemonic format. Before displaying memory in mnemonic format, you need to tell the emulator what value of M flag and X flag should be used to disassemble the memory contents. This is needed because the length of operand is variable according to M flag and X flag. Type:

```
R> mx -m0 -x0
```

To display memory in mnemonic format, type:

```
R> m -dm 0c000..0c047
```

You will see:

```
000c000  -              LDX #027fH
000c003  -              TXS
000c004  -              SEM
000c005  -              LDA B,#00H
000c008  -              STA B,DT:0100H
000c00c  -              LDA A,DT:0100H
000c00f  -              CMP A,#00H
000c011  -              BEQ 00c00cH
000c013  -              SEP #10H
000c015  -              LDX #00H
000c017  -              LDY #20H
000c019  -              STA B,DT:0200H,X
000c01d  -              INX
000c01e  -              DEY
000c01f  -              BNE 00c019H
000c021  -              CLP #10H
000c023  -              CMP A,#41H
000c025  -              BEQ 00c02dH
000c027  -              CMP A,#42H
000c029  -              BEQ 00c034H
000c02b  -              BRA 00c03bH
000c02d  -              LDA A,#11H
000c02f  -              LDX #d000H
```

```
000c032  -              BRA 00c040H
000c034  -              LDA A,#11H
000c036  -              LDX #d011H
000c039  -              BRA 00c040H
000c03b  -              LDA A,#0fH
000c03d  -              LDX #d022H
000c040  -              LDY #0200H
000c043  -              MVN 00H,00H
000c046  -              BRA 00c005H
```

When the inverse assembler encounters the following instruction, the **mx** command is set up automatically.

- SEM
- CLM
- SEP X
- CLP X
- SEP M
- CLP M

In the above example, the **mx** command is set up as following:

- SEM (0c004) **mx -m1 -x0**
- SEP X (0c013) **mx -m1 -x1**
- CLP X (0c021) **mx -m1 -x0**

When you display memory in mnemonic format without specifying the **mx** command, the last setting is used to disassemble the memory contents. Type:

```
R> m 0c015..0c02f
```

```
000c015  -              LDX #a000H
000c018  -              JSR PG:9d42H
000c01b  -              BRK
000c01d  -              INX
000c01e  -              DEY
000c01f  -              BNE 00c019H
000c021  -              CLP #10H
000c023  -              CMP A,#41H
000c025  -              BEQ 00c02dH
000c027  -              CMP A,#42H
000c029  -              BEQ 00c034H
000c02b  -              BRA 00c03bH
000c02d  -              LDA A,#11H
000c02f  -              LDX #d000H
```

As you can see, the memory contents is not disassembled correctly. To see the correct mnemonic, set up the **mx** command:

```
R> mx -x1
R> m 0c015..0c02f
```

```
000c015  -              LDX #00H
000c017  -              LDY #20H
000c019  -              STA B,DT:0200H,X
000c01d  -              INX
000c01e  -              DEY
000c01f  -              BNE 00c019H
000c021  -              CLP #10H
000c023  -              CMP A,#41H
000c025  -              BEQ 00c02dH
000c027  -              CMP A,#42H
000c029  -              BEQ 00c034H
000c02b  -              BRA 00c03bH
000c02d  -              LDA A,#11H
000c02f  -              LDX #d000H
```

Familiarize Yourself with the System Prompts

Note

The following steps are not intended to be complete explanations of each command; the information is only provided to give you some idea of the meanings of the various command prompts you may see and reasons why the prompt changes as you execute various commands.

You should gain some familiarity with the HP 64700 emulator command prompts by doing the following:

Ignore the current command prompt. Type:

```
*> rst
```

You will see:

```
R>
```

The rst command resets the emulation processor and holds it in the reset state. The "R>" prompt indicates that the processor is reset.

Type:

```
R> r 0c000
```

You will see:

```
U>
```

The r command runs the processor from address 2000 hex.

Type:

```
U> b
```

You will see:

```
M>
```

The **b** command causes the emulation processor to "break" execution of whatever it was doing and begin executing within the emulation monitor. The "M>" prompt indicates that the emulator is running in the monitor.

Running the Sample Program

Type:

```
M> r 0c000
```

The emulator changes state from background to foreground and begins running the sample program from location 0c000 hex.

Note

The default number base for address and data values within HP 64700 is hexadecimal. Other number bases may be specified. Refer to the *HP 64700 User's Reference* manual for further details.

Let's look at the registers to verify that the stack pointer was properly initialized. Type:

```
U> reg
```

You will see:

```
reg pg=00 pc=c00c ps=0023 dt=00 sp=027f a=ff00 b=0000 x=027f y=0017 dpr=0000
```

Notice that sp contains 27f hex.

Verify that the input area command byte was cleared during initialization.

Type:

```
U> m -db 100
```

You will see:

```
0000100..0000100     00
```

The input byte location was successfully cleared.

Now we will use the emulator features to make the program work. Remember that the program writes specific messages to the output area depending on what the input byte location contains. Type:

```
U> m 100=41
```

This modifies the input byte location to the hex value for an ASCII "A". Now let's check the output area for a message.

```
U> m 200..21f
```

You will see:

```
  0000200..000020f  54 48 49 53 20 49 53 20 4d 45 53 53 41 47 45 20
  0000210..000021f  41 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

These are the ASCII values for MESSAGE_A.

Repeat the last two commands twice. The first time, use 42 instead of 41 at location 100 and note that MESSAGE_B overwrites MESSAGE_A. Then try these again, using any number except 00, 41, or 42 and note that the INVALID_INPUT message is written to this area.

Note 

When you modify/display internal RAM or SFR, the emulator breaks into the monitor, and the monitor program reads the contents of memory. This is because the emulator uses internal RAM and SFR of emulation processor to perform emulation. Note that user program execution is suspended to modify/diplay internal RAM or SFR.

Note

You can configure the emulator so that data write cycles are performed to both internal RAM (or SFR) and emulation memory. In this case, you can display the data written to emulation memory without suspending user program execution. Refer to chapter 4 of this manual for more details.

Stepping Through the Program

You can also direct the emulator processor to execute one instruction or number of instructions. Type:

```
M> s 1 0c000;reg
```

This command steps 1 instruction from address 0c000 hex, and displays registers. You will see:

```
000c000  -               LDX #027fH
PC = 000c003
reg pg=00 pc=c003 ps=0021 dt=00 sp=027f a=ff00 b=0000 x=027f y=0021 dpr=0000
```

To step one instruction from present PC, you only need to type **s** at prompt. Type:

```
M> s;reg
```

You will see:

```
000c003  -               TXS
PC = 000c004
reg pg=00 pc=c004 ps=0021 dt=00 sp=027f a=ff00 b=0000 x=027f y=0021 dpr=0000
```

Note

When the you use the **s** command, the disassembled mnemonic displayed may not be accurate. You need to set up **mx** command before using the **s** command to see the correct mnemonic.

Note 

When the emulator performs single step execution with **s** command, all memory access is performed by byte access.

Easy Command Entry

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
```

```
000c004  -              SEM
  PC = 000c005
  reg pg=00 pc=c005 ps=0021 dt=00 sp=027f a=ff00 b=0000 x=027f y=0021 dpr=0000
```

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.

Tracing Program Execution

Note

For this example, you will be using the analyzer in the easy configuration, which simplifies the process of analyzer measurement setup. The complex configuration allows more powerful measurements, but requires more interaction from you to set up those measurements. For more information on easy and complex analyzer configurations and the analyzer, refer to the *HP 64700 Analyzer User's Guide* and the *User's Reference*.

Now let's use the emulation analyzer to trace execution of the program. Suppose that you would like to start the trace when the analyzer begins writing data to the message output area. You can do this by specifying analyzer trigger upon encountering write status to address 200 hex. Furthermore, you might want to store only the data written to the output area. This can be accomplished by modifying what is known as the "analyzer storage specification".

Now let's set the trigger specification. Type:

```
M> tg addr=200 and stat=write
```

To store only the accesses to the address range 200 hex through 21f hex, type:

```
M> tsto addr=200..21f
```

Let's change the data format of the trace display so that you will see the output message writes displayed in ASCII format:

```
M> tf addr,h data,a mne count,r seq
```

Start the trace by typing:

```
M> t
```

You will see:

```
Emulation trace started
```

To start the emulation run, type:

```
M> r 0c000
```

Now, you need to have a "command" input to the program so that the program will jump to the output routines (otherwise the trigger will not be found, since the program will never access address 200 hex). Type:

```
U> m 100=41
```

To display the trace list, type:

```
U> tl 0..10
```

You will see:

```
 Line   addr,H  data,A  M7700 Mnemonic,H                        count,R  seq
-----   ------  ------  ----------------------------------  ---------  ---
    0   000200  ..          xx00H  data write       mx            ---   +
    1   000201  ..          00xxH  data write       mx        56.00 uS   .
    2   000202  ..          xx00H  data write       mx        56.00 uS   .
    3   000203  ..          00xxH  data write       mx        56.00 uS   .
    4   000204  ..          xx00H  data write       mx        56.00 uS   .
    5   000205  ..          00xxH  data write       mx        56.00 uS   .
    6   000206  ..          xx00H  data write       mx        56.00 uS   .
    7   000207  ..          00xxH  data write       mx        56.00 uS   .
    8   000208  ..          xx00H  data write       mx        56.00 uS   .
    9   000209  ..          00xxH  data write       mx        56.00 uS   .
   10   00020a  ..          xx00H  data write       mx        56.00 uS   .
```

The above list shows the execution of the CLEAR_OUTPUT routine which cleared the output area. To see the states that the program wrote the message to the output area, type:

```
U> tl 30..40
```

```
 Line   addr,H  data,A  M7700 Mnemonic,H                        count,R  seq
-----   ------  ------  ----------------------------------  ---------  ---
   30   00021e  ..          xx00H  data write       mx        56.00 uS   .
   31   00021f  ..          00xxH  data write       mx        56.00 uS   .
   32   000200  HT          4854H  data write       mx        134.0 uS   .
   33   000202  SI          5349H  data write       mx        18.00 uS   .
   34   000204  I.          4920H  data write       mx        18.00 uS   .
   35   000206  .S          2053H  data write       mx        18.00 uS   .
   36   000208  EM          454dH  data write       mx        18.00 uS   .
   37   00020a  SS          5353H  data write       mx        18.00 uS   .
   38   00020c  GA          4741H  data write       mx        18.00 uS   .
   39   00020e  .E          2045H  data write       mx        18.00 uS   .
   40
```

If you look at the last lines of the trace listing, you will notice that the analyzer seems to have stored only part of the output message, even though you specified more than the full range needed to store all of the message. The reason for this is that the analyzer has a storage pipeline,

which holds states that have been acquired but not yet written to trace memory. To see all of the states, halt the analyzer by typing:

```
U> th
```

You will see:

```
Emulation trace halted
```

Now display the trace list:

```
U> tl 30..40
```

You will see:

```
 Line   addr,H  data,A  M7700 Mnemonic,H                        count,R  seq
-----   ------  ------  ----------------------------------  ---------  ---
   30   00021e  ..          xx00H  data write       mx        56.00 uS   .
   31   00021f  ..          00xxH  data write       mx        56.00 uS   .
   32   000200  HT          4854H  data write       mx        134.0 uS   .
   33   000202  SI          5349H  data write       mx        18.00 uS   .
   34   000204  I.          4920H  data write       mx        18.00 uS   .
   35   000206  .S          2053H  data write       mx        18.00 uS   .
   36   000208  EM          454dH  data write       mx        18.00 uS   .
   37   00020a  SS          5353H  data write       mx        18.00 uS   .
   38   00020c  GA          4741H  data write       mx        18.00 uS   .
   39   00020e  .E          2045H  data write       mx        18.00 uS   .
   40   000210  TA          xx41H  data write       mx        16.00 uS   .
```

As you can see, all of the requested states have been captured by the analyzer.

Predefined Status Equates

Common values for the status trace signals have been predefined. One of these equates "write" was used in the above example. You can see these equates with the following command.

```
U> equ
```

```
### Equates ###
equ bg=0x1xxxxxxy
equ byte=0xx1x1x1xy
equ cpu=0xx11xxxxy
equ data=0xx1x10xxy
equ dma=0xx10xxxxy
equ exec=0xx1101xxy
equ fetch=0xx1111x1y
equ fg=0x0xxxxxxy
equ hold=0xx01xxxxy
equ mx=1xxxxxxxy
equ read=0xx1x1xx1y
equ ref=0xx00xxxxy
equ word=0xx1x1x0xy
equ write=0xx1x1xx0y
```

Using Software Breakpoints

You can stop program execution at specific address by using **bp** (software breakpoint) command. When you define a software breakpoint to a certain address, the emulator will replace the opcode with BRK instruction as software breakpoint instruction. When the emulator detects the BRK instruction, user program breaks to the monitor, and the original opcode will be placed at the breakpoint address. A subsequent run or step command will execute from this address.

If the BRK instruction was not inserted as the result of **bp** command (in other words, it is part of the user program), the "Undefined software breakpoint" message is displayed.

Note

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

Note

Because software breakpoints are implemented by replacing opcodes with the software breakpoint instruction, you cannot define software breakpoints in target ROM. You can, however, copy target ROM into emulation memory by **cim** command when you are using the background monitor. (Refer to *HP 64700 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 following commands.

```
M> bc
```

```
bc -d bp #disable
bc -d rom #disable
bc -d bnct #disable
```

```
bc -d cmbt #disable
bc -d trig1 #disable
bc -d trig2 #disable
```

```
M> bc -e bp
```

Defining a Software Breakpoint

Now that the software breakpoint is enabled, you can define software breakpoints. Enter the following command to break on the address of the OUTPUT label.

```
M> bp 0c040
```

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

```
M> r 0c000
U> m 100=41
!ASYNC_STAT  615! Software break point: 000c040
```

```
M> reg
reg pg=00 pc=c040 ps=00a1 dt=00 sp=027f a=0011 b=0000 x=d000 y=0000 dpr=0000
```

Notice that PC contains c040.

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

```
M> bp
###BREAKPOINT FEATURE IS ENABLED###
bp 000c040 #disabled
```

```
M> bp -e 0c040
M> bp
###BREAKPOINT FEATURE IS ENABLED###
bp 000c040 #enabled
```

```
M> r 0c000
U> m 100=41
!ASYNC_STAT  615! Software breakpoint: 000c040
```

```
M> bp
###BREAKPOINT FEATURE IS ENABLED###
bp 000c040 #disabled
```

Searching Memory for Strings or Numeric Expressions

The HP 64700 Emulator provides you with tools that allow you to search memory for data strings or numeric expressions. For example, you might want to know exactly where a string is loaded. To locate the position of the string "THIS IS MESSAGE A" in the sample program. Type:

```
M> ser 0c000..0dfff="THIS IS MESSAGE A"
pattern match at address:  000d000
```

You can also find numeric expressions. For example, you might want to find all of the **BEQ** instructions in the sample program. Since a **BEQ** instruction begins with f0 hex, you can search for that value by typing:

```
M> ser -db 0c000..0c047=0f0

pattern match at address: 000c011
pattern match at address: 000c025
pattern match at address: 000c029
```

Making Program Coverage Measurements

In testing your program, you will often want to verify that all possible code segments are executed. With the sample program, we might want to verify that all of the code is executed if a command "A", command "B", and an unrecognized command are input to the program.

To make this measurement, we must first reset the coverage status.

```
M> cov -r
```

Note

You should **always** reset the coverage status before making a coverage measurement. Any emulator system command which accesses emulation memory will affect the coverage status bit, resulting in measurement errors if the coverage status is not reset.

Now, run the program and input the following commands:

```
                              M> r 0c000
                              U> cov 0c000..0c047
percentage of memory accessed:  % 30.5
                              U> m 100=41
                              U> cov 0c000..0c047
percentage of memory accessed:  % 81.9
                              U> m 100=42
                              U> m 100=43
                              U> cov 0c000..0c047
percentage of memory accessed:  % 100.0
```

Trace Analysis Considerations

Restriction of the Analyzer

There is a restriction on the function of the emulation analyzer. It **cannot** trace the data which is read from internal RAM or SFR area. Such data always appears ff hex in the trace listing. This is because the emulator uses the internal RAM and SFR of emulation processor to perform emulation. Data read from internal RAM or SFR doesn't appear on the analyzer data bus.

As an example, trace the accesses to the INPUT_POINTER.

To initialize the analyzer, type:

```
U> tinit
```

Set up the trigger condition and perform the trace:

```
U> tg addr=0c00c and stat=exec
U> t
U> tl 0..20
```

```
 Line   addr,H  M7700 Mnemonic,H                          count,R  seq
-----   ------  ------------------------------------  ---------  ---
    0   00c00c  LDA A,DT:0100H                         2.000 uS   +
    1   00c00e       c901H  opcode fetch     mx        6.000 uS   .
    2   000100       xxffH  data read        mx        4.000 uS   .
    3   00c00f  CMP A,#00H                             2.000 uS   .
    4   00c010       f000H  opcode fetch     mx        6.000 uS   .
    5   00c011  BEQ 00c00cH                            2.000 uS   .
    6   00c012       e2f9H  opcode fetch               6.000 uS   .
    7   00c014       a210H  opcode fetch               8.000 uS   .
    8   00c00c       00adH  opcode fetch               8.000 uS   .
    9   00c00c  LDA A,DT:0100H                         2.000 uS   .
   10   00c00e       c901H  opcode fetch     mx        6.000 uS   .
   11   000100       xxffH  data read        mx        4.000 uS   .
   12   00c00f  CMP A,#00H                             2.000 uS   .
   13   00c010       f000H  opcode fetch     mx        6.000 uS   .
   14   00c011  BEQ 00c00cH                            2.000 uS   .
   15   00c012       e2f9H  opcode fetch               6.000 uS   .
   16   00c014       a210H  opcode fetch               8.000 uS   .
   17   00c00c       00adH  opcode fetch               8.000 uS   .
   18   00c00c  LDA A,DT:0100H                         2.000 uS   .
   19   00c00e       c901H  opcode fetch     mx        6.000 uS   .
   20   000100       xxffH  data read        mx        4.000 uS   .
```

As you can see in line 2 of the listing above, data read from internal RAM (which should be 00 hex) appears ff hex.

Notes

3

Using the 7700 Series Emulator In-Circuit

When you are ready to use the HP 64146A/B 7700 Series Emulator in conjunction with actual target system hardware, there are some special considerations you should keep in mind.

- installing the emulator probe
- properly configure the emulator

We will cover the first topic in this chapter. For complete details on in-circuit emulation configuration, refer to Chapter 4.

Installing the Target System Probe

Caution

The following precautions should be taken while using the HP 64146A/B 7700 Series Emulator. Damage to the emulator circuitry may result if these precautions are not observed.

Power Down Target System. Turn off power to the user target system and to the HP 64146A/B 7700 Series 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 HP 64146A/B 7700 Series 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.

Caution

You target system **must** have a clock generation circuit. The emulation pod cannot generate clock signal using a ceramic (or quartz crystal) resonator.

Installing the Target System Probe

1. Set up the switches inside the emulation pod. When you are using the HP 64146-61001 or 64146-61002 emulation pod, refer to 7700 Series Emulation Pod User's Guide. When you are using an other emulation pod, refer to the manual provided with your emulation pod.

2. Remove the 7700 Series microprocessor from the target system socket. Note the location of pin 1 on the processor and on the target system socket.

3. Store the microprocessor in a protected environment (such as antistatic foam).

4. Install the target system probe into the target system microprocessor socket. (See figure 3-1.)

5. Turn on power of your target system, and then, turn on the emulator.

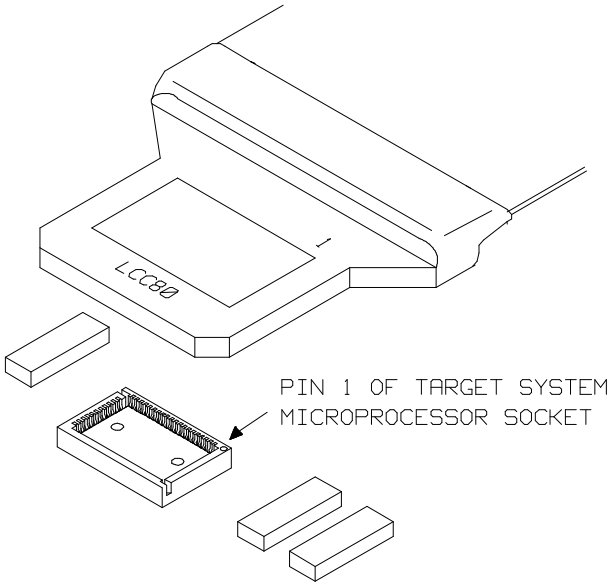


Figure 3-1. Installing the Probe to LCC80 Socket

When your target system uses 64 pin shrink DIP socket, use the adapter as shown in figure 3-2.

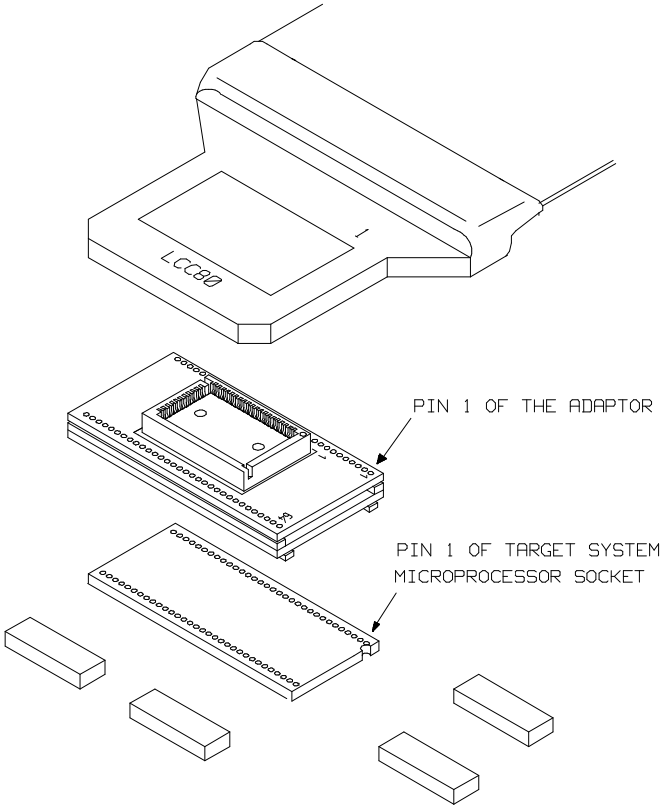


Figure 3-2. Installing the Probe to SDIP64 Socket

4

Configuring the 7700 Series Emulator

In this chapter, we will discuss:

- how to configure the HP 64700 emulator for 7700 Series microprocessor to fit your particular measurement needs.
- some restrictions of HP 64700 emulator for 7700 Series microprocessor.

Types of Emulator Configuration

The HP 64700 Emulator is different from other HP emulators (such as those in the HP 64000-UX system) in that there are several different classes of configuration commands.

Emulation Processor to Emulator/Target System

These are the commands which are generally thought of as "configuration" items in the context of other HP 64000 emulator systems. The commands in this group set up the relationships between the emulation processor and the target system, such as determining how the emulator responds to requests for the processor bus. Also, these commands determine how the emulation processor interacts with the emulator itself; memory mapping and the emulator's response to certain processor actions are some of the items which can be configured.

These commands are the ones which are covered in this chapter.

Commands Which Perform an Action or Measurement

Several of the emulator commands do not configure the emulator; they simply start an emulator program run or other measurement, begin or halt an analyzer measurement, or allow you to display the results of such measurements.

These commands are covered in the examples presented in earlier manual chapters; they are also covered in the HP 64700 Terminal Interface: User's Reference manual.

Coordinated Measurements

These commands determine how the emulator interacts with other measurement instruments, such as external analyzers, or other HP 64700 emulators connected via the CMB (Coordinated Measurement Bus).

These commands are covered in the HP 64700 CMB User's Guide and in the HP 64700 Terminal Interface: User's Reference Manual.

Analyzer

The analyzer configuration commands are those commands which actually specify what type of measurement the analyzer is to make.

Some of the analyzer commands are covered earlier in this manual. You can also refer to the HP 64700 Terminal Interface: Analyzer User's Guide and the HP 64700 Terminal Interface: User's Reference manual.

System

This last group of commands is used by you to set the emulator's data communications protocol, load or dump contents of emulation memory, set up command macros, and so on.

These commands are covered earlier in this manual and in the manual titled HP 64700 Terminal Interface: User's Reference.

Emulation Processor to Emulator/Target System

As noted before, these commands determine how the emulation processor will interact with the emulator's memory and the target system during an emulation measurement.

cf

The **cf** command defines how the emulation processor will respond to certain target system signals. It also defines the type of emulation monitor to be used and optionally defines the location of that monitor in emulation memory.

To see the default configuration settings defined by the **cf** command, type:

```
M> cf
```

You will see:

```
cf chip=7702M2
cf isfr=0..07f
cf iram=080..27f
cf irom=0c000..0ffff
cf ipmr=05e
cf mode=single
cf mon=bg
cf clk=int
cf int=en
cf rdy=dis
cf rush=dis
cf wdog=dis
cf rsp=27f
cf rrt=dis
cf dmdt=dis
cf tdma=dis
cf trfsh=dis
cf thold=dis
```

Let's examine each of these emulator configuration options, with a view towards how they affect the processor's interaction with the emulator.

cf chip

The **chip** configuration item defines the chip you emulate. This command looks a prepared table, and set up the **isfr**, **iram**, **irom** and **rsp** configuration items. The **rsp** configuration item is set to the end of internal RAM area.

```
M> cf chip=<chip_name>
```

Processors supported by this command and their <chip_name> are listed in table 4-1.

```
M> cf chip=other
```

If your processor is not listed in table 4-1, you need to select "others" for this item. In this case, you should set up proper value for **cf isfr**, **cf iram**, **cf irom**, **cf ipmr**, and **cf rsp** by yourself.

Note

Executing this command will drive the emulator into the reset state.

cf isfr

The **isfr** (internal Special Function Register:SFR) configuration item defines the location of SFR area.

Note

This item is automatically set up by **cf chip** command. Therefore, you don't have to set up this item when your processor can be specified by **cf chip** command. If your processor is not supported by **cf chip** command (when you select "other" for the configuration item), you need to set up proper value for **cf isfr** command. Refer to **cf chip** section to determine if your processor is supported by **cf chip** command or not.

```
M> cf isfr=<address>..<address>
     [,<address>..<address>]
```

You can specify the start address and end address of SFR area by above command. The start address and end address of this area can be defined on 16 byte boundaries.

```
M> cf isfr=none
```

<chip_name>	Processor
7700M2	M37700M2-xxxFP M2AxxxFP M37701M2-xxxSP M2AxxxSP
7700M4	M37700M4-xxxFP M4AxxxFP M37701M4-xxxSP M4AxxxSP
7700S	M37700SFP SAFP M37701SSP SASP
7700S4	M37700S4FP S4AFP M37701S4SP S4ASP
7702M2	M37702M2-xxxFP M2AxxxFP M2BxxxFP M37703M2-xxxSP M2AxxxSP M2BxxxSP
7702M4	M37702M4-xxxFP M4AxxxFP M4BxxxFP M37703M4-xxxSP M4AxxxSP M4BxxxSP
7702M6	M37702M6BxxxFP M6LxxxFP
7702S1	M37702S1FP S1AFP S1BFP M37703S1SP S1ASP S1BSP
7702S4	M37702S4FP S4AFP S4BFP M37703S4SP S4ASP S4BSP
7704M2	M37704M2-xxxFP M2AxxxFP M37705M2-xxxSP M2AxxxSP
7704M3	M37704M3BxxxFP
7704M4	M37704M4BxxxFP
7704S1	M37704S1FP S1AFP M37705S1SP S1ASP
7710M4	M37710M4BxxxFP
7710S4	M37710S4BFP
7720S1	M37720S1FP S1AFP
7730S2	M37730S2FP S2AFP S2SP S2ASP
7732S4	M37732S4FP S4AFP
7780S	M37780STJ STFP
7781M4	M37781M4TxxxJ M4TxxxFP
7781E4	M37781E4TxxxJ E4TxxxFP
7795S	M37795SJ STJ
7796E4	M37796E4-xxxJ E4TxxxJ E4TxxxFP

Table 4-1. Supported Chip by cf chip Command

ere is no SFR in your processor, you should select "**none**" to this configuration item.

Note

Executing this command will drive the emulator into the reset state.

cf iram

The **iram** (internal RAM) configuration item defines the location of internal RAM area.

Note

This item is automatically set up by **cf chip** command. Therefore, you don't have to set up this item when your processor can be specified by **cf chip** command. If your processor is not supported by **cf chip** command (when you select "other" for the configuration item), you need to set up proper value for **cf iram** command. Refer to cf chip section to determine if your processor is supported by **cf chip** command or not.

```
M> cf iram=<address>..<address>
     [,<address>..<address>]
```

You can specify the start address and end address of internal RAM area by above command. The start address and end address can be defined on 16 byte boundaries.

```
M> cf iram=none
```

When there is no internal RAM in your processor, you should select "**none**" to this configuration item.

Note

Executing this command will drive the emulator into the reset state.

cf irom

The **irom** (internal ROM) configuration item defines the location of internal ROM area.

Note 

This item is automatically set up by **cf chip** command. Therefore, you don't have to set up this item when your processor can be specified by **cf chip** command. If your processor is not supported by **cf chip** command (when you select "other" for the configuration item), you need to set up proper value for **cf irom** command. Refer to **cf chip** section to determine if your processor is supported by **cf chip** command or not.

```
M> cf irom=<address>..<address>
```

You can specify the start address and end address of internal ROM area by above command. The start address and end address can be defined on 16 byte boundaries.

```
M> cf irom=none
```

When there is no internal ROM in your processor, you should select "**none**" to this configuration item.

Note 

Executing this command will drive the emulator into the reset state.

---

cf ipmr

The **ipmr** (processor mode register) configuration item defines the location of the processor mode register.

Note 

This item is automatically set up by **cf chip** command. Therefore, you don't have to set up this item when your processor can be specified by **cf chip** command. If your processor is not supported by **cf chip** command (when you select "other" for the configuration item), you need to set up proper value for **cf ipmr** command. Refer to **cf chip** section to determine if your processor is supported by **cf chip** command or not.

```
M> cf ipmr=<address>
```

You can specify the address of processor mode register by above command.

Note 

This configuration item is needed to manage the processor mode. Note that the address is correctly specified, when you set up this configuration item by yourself.

Note 

Executing this command will drive the emulator into the reset state.

---

cf mode

The **mode** (cpu operation mode) configuration item defines operation mode of the processor and data bus width.

```
M> cf mode=<mode>
```

Valid <mode> are following:

<mode>	Description
single	The emulator will operate in single chip mode.
ext8	The emulator will operate in memory expansion mode with 8 bits external data bus width.
ext16	The emulator will operate in memory expansion mode with 16 bits external data bus width.
proc8	The emulator will operate in microprocessor mode with 8 bits external data bus width.
proc16	The emulator will operate in microprocessor mode with 16 bits external data bus width.

Note

You may need to set up a switch inside the emualtion pod in addition to this configuration. Refer to the manual provided with your emulation pod.

Note

Executing this command will drive the emulator into the reset state.

cf mon

The **mon** (monitor) configuration item allows you to choose between a foreground monitor supplied by you or the background monitor supplied with the emulator.

The emulation monitor is the program that handles communication between the emulation controller and the emulation processor. For example, when you ask for a register display, the processor is broken to the monitor, executes some code to store its register contents in an array of memory locations, then returns to executing your program.

The background monitor provided with the emulator offers the greatest degree of transparency to your target system (that is, your target system should generally be unaffected by monitor execution). However, in some cases you may require an emulation monitor tailored to the requirements of your system. In this case, you will need to use a foreground monitor linked into your program modules. See Appendix A of this manual for more information on foreground monitors.

```
M> cf mon=bg
```

You select the use of the built-in background monitor through the above command. A memory overlay is created and the background monitor is loaded into that area.

```
M> cf mon=fg..XXXX
```

You select the use of your foreground monitor using this command.

XXXX defines an 16 bits hex address where the monitor will be located. (Note: this will not load the monitor, it only specifies its location). You can define the location on a 2 kbyte boundary (address ending in 000 hex or 800 hex). You can not locate the monitor at internal RAM area or Special Function Register area.

Remember that you must assemble and link your foreground monitor starting at the 2 kbyte boundary specified in the command above. You must also load the monitor into emulation memory.

Note

If you intend to use a foreground monitor, the monitor must be loaded before attempting to load any information into target system memory.

A memory mapper term is automatically created when you execute the **cf mon=fg** command to reserve 2 kilobytes of memory space for the monitor.

The memory map is reset any time **cf mon=fg** is entered. It is only reset when **cf mon=bg** if the emulator is not already configured to use the background monitor.

Note

Executing this command will drive the emulator into the reset state.

cf clk

The **clk** (clock) option allows you to select whether the emulation processor's clock will be sourced by your target system or by the emulation pod.

```
M> cf clk=int
```

You can select the emulator's internal clock using the above command. The clock is provided from the circuit in the emulation pod. In the case of HP 64146-61001 or HP 64146-61002 emulation pod, the internal clock speed is 1 MHz. When you use an emulation pod with clock faster than 16 MHz, you need to enable the **cf rush** configuration or insert one wait state.

```
M> cf clk=ext
```

You can specify that the emulator should use the clock input to the emulator probe from the target system as the system clock. You must use a clock input conforming to the specifications for the 7700 Series microprocessor. When clock is faster than 16 MHz, you need to enable the **cf rush** configuration or insert one wait state.

Note

You can insert a wait state with one of the following methods.

- Providing the /RDY from the target system.
- Configuring the emulator to generate the /RDY signal. Refer to the section describing the **cf rdy** command.

Note

When the external clock is selected, your target system must have a clock generation circuit. The HP 64146-61001 and 64146-61002 emulation pods cannot generate clock signal using a ceramic (or quartz crystal) resonator.

Note

Executing this command will drive the emulator into the reset state.

cf int

The **int** configuration item determines whether or not the emulator responds to interrupt signals from the target system during foreground operation.

```
M> cf int=en
```

Using the above command, you can specify that the emulator will respond to interrupts from the target system.

```
M> cf int=dis
```

The emulator won't respond to interrupts from the target system.

If you are using the background monitor, the emulator does not accept any interrupt during background execution. Edge sensed interrupts occurred during in background is latched only the last one, and this interrupt will occur when context is changed to foreground. Level sensed interrupts are ignored during in background operation.

Note 

You may need to set up switches inside the pod to accept interrupts from the target system. Refer to the manual provided with your emulation pod.

Note 

Executing this command will drive the emulator into the reset state.

cf rdy

The **rdy** configuration item defines whether or not the emulator introduces /RDY input when it accesses any memory. This feature is used to run the emulator with clock faster than 16 MHz.

```
M> cf rdy=dis
```

You can disable the /RDY input by the emulator with above command. When clock is equal or slower than 16 MHz, always use the emulator with **cf rdy=dis**.

```
M> cf rdy=en
```

When enabled, the emulator activate /RDY input for one clock cycle, every time the emulator accesses memory.

Note 

Executing this command will drive the emulator into the reset state.

cf rush

The **rush** configuration item enables/disables the high speed access mode of the emulator.

```
M> cf rush=dis
```

You can disable the high speed access mode with the above command.

When you disable the high speed access mode:

- You can define up to 16 different map terms which can be placed wherever you like. (Refer to the "Memory Mapping" section in this chapter.)
- The emulator can run with no wait state, up to 16 MHz.
- The emulator can run with one wait state, up to 25 MHz.

```
M> cf rush=en
```

You can enable the high speed access mode with the above command.

When you enable the high speed access mode:

- The emulator can access emulation memory with no wait state, up to 25 MHz.
- You can map the emulation memory only to the following location.

Memory	Monitor	Available location
128K	Background	000000-01F7FF
128K	Foreground	000000-01FFFF
512K	Background	000000-07F7FF
512K	Foreground	000000-07FFFF
1M	Background	000000-0FF7FF
1M	Foreground	000000-0FFFFF
2M	Background	000000-1FF7FF
2M	Foreground	000000-1FFFFF

cf wdog

The **wdog** (watch dog timer) configuration item defines whether the watch dog timer is enabled or not.

```
M> cf wdog=dis
```

You can disable the watch dog timer with above command.

```
M> cf wdog=en
```

You can enable the watch dog timer with above command.

cf rsp

The **rsp** (reset stack pointer) configuration item allows you to specify a value to which the stack pointer and stack page register will be set upon the transition from emulation reset into the emulation monitor.

Note

This item is automatically set up by **cf chip** command. The **cf chip** command set up **rsp** to the end of internal RAM. When you select a processor which has no internal RAM, **rsp** is set to FFF hex.

```
R> cf rsp=XXXX
```

where XXXX is a 16 bits address, will set the stack pointer to that value upon entry to the emulation monitor after an emulation reset. When the emulator breaks to the background monitor, the monitor program needs 5 bytes of stack.

For example, to set the stack pointer to 27f hex, type:

```
R> cf rsp=27f
```

Now, if you break the emulator to monitor using the **b** command, the stack pointer will be modified to the value 27f hex.

Caution

Without a stack pointer, the emulator is unable to make the transition to the run state, step, or perform many other emulation functions.

cf rrt

The **rrt** (restrict to real time) option lets you configure the emulator so that commands which cause the emulator to break to monitor and return to the user program will be rejected by the emulator command interpreter.

```
M> cf rrt=en
```

You can restrict the emulator to accepting only commands which don't cause temporary breaks to the monitor by entering the above command. Only the following emulator run/stop commands will be accepted:

rst (resets emulation processor)

b (breaks processor to background monitor until you enter another command)

r (runs the emulation processor from a given location)

s (steps the processor through a piece of code -- returns to monitor after each step)

Commands which cause the emulator to break to the monitor and return, such as **reg**, **m** (for target memory display), and others will be rejected by the emulator.

Caution 

If your target system circuitry is dependent on constant execution of program code, you should set this option to **cf rrt=en**. This will help insure that target system damage doesn't occur. However, remember that you can still execute the rst, b and s commands; you should use caution in executing these commands.

```
M> cf rrt=dis
```

When you use this command, all commands, regardless of whether or not they require a break to the emulation monitor, are accepted by the emulator.

cf dmdt

This configuration item is reserved for use by the Softkey/PC Interface.

cf tdma

The tdma (trace DMA cycles) configuration item defines whether or not the emulator traces DMA cycles.

```
M> cf tdma=en
```

When you enable this item with the above command, each time DMA performed, one emulation analyzer state will be generated to recognize the DMA cycle.

```
M> cf tdma=dis
```

When disabled, no analyzer state will be generated at the occurrence of DMA. Therefore, any DMA cycle will be ignored by the analyzer.

cf trfsh

The **trfsh** (trace bus release cycles) configuration item defines whether or not the emulator traces refresh cycles.

```
M> cf trfsh=en
```

When you enable this item with the above command, refresh cycles are traced by the emulation analyzer.

```
M> cf trfsh=dis
```

When disabled, refresh cycles are not traced by the analyzer.

cf thold

The **thold** (trace hold cycles) configuration item defines whether or not the emulator traces hold cycles.

```
M> cf thold=en
```

When you enable this item with the above command, the emulation analyzer will trace hold cycles.

```
M> cf thold=dis
```

When disabled, hold cycles are not traced by the emulation analyzer.

Memory Mapping

Before you begin an emulator session, you must specify the location and type of various memory regions used by your programs and your target system (whether or not it exists). You do this for several reasons:

- the emulator must know whether a given memory location resides in emulation memory or in target system memory. This allows the emulator to properly orient buffers for the given data transfer.

- the emulator needs to know the size of any emulation memory blocks so it can properly reserve emulation memory space for those blocks.

- the emulator must know if a given space is RAM (read/write), ROM (read only), or doesn't exist. This allows the emulator to determine if certain actions taken by the emulation processor are proper for the memory type being accessed. For example, if the processor tries to write to a emulation memory location mapped as ROM, the emulator will not permit the write (even if the memory at the given location is actually RAM). (You can optionally configure the emulator to break to the monitor upon such occurrence with the bc -e rom command.) Also, if the emulation processor attempts to access a non existent location (known as "guarded"), the emulator will break to the monitor.

You use the map command to define memory ranges and types for the emulator. The HP 64146A/B 7700 Series emulator memory mapper allows you to define up to 16 different map terms; each map term has a minimum size of 256 bytes. If you specify a value less than 256 byte, the emulator will automatically allocate an entire block. You can specify one of five different memory types (erom, eram, trom, tram, grd).

For example, you might be developing a system with the following characteristics:

- input port at 500 hex
- output port at 580 hex
- program and data in external ROM from c000 through dfff hex

Suppose that the only thing that exists in your target system at this time are input and output ports and some control logic; no memory is available. You can reflect this by mapping the I/O ports to target system memory space and the rest of memory to emulation memory space. Type the following commands:

```
R> map -d *
R> map 500..580 tram
R> map 0c000..0dfff erom
R> map
```

```
# remaining number of terms   : 14
# remaining emulation memory  : 7df00h bytes
map  0000500..00005ff     tram    # term 1
map  000c000..00dfff      eram    # term 2
map  other tram
```

As you can see, the mapper rounded up the second term to 256 bytes block, since those are minimum size blocks supported by the HP 64146A/B 7700 Series emulator.

Note When you emulate the internal ROM area, you **must** map the address to emulation memory.

Note You cannot map Internal RAM and SFR as guarded (**grd**).

Note

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

Internal RAM and SFR

The emulator uses internal RAM of emulation processor to emulate user program. When you direct the emulator to display the contents of internal RAM (or SFR) area, the emulator breaks to the monitor and the monitor program reads the contents of memory. Therefore, execution of user program is suspended to perform your direction. However, you can configure the emulator so that write cycles are performed to both internal RAM (or SFR) and emulation memory. In this case, you can see the data written to emulation memory without suspending program execution.

To use this feature, you need to map these area to emulation RAM (eram). When you do this, you can display the contents of emulation memory with **m** command without suspending user program execution. You still can display the contents of internal RAM by appending "@i" to address specification in **m** command.

For example, to see the content of address 100 hex in internal RAM, you can do both of the following:

```
M> m 100   (This command accesses emulation
            memory)
M> m 100@i (This command accesses internal
            RAM of emulation processor.)
```

When you don't map the internal RAM and SFR area to emulation RAM, you can access the internal RAM and SFR without appending "@i".

Note

The contents of emulation memory is updated only when user program writes data to internal RAM or SFR. Therefore, the contents of emulation memory may be different from the actual value of internal RAM or SFR. Especially, you should pay a close attention when seeing flags of SFR.

Note

When you modify memory, the emulator breaks to the monitor, and writes data to internal RAM or SFR. Therefore, user program is suspended when modifying internal RAM or SFR.

For further information on mapping, refer to the examples in earlier chapters of this manual and to the *HP 64700 Terminal Interface User's Reference* manual.

Break Conditions

The bc command lets you configure the emulator's response to various emulation system and external events.

Write to ROM

If you want the emulator to break into the emulation monitor whenever the user program attempts to write to a memory region mapped as ROM, enter:

```
M> bc -e rom
```

You can disable this function by entering:

```
M> bc -d rom
```

When disabled, the emulator will not break to the monitor upon a write to ROM; however, it will not modify the memory location if the memory at that location is actually RAM.

Software Breakpoints

The bp command allows you to insert software traps in your code which will cause a break to the emulation monitor when encountered during program execution. If you want to enable the insertion and use of software breakpoints by the bp command, enter:

```
M> bc -e bp
```

To disable use of software breakpoints, type:

```
M> bc -d bp
```

Any breakpoints which previously existed in memory are disabled, but are not removed from the breakpoint table.

Trigger Signals

The HP 64700 emulator provides four different trigger signals which allow you to selectively start or stop measurements depending on the signal state. These are the bnct (rear panel BNC input), cmbt (CMB trigger input), trig1 and trig2 signals (provided by the analyzer).

You can configure the emulator to break to the monitor upon receipt of any of these signals. Simply type:

```
M> bc -e <signal>
```

For example, to have the emulator break to monitor upon receipt of the trig1 signal from the analyzer, type:

```
M> bc -e trig1
```

(Note: in this situation, you must also configure the analyzer to drive the trig1 signal upon finding its trigger by entering tgout trig1).

Restrictions and Considerations

Access to Internal RAM

Modifying internal RAM or SFR suspends user program execution.

Trace Internal RAM

Read data from the internal RAM or SFR is not traced correctly by the emulation analyzer.

Note

Write data is also not traced correctly, when the following conditions are met:

- The emulator is used with the M37795 emulation pod.
- The processor is operating in the memory expansion or microprocessor mode with 8 bit external bus.

DMA Support

Direct memory access to emulation memory is not allowed.

Watch Dog Timer in Background

Watch dog timer suspends count down while the emulator is running in background monitor.

Step Command with Foreground Monitor

Step command is not available when the emulator is used with foreground monitor.

Step Command and Interrupts

When an interrupt occurs while the emulator is ruuning in monitor, the emulator fails to do the first step operation. The emulator will display the mnemonic of the instruction which should be stepped, but the instruction is not actually executed. The second step operation will step the first instruction of the interrupt routine.

Emulation Commands in Stop/Wait Mode

When the 7700 microprocessor is in the stop or wait mode, emulation commands which access memory or registers will fail. You need to break the emulator into the monitor to use these commands. Once you break the emulator into the monitor, the stop or wait mode will be released.

Stack Address

In some versions of 7700 microprocessor, the stack can be located in Bank FF. However, the HP 64146A/B 7700 Series emulator doesn't support the feature. The stack must be located in Bank 0.

A

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.

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. Entry into the monitor is normally accomplished by jamming the monitor addresses onto the processor's address bus.

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

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

Foreground Monitors

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

You can tailor the foreground monitor to meet your needs, such as servicing target system interrupts. However, the foreground monitor does use part of the processor's address space, which may cause problems in some target systems. You must also properly configure the emulator to use a foreground monitor (see Chapter 4 and the examples in this appendix); and, you must link the monitor with your other program code.

An Example Using the Foreground Monitor

In the following example, we will illustrate how to link a foreground monitor with the sample program from Chapter 2. By using the emulation analyzer, we will also show how the emulator switches from state to state using a foreground monitor.

Note

The foreground monitor program provided with HP 64146A/B emulator is written for Mitsubishi RASM77 Assembler.

Modify Location Declaration Statement

Monitor Address

To use the monitor, you may need to modify the .EQU statement just after the first comment section of the monitor program listing. You should see the line below:

```
LOCATE_ADRS     .EQU    0B800H      ;start monitor on 2k boundary in bank 0
                                    ;rather than sfr/iram area
PROCMODEREG     .EQU    0005EH      ;processor mode register's address
```

You can specify the monitor location by modifying this label LOCATE_ADRS. For example, if you want locate the monitor program at a000 hex, make above line to as below:

```
LOCATE_ADRS     .EQU    0A000H      ;start monitor on 2k boundary in bank 0
                                    ;rather than sfr/iram area
PROCMODEREG     .EQU    0005EH      ;processor mode register's address
```

You can load the **fm7700b.a77** monitor on a 2k byte boundary of bank 0 (except internal RAM area and SFR area). In this example, we will locate the monitor at b800 hex. Therefore, you don't have to modify the LOCATE_ADRS label.

Processor Mode Register Address

You may need to modify the .EQU statement at the PROCMODEREG label. This value defines the location of processor mode register. If your processor has processor mode register at address other than 5e hex, modify this value to appropriate value. The following list shows the address of processor mode register.

Processor	Processor Mode Register Address	Processor	Processor Mode Register Address
7700M2	5e	7704M2	5e
7700S	5e	7704S1	5e
7700M4	5e	7720S1	5e
7700S4	5e	7730S2	5e
7702M2	5e	7732S4	d8
7702S	5e	7795S	d8
7702M4	5e	7796E4	d8
7702S4	5e		

Configure the Emulator

Before configuring the emulator, you should initialize the emulator to a known state. Type:

```
R> init -p
```

Select processor you are going to emulate. Type:

```
R> cf chip=<chip_name>
```

You need to tell the emulator that you will be using a foreground monitor and allocate the memory space for the monitor. This is all done with one configuration command. To locate the monitor on a 2k boundary starting at b800 hex, type:

```
R> cf mon=fg..0b800
```

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

```
R> map
```

```
# remaining number of terms   : 15
# remaining emulation memory  : 7f800h bytes
map  000b800..000bfff     eram   # term 1
map  other tram
```

Notice that a 2k byte block from b800 through bfff hex was mapped.

Now, you need to map memory space for the sample program. Type:

```
R> map 0c000..0dfff erom
```

If you are going to emulate a processor which has no internal RAM, map 100 hex through 2ff hex as emulation RAM.

Set a Stack Pointer

You need to set up the stack pointer for use by the foreground monitor. The foreground monitor use the stack when transit from foreground monitor to user program. You can use the **cf rsp** command to define the stack pointer location; the stack pointer will be initialized on each transition from emulation reset to the monitor. Type:

```
R> cf rsp=27f
```

Load the Program Code

Now it's time to load the sample program and monitor. Assemble and link the monitor program.

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.) See Chapter 2 for information.

Load the sample program by typing:

```
R> load -ios "cat cmd_rds.hex"
```

Load the monitor program by typing:

```
R> load -ios "cat fm7700b.hex"
```

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

```
R> b
```

Now you can run the sample program.

```
M> r 0c0000
```

Limitations of Foreground Monitors

Step Command

Step command (**s** command) is not available when you are using the foreground monitor.

cim Command

cim command is not available when you are using the foreground monitor.

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**.

B

7700 Series Emulator Specific Command Syntax

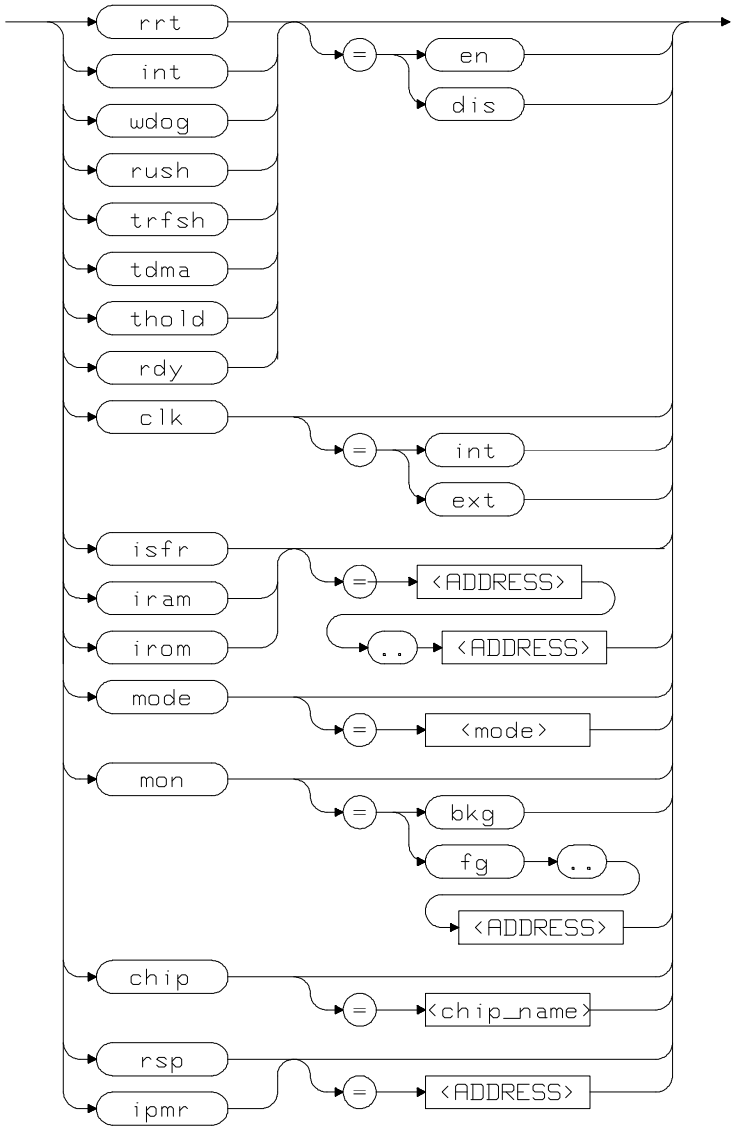
The following pages contain descriptions of command syntax specific to the 64146A/B 7700 Series emulator. The following syntax items are included (several items are part of other command syntax):

- <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), and **ser** (search memory for data) commands. The display mode is used when memory locations are displayed or modified.
- <ADDRESS>. May be specified in emulation commands which allow addresses to be entered.
- <REG_NAME>. May be specified in the **reg** (register) command.

CONFIG_ITEMS

Summary 7700 Series emulator configuration items.

Syntax



Description

The 64146A/B 7700 Series emulator has several dedicated configuration items which allow you to specify the emulator's interaction with the target system and the rest of the emulation system. These items are:

chip	Select chip to be emulated.
isfr	Define the location of Special Function Register.
iram	Define the location of internal RAM.
irom	Define the location of internal ROM.
ipmr	Define the address of processor mode register
mode	Select operation mode of the processor.
rush	Enable/disable high speed access mode.
mon	Select background or foreground monitor.
clk	Select internal/external clock source.
int	Enable/disable interrupts from target system.
rdy	Enable/disable /RDY input by the emulator.
wdog	Enable/disable the Watch Dog Timer.
rsp	Specify reset value of the stack pointer.
rrt	Restrict emulator to real time runs
tdma	Enable/disable tracing DMA cycles.
trfsh	Enable/disable tracing refresh cycles.
thold	Enable/disable tracing hold cycles.

Complete explanations of all configuration items are given in chapter 4 of this manual.

Examples

To select an external clock, type:

```
M> cf clk=ext
```

You can obtain the status of configuration items by typing the item name without a value. You can also specify multiple configuration items on the same line. Type:

```
M> cf mon=fg..08000 rrt=dis clk
```

```
cf clk=int
```

Here, we changed to a foreground monitor located at address 8000 hex, disabled the real-time runs restriction, and ask processor clock source. Notice that items which are changed do not have status printed; you could explicitly request the new status by repeating the configuration item on the command line after the change but without a value. For example:

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

```
cf mon=fg..2000
```

Related information

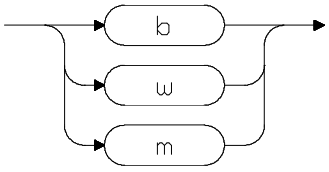
Refer to the **cf** syntax pages in the *User's Reference* manual. Also, refer to chapter 4 of this manual for complete information about each configuration item.

DISPLAY_MODE

Summary

Specify the memory display mode

Syntax



Description

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

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 operands. When memory locations are modified, the last non-mnemonic display mode specification is used. You cannot specify this display mode in the **ser** (search memory for data) command.

Defaults

The **<DIPLAY_MODE>** is **b** 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 Information

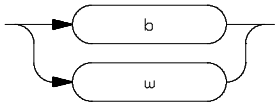
Refer to the **mo** syntax information in the Terminal Interface Reference manual for more details on mode command.

ACCESS_MODE

Summary

Specify the memory access mode

Syntax



Description

Access mode defines how the emulator accesses target system memory. The 64146A/B 7700 Series emulator allows the following access modes:

b - byte access mode

w - word access mode

The emulator monitor uses the access mode to determine whether to use byte or word instructions during target system memory accesses, such as memory modification or display. (note that it does **not** affect how that data is displayed on screen, which is controlled by the display mode.)

Defaults

The **<ACCESS_MODE>** is **b** at power up initialization. Display mode specification are saved; that is, when a command changes the access mode, the new access mode becomes the current default.

Related Information

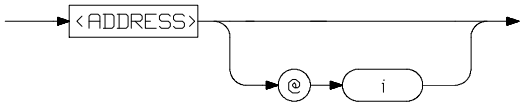
Refer to the **mo** syntax information in the Terminal Interface Reference manual for more details on mode command.

ADDRESS

Summary

Address specification used in emulation commands.

Syntax



Description

The **<ADDRESS>** parameter used in emulation commands is specified in 24 bits address information.

The **@i** specification is needed to access internal RAM or SFR when you map these area to emulation RAM. When you map these area to emulation RAM, data write cycles are performed to both internal RAM (or SFR) and emulation memory. Therefore, you can display the data written to emulation memory without suspending user program execution. To display internal RAM or SFR, you need to specify **@i** after address expression.

When you don't map internal RAM and SFR to emulation memory, you can access the actual RAM or SFR without specifying **@i** after address expression.

Examples

m 1000
m 20000..200ff
m 100=41

REGISTERS

Summary 7700 Series register designators.

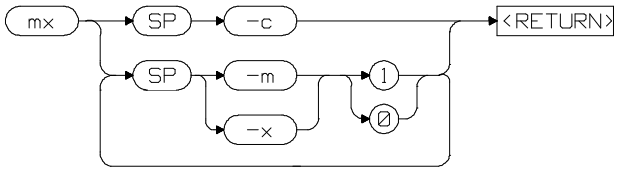
<REG_NAME> Following registers are available:

pg	Program Bank Register
pc	Program Counter
ps	Processor Status Register
dt	Data Bank Register
sp	Stack Pointer
a	Accumulator A
b	Accumulator B
x	Index Register X
y	Index Register Y
dpr	Direct Page Register

Related Commands reg (register display/modify)

mx Command

Syntax



Summary

The 7700 Series microprocessors have M flag and X flag which determine data length and index register length. The inverse assembler of emulator needs to know the value of these flags to disassemble the memory contents. The **mx** command tells the emulator the value of M flag and/or X flag.

The **-c** option can be specified to set the current value of M and X flag in processor status register.

Notes

Index