HP 64753 Emulator Terminal Interface: Z80 Emulator User's Guide

Edition1

64753-90901E1187
Printed in U.S.A. 11/87

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.

Logic Systems Division
8245 North Union Boulevard
Colorado Springs, CO 80918, 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, conversely, 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 11/87 64753-90901E1187

Using This Manual

This manual, the *Terminal Interface Z80 User's Guide,* explains how to use the Z80 Emulator with the built-in Terminal Interface firmware. It covers general use of the Z80 Emulator, including:

- Getting Started - Chapter 1
- Features Of The Z80 Emulator - Chapter 1
- How To Configure The Z80 Emulator - Chapter 1
- How To Map Z80 Emulation Memory - Chapter 1
- How To Load A Sample Z80 Program - Chapter 1
- How To Modify Z80 Memory - Chapter 2
- Use The Z80 Features - Chapter 2
- Display And Modify I/O Locations - Chapter 2
- Display And Modify Registers - Chapter 2
- Set Up, Execute, And Display A Trace - Chapter 2
- In-Circuit Emulation - Chapter 3
- Z80 Emulator Characteristics - Appendix A
- Z80 Emulator Specific Syntax - Appendix B
- Z80 Error Messages - Appendix C

The index contains terms and corresponding page numbers so that you can locate information quickly.

If you do not understand a term in this manual, refer to the *HP 64700 Emulators Glossary Of Terms* for a definition.

Use this manual in conjunction with your *HP 64700 Emulators Terminal Interface User's Reference.* That manual contains details about all of the Terminal Interface commands.

Refer To The Maps

The HP 64700 Series Manual Maps will lead you in the right direction for getting started with the various interfaces, and with using your emulator/analyzer. You can find the maps in the package marked Read Me First.

Contents

Chapter 1 Getting Started

Chapter 2 How To Use The Z80 Emulator

Chapter 3 In-Circuit Emulation

Appendix A Z80 Emulator Characteristics

Appendix B Z80 Emulator Specific Syntax

Appendix C Z80 Error Messages

Illustrations

Tables

Notes

1

Getting Started

Introduction

The topics in this chapter include:

- Purpose Of The Z80 Emulator
- Features Of The Z80 Emulator
- Before Using The Z80 Emulator
- How To Configure The Z80 Emulator
- How To Load A Sample Z80 Program

Note 

If you do not understand a term in this manual, refer to the *HP 64700 Glossary Of Terms* for a definition.

Purpose Of The Z80 Emulator

The HP 64753 Z80 Emulator is designed to replace the Z80 microprocessor in your target system so that you can control operation of the target system, or to create and debug software without a target system connected. The Z80 emulator performs just like the Z80 microprocessor, but is a device that allows you to

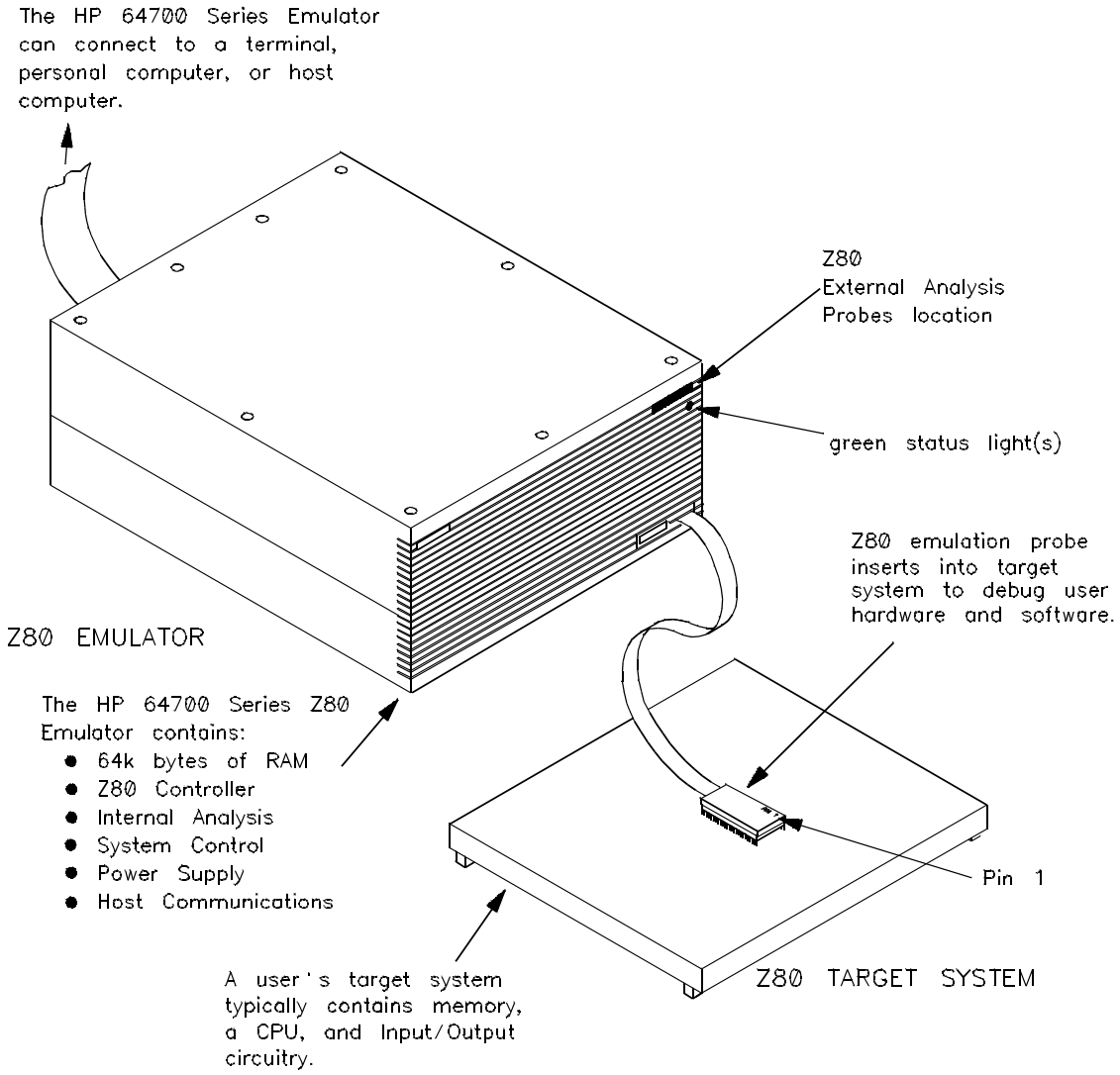


Figure 1-1. HP 64753 Z80 Emulation System

control the Z80 capabilities, such as memory and register content.

Features Of The Z80 Emulator

Supported Microprocessors: The Z80, Z80A, Z80B and Z80H and CMOS versions of the Z80 are supported.

Clock Speeds: Up to 10 MHz clock speeds are supported when using an external (target system) clock. The internal clock speed is 8 MHz.

Emulation Memory: There are 64K bytes of emulation memory that you can configure 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), and guarded (GRD) memory. The Z80 emulator will check for reads or writes to guarded memory or writes to ROM.

Analysis: The analyzer supplied with the Z80 emulator monitors the Z80 emulation processor using its own analysis bus. This analyzer performs only state analysis, and is sometimes referred to as the "emulation" analyzer. The optional "external" analyzer consists of 16 probes that you can connect to your target system. You can configure the "external" analyzer to perform state or timing analysis measurements.

Register Support: You can control the Z80 emulation processor main and alternate registers by displaying and modifying them. You can use the program counter register content to start the Z80 running.

Illegal Opcode Detection: Illegal opcodes are detected by the Z80 emulator control card. Status information sent to the analyzer contains the details.

Single-Step: You can have the Z80 emulation processor execute a single instruction, or a specified number of instructions.

Breakpoints: Hardware breakpoints, set by the analyzer, accurately reflect the state of the Z80 emulation processor at the time the break occurred. Software breakpoints are achieved by including an instruction not commonly used (LD B,B).

Reset Support: A emulator request to reset the Z80 emulator always resets the emulator to the monitor. Target system reset is accepted while running user programs, or while in the monitor, following a run from reset (**r rst**) command.

User Interface: The user interface is normally passive while running the monitor program. The interface can optionally be active (running bus cycles) while running in the monitor. You can select the value of address lines A12-A15 while the monitor is running.

Real-Time Operation: Real-time signifies continuous execution of the target system program without interference. Interference occurs when you initiate a break to the monitor, or when the break happens automatically.

Emulator features performed in real-time include: running, displaying, loading, modifying and storing emulation memory, and tracing.

Emulator features not performed in real-time include: displaying, loading, modifying, and storing target system memory, displaying or modifying registers, and single stepping.

Limitations And/Or Restrictions

DMA Support: Direct Memory Access operations to Z80 emulation memory is not permitted. However, DMA is supported to target system memory.

How The Emulation Components Communicate

The Z80 emulation components communicate with each other as shown in figure 1-2. The arrows show the direction of communication.

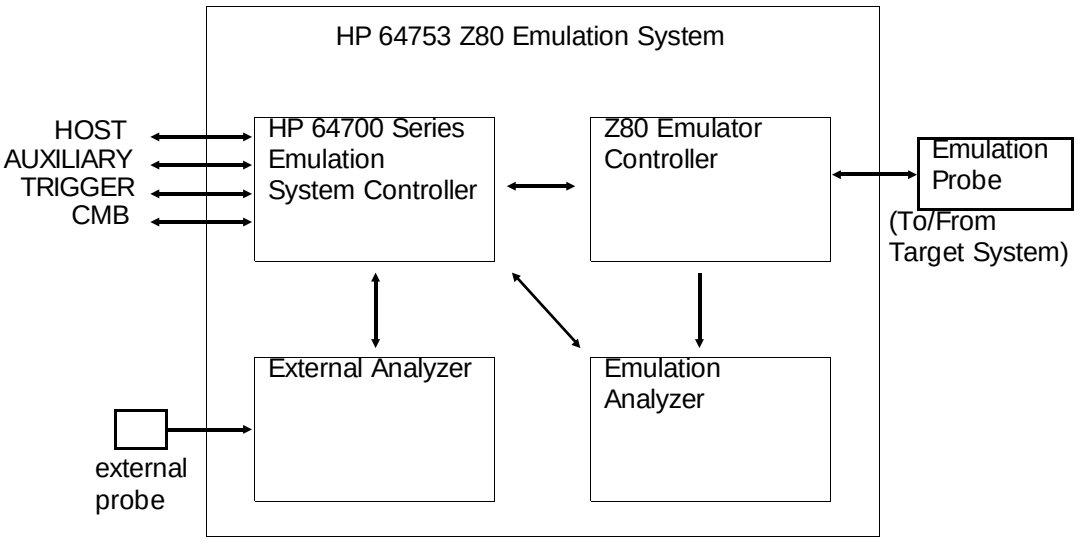


Figure 1-2. How The Emulation Components Communicate

Information On The Emulation Components

Refer to the *HP 64700 Emulators Hardware Installation And Configuration* manual for details on emulation components.

Using The Help Command To Assist You

At any time, you can type "help" or "?" to display more information about HP 64700 Series commands. The help command allows you to display information about all of the HP 64700 Series emulation and analysis commands, and about various other groups of commands.

Note  Throughout the examples in this manual, to put the commands into effect you must press **Enter** (or **Return**) after each one.

To observe what the help command provides....

TYPE: **help**

You should see:

```
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
     xtrc      - external trace analysis commands
     *         - All command groups
```

To display a comprehensive list of all of the commands....

TYPE: **help -s**

You should see:

```
mem    : cim, cov, cp, dump, io, load, m, map, ser
   run    : b, bc, bp, cf, cmb, es, r, reg, rst, rx, s
   sys    : ?, bnct, cmbt, dt, echo, equ, help, init, mac, mo, po, pv,
            rep, stty, ver, w, x, xp
   trc    : t, ta, tarm, tcf, tck, tcq, telif, tf, tg, tgout, th, tif,
            tinit, tl, tlb, tp, tpat, tpq, trng, ts, tsck, tsq, tsto, tx
   xtrc   : xt, xtarm, xtcf, xtck, xtcq, xtelif, xtf, xtg, xtgout, xth,
            xtif, xtl, xtlb, xtmo, xtp, xtpat, xtpq, xtrng, xts, xtsck,
            xtsq, xtsto, xtv, xtx
```

To display details about any of the commands....

TYPE: **help < command>**

To display information specifically about the Z80 emulator....

TYPE: **help proc**

You should see:

```
--- Address format ---
     16 bit address for memory and I/O addresses
   --- Emulation Status Characters ---
     R - emulator in reset state           c - no target system clock
     U - running user program              r - target system reset active
     M - running monitor program           h - processor halted
     W - waiting for CMB to become ready   g - bus granted
     T - waiting for target system reset   b - no bus cycles
     ? - unknown state
   --- Equates for Analyzer Label stat ---
     opcode    - first instruction byte    im0     - mode 0 acknowledge
     operand   - other instruction byte    im1     - mode 1 acknowledge
     dataread  - memory data read          im2     - mode 2 acknowledge
     datawrite - memory data write         nmi     - NMI acknowledge
     input     - input port read           busack  - bus grant acknowledge
     output    - output port write         illegal - previous byte illegal
     refresh   - memory refresh            grd     - guarded memory access
     user      - user program              wrrom   - write to rom
     monitor   - monitor program
   --- Macros ---
     outcircuit - setup emulator for operation without a target system
     incircuit  - setup emulator for operation plugged into a target system
     tbrk       - trace before break to monitor program
```

How To Use Macros

Did you notice that the last three lines of the "help proc" display define macros? The macros are set up by default for the Z80 emulator. They allow you to execute a set of commands by typing one word. Here's how it works:

To observe macros that are already defined....

TYPE: **mac**

You should see:

```
mac  incircuit={rst ; map -d * ; map other tram ; cf clk=ext }
mac  outcircuit={rst ; map -d * ; map other eram ; cf clk=int }
mac  tbrk={tcf -e; tck -ub ; tp e ; tsq -i 3 ; tif 1 stat=user ; tif 2 stat=moni-
tor ; t ; tck -u }
```

These lines define three macros for in-circuit emulation, out-of-circuit emulation, and tracing before a break. (In-circuit emulation refers to Z80 emulator operation while connected to a Z80 target system. Out-of-circuit emulation refers to Z80 emulator operation while not connected to a target system.)

The first two macros each accomplish four things:

1. Resets the emulator.
2. Deletes all current memory map terms.
3. Maps all other memory as target RAM (tram) or emulation RAM (eram).
4. Configures the clock as either external (target system) or internal (emulator).

The third macro sets up the emulation analyzer to:

1. Select the easy trace configuration.
2. Traces user and background monitor code.
3. Positions the trigger at the end of the trace list.
4. Specifies two trace sequence primary branch expressions.
5. Starts a trace.
6. After the trace is complete, returns to tracing only user programs.

Notice that commands that make up each macro are separated by a semicolon (;).

To see what the outcircuit macro accomplishes....

TYPE: **outcircuit**

You should see:

```
rst ; map -d * ; map other eram ; cf clk=int
```

Here's the response you get if you type "incircuit", and don't have a target system connected....

```
rst ; map -d * ; map other tram ; cf clk=ext
```

The prompt then changes to "c> " to indicate that no target system clock is available.

To get out of this mess....

TYPE: **outcircuit**

The system will display:

```
map -d * ; map other eram ; cf clk=int ; rst -m
```

To see what the tbrk macro accomplishes...

TYPE: **tbrk**

You will see:

```
tcf -e; tck -ub ; tp e ; tsq -i 3 ; tif 1 stat=user ; tif 2 stat=monitor ; t ; tck
-u
   Emulation trace started
```

You can halt the trace by typing **th**. Refer to the *HP 64700 Analyzer User's Guide* for details about the HP 64700 analyzer.

How To Create Your Own Macros

If you want to create your own macros, follow this syntax:

mac < macro_name> = {command 1; command 2; command 3}

By the way, you can use more than 3 commands in a macro definition. Refer to the *HP 64700 Emulators Hardware Installation And Configuration* manual for details about the **mac** command.

For example, if you want to display Z80 registers, have the Z80 emulator run, trace, then display a trace list, you could define a gmacro to do all of that for you when you type the macro name. For example (be sure to type the brackets)....

TYPE: **mac z80go= {reg;r;t;tl}**

To execute the macro....

TYPE: **z80go**

Before Using The Z80 Emulator

Before you can use the Z80 emulator, make sure you have completed these steps:

1. Understand the concept of emulation and the HP 64000-PC system architecture as presented in the *System Overview Manual*. This may help avoid problems later on.

2. You must have installed the Z80 emulator hardware in the configuration you desire (either standalone, transparent or remote mode). Refer to the *HP 64700 Hardware Emulators Installation And Configuration* manual for details on configurations.

3. If you are using remote mode, you must have installed and configured the proper program for your particular host computer. This provides the necessary data communications routines that allow the host computer to emulate the standalone or transparent command modes. Refer to the proper *Software Installation Instructions* (for your particular host computer) for details.

4. Press the **Enter** key so that the system displays one of these prompts:

U>

R>

M>

Upon powerup, the HP 64700 prompt should be "R> ". If one of these prompts does not appear, perform steps 2 and 3 again while referring to the manuals mentioned. Then try pressing the Enter key again. If one of these prompts do not then appear, refer to the *HP 64700 Emulators Support Services* manual for details on basic troubleshooting procedures. This manual also contains a list of the HP Sales And Service Offices.

Note 

One of these prompts must appear on your screen before you can continue.

About Operating The Z80 Emulator In A Target System

If you want to operate the Z80 emulator in-circuit (in a target system), see chapter 3 for details about installing the Z80 emulator probe and an example of how to use the emulator in a target system.

Familiarize Yourself With The System Prompts

The following steps are included to familiarize you with the system prompts that appear while operating the Z80 emulator.

1. TYPE: **rst**

You will see:

R>

The **rst** (reset) command resets the emulation processor and keeps it in the reset state. The "R> " prompt indicates the processor is reset.

2. TYPE: **r**

You will see:

U>

The **r** (run) command causes the emulation processor to run from the current program counter address. The "U> " prompt indicates that the processor is running a user program (rather than running in the monitor).

3. TYPE: **b**

You will see:

M>

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

4. The emulation processor may encounter parts of a program that cause it to halt. When this occurs, the system prompt will change to "h> ". An interrupt or reset is required to exit the halt state.

Initialize The Z80 Emulator

For this tutorial, it is important that you initialize the Z80 emulator to a known state.

Before you initialize the Z80 emulator:

1. Verify that no one is using the emulator.
2. Verify that no one needs the current emulator configuration.

Note 

It is important that you accomplish steps 1 and 2 if you are operating the Z80 emulator in standalone mode controlled only by a data terminal. In standalone mode, the only way to put a program into memory is by modifying the memory locations with the proper program data.

Initialize the emulator by typing: **init**

The emulation system will respond that the emulator has been initialized.

How To Configure The Z80 Emulator

You will use the **cf** command to configure the Z80 emulator. Configuring the emulator requires that you modify existing Z80 configuration choices to meet your needs. Here is how it works:

TYPE: **cf** and press **Enter**

You should see:

```
cf clk=int
cf rrt=dis
cf qbrk=en
cf trfsh=dis
cf tbusack=dis
cf busreq=en
cf int=en
cf nmi=en
cf waitem=en
cf wrdata=dis
cf moncyc=dis
cf monbase=0
```

For now, let's change just one of the values. Let's suppose that you want to use the target system clock rather than the emulator clock.

TYPE: **cf clk= ext**

NOW TYPE: **cf**

Notice that the first entry has changed to "clk= ext". It's this easy! For the rest of this tutorial we will use the clock that is internal to the Z80 emulator, so let's change it back.

TYPE: **cf clk= int**

Some Background On The Z80 Configuration Items

The following several pages contain Z80 configuration details that will help you become familiar with the configuration items.

The Z80 emulator parameters you can configure include:

- CLOCK (clk)
- RESTRICT TO REAL-TIME RUNS (rrt)
- BREAK CONDITIONS (qbrk)
- TRACING ON REFRESH CYCLES (trfsh)

- TRACING ON BUS ACKNOWLEDGE CYCLES (tbusack)
- BUS REQUEST (busreq)
- INTERRUPT (int)
- NON-MASKABLE INTERRUPT (nmi)
- TARGET SYSTEM WAIT (waitem)
- DATA WRITE TO TARGET SYSTEM (wrdata)
- Z80 MONITOR CYCLES (moncyc)
- Z80 MONITOR LOCATION (monbase)

Other Z80 Configuration Items

There are other configuration items that you can change, but are not part of the **cf** command. These include:

- Memory Map
- Access And Display Modes
- Break Conditions
- Software Breakpoints
- Coordinated Measurement Bus Operation

These configuration items are covered later in this chapter.

Getting Help On Z80 Emulator Configuration Items

You can get help on the Z80 configuration items by typing **help cf**. To get help on a specific configuration item, such as the clock, type **help cf clk**. If you have any problems, refer to your *HP 64700 Emulators Terminal Interface User's Reference* manual.

Using "cf" To Configure The Emulator

cf is the emulation configuration command. You can use it in several ways to modify the Z80 emulator configuration.

1. You can display the entire Z80 emulator configuration by typing: **cf**

2. You can display just one configuration item by typing: **cf < item>**

3. You can equate a configuration item to a value by typing: **cf < item> = < value>**

Clock (clk)

The emulator can either run from the internal 8 MHz clock or the CLK input from a target system. When using the internal clock of the emulator, your program execution is relative to the internal clock speed. Make sure that you select the internal clock when running the emulator out-of-circuit (not connected to a target system).

For example:

cf clk= int - selects internal clock in the emulator

cf clk= ext - selects CLK input in the target system

Restrict To Real-Time Runs (rrt)

When "rrt" is enabled while the emulator is running a user program, only these four commands that require a break to the monitor will be accepted: **rst, b, r,** and **s.** The emulator will reject all other commands that require a break to the monitor.

When "rrt" is disabled, all commands that require a break to the monitor will always be accepted.

For example:

cf rrt= en - restricts the Z80 emulator to real-time runs

cf rrt= dis - disables the emulator real-time run restriction

Note

Modifying the memory map or changing the emulation clock will place the emulator in the reset state, even if you have specified **rrt= en**.

Quick Break (qbrk)

You can "quickly" break to the monitor, and return to the user program, for operations such as displaying registers.

With this option enabled, when the emulator temporarily breaks to the monitor, the time it spends there will be very brief. The amount of time spent depends on the processor clock speed; at 8 MHz, the time spent in the monitor during a quick break to display registers, is about 200 microseconds. If CMB operation is enabled, any other emulators on the CMB will not break to the monitor when this emulator does a temporary break.

With this option disabled, when the emulator temporarily breaks to the monitor, the time it spends there will be lengthened. A display of registers will take about 6 milliseconds. If CMB operation is enabled, any other emulators on the CMB will break to the monitor.

A temporary break occurs if while running user code a **reg** or **io** command, or one that accesses target memory is received.

For example:

cf qbrk= en - enables the emulator to quickly enter and exit the monitor

cf qbrk= dis - disables the emulator's quick access time to the monitor

Trace Refresh Cycles (trfsh)

When enabled, the emulation analyzer will receive refresh cycles.

For example:

cf trfsh= en - enable the analyzer to trace emulator refresh cycles

cf trfsh= dis - disables the analyzer from tracing emulator refresh cycles

Trace Bus Acknowledge Cycles (tbusack)

When enabled, the emulation analyzer will receive one state for each bus request or bus acknowledge cycle.

cf tbusack= en - enable the analyzer to trace bus acknowledge cycles

cf tbusack= dis - disables the analyzer from tracing bus acknowledge cycles

Bus Request (busreq)

The Z80 emulator can either respond to or ignore the /BUSREQ input from a target system. During these times the target system requests to put valid data onto the data bus. This is in effect both while running the monitor program and while running a user program.

For example:

cf busreq= en - enables /BUSREQ input

cf busreq= dis - disables /BUSREQ input

Interrupt (int)

The emulator can either respond to or ignore the /INT input from the target system while running the user program. This signal is generated by the target system each time it seeks to interrupt the Z80 emulation processor. While running the monitor program the /INT input is **always disabled**.

For example:

cf int= en - enables emulator's response to the /INT input signal

cf int= dis - disables emulator's response to the /INT input signal

Non-Maskable Interrupt (nmi)

The emulator can either respond to or ignore the /NMI input from the target system. If enabled, a /NMI input received while the emulation processor is running in the monitor will be acknowledged when the emulator returns to the user program.

For example:

cf nmi= en - enables emulator's response to the /NMI input signal

cf nmi= dis - disables emulator's response to the /NMI input signal

Target System Wait (waitem)

If enabled, the emulator will respond to the /WAIT input from the target system during emulation memory reads and writes. Wait states will be inserted if requested. Wait states are never requested by emulation memory. If disabled, no wait states will be inserted during emulation memory accesses at any CLK frequency even if the /WAIT input is active.

Whether enabled or disabled, the emulator will respond to the /WAIT input during accesses to the target system.

For example:

cf waitem= en - enables emulator's response to the /WAIT input signal

cf waitem= dis - disables emulator's response to the /WAIT input signal

Data Write To Target System (wrdata)

If enabled, the emulator will drive the data bus to the target system (with the value read from emulation memory) during all read cycles from emulation memory. Control signals to the target system will indicate a memory read cycle but the data bus will be driven with the value read from emulation memory. This may be needed for Z80 peripheral devices to decode the RETI

instruction if the interrupt service routine is located in emulation memory.

Note  This could cause bus contention in the target system!

If disabled, the emulator will drive the data bus only during memory write and output cycles.

For example:

cf wrdata= en - enables emulator to drive data to the target system

cf wrdata= dis - disables emulator from driving data to the target system

Z80 Monitor Cycles (moncyc)

If disabled, the emulator will appear to the target system to be completely passive while running the monitor program (no bus cycles, including refresh). If enabled, the target system will recognize that the monitor program is running. All memory addresses will remain within a 4K byte range beginning with the value specified by configuration item monbase. Memory write cycles will be inhibited.

For example:

cf moncyc= en - enables monitor cycles

cf moncyc= dis - disables monitor cycles

Z80 Monitor Location (monbase)

Select the value for the monitor location that will be driven to the target system on address lines A12-A15 during background monitor operation. If bus cycles are visible to the target system while running the monitor program (that is, the configuration

item moncyc is enabled), you can locate the monitor program at an address range where memory reads will not cause undesirable interaction with the target system.

For example:

cf monbase= 6 - sets upper 4 address lines to 0110 (binary).

About The Other Configuration Items

Other items that are not configured with the **cf** command are described in the following paragraphs.

Memory Map

The memory map allows you to define whether memory located in the target system, or memory located in the emulator will respond to specific ranges of addresses. In addition, you can define an address range as RAM or ROM. Memory reads and writes are permissible to RAM, but only reads are permissible from ROM. The emulator will break to the monitor if a write to ROM occurs, and a break on write to ROM is enabled (**bc -e rom**). You can also define an address range as guarded (grd). This is useful to detect a read or write to an address where memory is not used or is non-existent. A break to the monitor will occur when guarded memory is accessed.

You can display and modify the Z80 emulator memory map using the "map" command. Each memory map term will automatically be a multiple of 256-byte blocks. If you specify a term (the number assigned to a mapped block of memory) that contains only part of a 256-byte block, the Z80 emulator will increase the allocation to include an entire block to the partial term. You can specify a total of 16 terms.

Note 

Defining a memory map has no effect on I/O (input/output) addresses. I/O reads and writes access the target system.

For each memory mapper term, you can specify one of these valid types:

- eram (emulation RAM)
- erom (emulation ROM)
- tram (target RAM)
- trom (target ROM)
- grd (guarded memory)

Any memory addresses not covered by a map term are defined by the "other" term.

For example:

map - displays the current memory map

map 0..0ffh erom - defines range as emulation ROM

map other tram - defines other addresses as target system RAM

map -d 4 - deletes term # 4 from the current map

map -d * - deletes all currently defined terms

If you mapped all of memory to emulation RAM, the memory map would resemble:

```
# remaining number of terms   : 15
# remaining emulation memory  : 0h bytes
map 00000..0ffff eram  #term 1
map  other tram
```

Access And Display Modes

The access mode refers to the type of processor data cycles that the emulation monitor uses to access a portion of user memory.

Note

For the Z80 emulator, the access mode is always set to "bytes" because the Z80 only supports an 8-bit data bus.

The display mode setting determines the format for displaying data. You can display either bytes, words, or mnemonics.

For example:

mo - displays current Z80 mode settings

mo -db - sets display mode to bytes

mo -dw - sets display mode to words (2 bytes)

mo -dm - sets display mode to processor mnemonics

mo -ab - sets access mode to bytes

Note

When memory is displayed in word format, two memory location are displayed as a 16-bit value. The low byte represents the lower addressed memory location, and the high byte represents the higher addresssed location. This is consistent with the way in which the Z80 loads a 16-bit register from memory.

Break Conditions

If break conditions are enabled, when a specified break condition occurs the emulator will break to the monitor.

If break conditions are disabled, when a specified break conditions occurs the emulator will not break into the monitor.

Possible break conditions include:

- bp - software breakpoints
- rom - write to ROM
- bnct - BNC trigger signal

- cmbt - CMB trigger signal
- trig1 - trig1 signal
- trig2 - trig2 signal

For example:

bc - displays current break conditions

bc -e [condition] [condition]... - enables break conditions

bc -d [condition] [condition]... - disables break conditions

Breakpoints

Software breakpoints allow you to stop (break) program execution when the program has reached a certain point. You can display, set, and delete software breakpoints.

Note

Software breakpoints must be enabled (with the command **bc -e bp**) before you can set them.

For example:

bp - displays current breakpoints

bp 12 450 - sets breakpoints at addresses 12h and 450h

bp -r 450 7000 - removes breakpoints at 450h and 7000h

bp -r * - removes all breakpoints

bp -e - enables all breakpoints

bp -d - disables all breakpoints

The instruction LD B,B is used as a software breakpoint by the Z80 emulator. Whenever you set a breakpoint, you will find that the content of the breakpoint address location has been modified to 40H. If your programs contain the instruction LD B,B, program execution will stop when this instruction is executed, and breakpoints are enabled.

Coordinated Measurement Bus Operation

The CMB is a connection between multiple emulators that allows you to make synchronous measurements between the emulators. You can determine whether the Z80 emulator will participate in a coordinated measurement using the **cmb** command.

For example:

cmb - display current setting of CMB

cmb -e - enable CMB interaction

cmb -d - disable CMB interaction

The **cmb** command does not affect operation of the emulation analyzer cross-triggering.

Refer to the *CMB User's Guide* for additional details about the CMB.

That's All About Configuration Items

This should be all you need to know about the Z80 emulator configuration items to configure the Z80 for your needs. However, if you need additional details, refer to the *Terminal Interface User's Reference.*

The rest of this chapter shows you tasks you may want to perform with your Z80 emulator before using the emulator in-depth.

How To Map Z80 Emulation Memory

Before loading a program into memory, you must be sure that the Z80 emulation memory map is configured properly to receive the data. To accomplish this:

TYPE: **map**

You should see:

```
# remaining number of terms   : 16
# remaining emulation memory  : 10000h bytes
map  other eram
```

This is the default Z80 memory map.

TYPE: **map 0..5ffh eram**

NOW TYPE: **map**

You should now see:

```
# remaining number of terms   : 15
# remaining emulation memory  : fa00h bytes
map  00000..005ff  eram   # term 1
map  other eram
```

You have just defined one term for the Z80 memory map.

TYPE: **map 600..7fffh erom**

NOW TYPE: **map**

You should see:

```
# remaining number of terms   : 14
# remaining emulation memory  : 8000h bytes
map  00000..005ff  eram   # term 1
map  00600..07fff  erom   # term 2
map  other eram
```

Notice that two terms are mapped. As stated earlier, the Z80 memory mapper maps memory in 256-byte blocks. If you specify a value less than 256 bytes, the emulator will automatically allocate an entire block. Every time you add a new map specification, you will see the new entry and a term number next to it.

The memory map you have just set up will work fine for the rest of this tutorial, so let's just leave it as is.

How To Load A Sample Z80 Program

There are several ways to load a sample Z80 program. These methods are described in detail in the following paragraphs.

1. If you are using the emulator with an RS-232 interface connected to a data terminal only, you will manually modify Z80 memory locations. This is referred to as the Terminal Interface in standalone mode.

2. If you are using the emulator with a host computer (HP 9000 Series 300, for example), you can create, assemble and link the Z80 program on the host computer, and then download the absolute file to Z80 emulation memory. In this case, data is sent through the port that is connected to the host computer. This is referred to as a transparent configuration.

3. If you use the host computer as a "remote" system, you can still download the absolute file into emulation memory, but the data will be sent through the same port where the emulator is connected, rather than through the port connected to the host computer. This is referred to as a remote configuration.

Refer to the *HP 64700 Emulators Hardware Installation And Configuration* manual for details on these configurations.

Note

Even if your emulator is connected to a host computer, you don't have to use the methods in steps 2 or 3 to load the program into emulation memory. You could use step 1 if you prefer.

```
"Z80"  EXPAND
       GLB  MSG,VIDEO,COUNT
       COMN
MSG    DEFB "Z80 PROGRAM1"
VIDEO  EQU  3C00H
COUNT  EQU  12
       PROG
START  LD   HL,MSG
       LD   DE,VIDEO
       LD   BC,COUNT

LOOP   LD   A,[HL]
       LD   [DE],A
       INC  HL
       INC  DE
       DEC  BC
       LD   A,B
       OR   C
       JP   NZ,LOOP
END    JP   END
```

Figure 1-3. An Example Z80 Program

This program performs the following:

1. Defines a message to be output to memory.

2. Defines start location where the message will be displayed in memory (3C00H).

3. Sets up a counter to keep track of the length of the message.

4. Loads the HL register with the first message character.

5. Loads the DE register with the first output location to write the message character.

6. Loads count into counter register BC.

7. Temporarily loads the accumulator with a message character.

8. Stores the message character (in the accumulator) into the output location.

9. Locates the next message character to be displayed.

10. Locates the next output location to store message character.

11. Decrements counter to keep track of number of characters.

12. Loads accumulator with value in upper half of counter.

13. Checks if upper half of counter (B) or'ed with lower half of counter (C) equals 0.

14. If step 13 is not true (BC does not equal 0), there are more message characters to process.

15. If step 13 is true (BC= 0), there are no more message characters to process. Therefore, the program ends.

We'll go through the first step of loading a program into memory, so that you will learn to store your programs in memory, whether the emulator is connected to a host computer or not.

Loading A Program By Modifying Memory

Loading a program into emulation memory simply involves modifying memory locations. Follow these steps. Press **Enter** after each command.

1. Clear a range of memory by typing:

m 0..0ffh= 0

2. Type these lines:

m 0= 5A,38,30,20,50,52,4F,47,52,41,4D,31,21,00,00,11

m 10= 00,3C,01,0C,00,7E,12,23,13,0B,78,0B1,0C2,15,00,0C3

m 20= 1F,00

3. Run the program by typing:

r 0C

You must run the program from 0CH because the first 12 locations are ASCII strings.

Loading A Program In Transparent Configuration

Remember to refer to the *HP 64700 Emulators Hardware Installation And Configuration* manual for details about transparent configurations.

Transferring code using transparent mode requires that:

1. your Z80 emulator and computer are properly connected to a host computer. Refer to the *Hardware Installation And Configuration* manual for a list of valid host computers.

2. your host computer is running the HP-UX operating system.

3. you are using an HP 64000 Hosted Z80 Assembler/Linker.

After these requirements are met....

1. Clear a range of memory by typing:

m 0..0ffh= 0

2. Establish communication with your host computer using the transparent mode link provided by the system by typing: **xp -s 31**

The escape character is now set to "1". Pressing **< ESC> 1** toggles the transparent mode software capabilities for one command. The system prompt will remain on screen.

3. Enable the transparent configuration link by typing: **xp -e**

4. Press the **Enter** key to get the host system login prompt.

5. Log on to your host computer.

6. Access an editor of your choice.

7. Create the example program, shown in figure 1-3, on your host computer. This program will allow you to observe Z80 memory and register activity when you run it in the emulator..

8. Save the program to a file named tutor.S.

9. Assemble the program using the HP 64000 Hosted Z80 Assembler by typing: **asm -oex tutor.S**

Correct any errors that occur. This command generates an expanded listing with a cross reference of the symbols used in your program, and their references. A relocatable object file (file with an extension of .R) is automatically created at this point. The listing should resemble:

```
FILE: ~rs/bonnieh/tutor.S  HEWLETT-PACKARD: Z80 Assembler V1.11
00001000
00002000 LOCATION OBJECT CODE LINE     SOURCE LINE
00003000
00004000                       1 "Z80"   EXPAND
00005000                       2
00006000                       3       GLB  MSG,VIDEO,COUNT
00007000                       4
00008000                       5         COMN
00009000    0000 5A38302050    6 MSG     DEFB  "Z80 PROGRAM1"
00010000    0005 524F475241
00011000    000A 4D31
00012000              3C00     7 VIDEO   EQU   3C00H
00013000              000C     8 COUNT   EQU   12
00014000                       9
00015000                      10         PROG
00016000    0000 210000       11 START   LD  HL,MSG
00017000    0003 113C00       12         LD  DE,VIDEO
00018000    0006 01000C       13         LD  BC,COUNT
00019000                      14
00020000    0009 7E           15 LOOP    LD  A,[HL]
00021000    000A 12           16         LD  [DE],A
00022000    000B 23           17         INC HL
00023000    000C 13           18         INC DE
00024000    000D 0B           19         DEC BC
00025000    000E 78           20         LD  A,B
00026000    000F B1           21         OR  C
00027000    0010 C20009       22         JP  NZ,LOOP
00028000    0013 C30016       23 END     JP  END
00029000
00030000 Errors=    0
00031000 FILE: ~rs/bonnieh/tutor.S  CROSS REFERENCE TABLE          PAGE 2
00032000
00033000 LINE#   SYMBOL        TYPE     REFERENCES
00034000
00035000    8  COUNT            A     3,  13
00036000   24  END              P    23
00037000   15  LOOP             P    22
00038000    6  MSG              C     3,  11
00039000   11  START            P
00040000    7  VIDEO            A     3,  12
00041000
```

Figure 1-4. Expanded Listing Of Assembled Z80 Program

10. Link the relocatable file by typing:

lnk and pressing **Enter**

You must provide details for the link questions. Type the terms in bold:

object files **tutor**

library files **Enter**

load addresses: PROG,DATA,COMN **0CH,0,0**

more files (y or n) **n**

absolute file name **tutor**

The Hosted Z80 Linker generates the absolute file required, and stores it in tutor.X.

Note

You can assemble and link your programs on the host computer before enabling transparent mode. Then, enable transparent mode and transfer the absolute file into the emulator. The original absolute file remains on the host computer.

---

11. Transferring the absolute file into the emulator involves three steps:

a. Disable transparent mode so that you can communicate directly with the emulator by pressing:

< ESC> **1** and then TYPE **xp -d**

"< ESC> 1" toggles transparent mode temporarily so that the emulator can accept the command you typed. "xp -d" disables transparent mode completely.

b. Initiate loading of the absolute file into the emulator by typing:

load -hbo and then press **Enter**

Caution  Do not press the Enter key after the next command! If you do, the file transfer may not work properly!

c. Initiate the file transfer by typing:

transfer -rtb tutor.X and pressing **< ESC> 1**

During the transfer process, the emulation system will display a "pound sign" for each record transferred. You should see the following on your display:

# #

m

What occurred? The command "load -hbo" instructed the emulator to initiate loading of the absolute file in HP binary file format, and to expect the data from the port connected to the host computer. You then initiated the HP 64000 transfer utility and transferred the absolute file using "transfer -rtb tutor.X. "< ESC> 1" caused the system to return to the emulator after the file transfer is complete.

12. Observe a portion of memory to verify that the absolute file was loaded into the emulator correctly by typing:

m 0..21h

Your display should match:

```
00000..0000f  5a 38 30 20 50 52 4F 47 52 41 4D 31 21 00 00 11
00010..0001f  00 3C 01 0C 00 7E 12 23 13 0B 78 B1 C2 15 00 C3
00020..00021  1F 00
```

Figure 1-5. Content Of Z80 Memory After File Transfer

Run The Example Program

If you have loaded the example program, take some time now to run or step through the program. While you step through the program, you can observe several actions taking place:

1. Display memory locations 3C00H through 3C0CH after each single step (beginning at 0CH), to verify that the message "Z80 PROGRAM1" is being written to those locations.

2. Display registers after each step, to watch the contents change as the program executes.

3. Watch the counter (register BC) decrement after each message character is written to its memory location.

4. Use the HP 64700 Analyzer to trace program execution.

Remember To Use The Manuals

If you run into a problem, other manuals are available to help you. The *HP 64700 Emulators Terminal Interface User's Reference* contains in-depth information about the Terminal Interface commands that you use to control your Z80 Emulator/Analyzer. The *HP 64700 Emulators Terminal Interface Analyzer User's Guide* contains information about using the HP 64700 Analyzer commands. If all else fails, refer to the *HP 64700 Emulators Support Services* manual.

Notes

2

How To Use The Z80 Emulator

What Is In This Chapter?

- Modify And Display Z80 Memory
- Use The Z80 Run/Stop Features
- Display And Modify I/O Locations
- Display And Modify Registers
- Set Up, Execute, And Display A Trace
- Defining Logical Expressions
- Determining How Much Of Memory Is Accessed
- Making Coordinated Measurements

Note 

After entering each command, press **Enter** (or **Return**) to put the command into effect.

Modify Z80 Memory

Before running a program, you must enter that program into emulation memory. This is the process of modifying memory lo-

cations. Just to learn how to use the Z80 emulator, let's use this simple program as an example:

```
        LD   A,4
        LD   B,8
LOOP    INC  A
        DJNZ LOOP
END     JP   END
```

To enter this program into memory beginning at location 0....

TYPE: **m 0= 3e,04,06,08,3c,10,0fd,0c3,07,00**

Observe that memory was modified, and that your program was entered as you expected....

TYPE: **m -dm 0..8**

This will show a listing of your program as it appears in emulation memory. The -dm option causes the content of memory locations you specify to be displayed in mnemonic form. The result is:

```
00000  3e04        LD A,04
00002  0608        LD B,08
00004  3c          INC A
00005  10fc        DJNZ 0004
00007  c30000      JP 0007
```

Use The Z80
Run/Stop Features

Now to run this program....

TYPE: **r 0**

The prompt will change from M> to U> to indicate that a "User" program is running.

You can also run the program by stepping one instruction at a time. Here's how:

TYPE: **s 1 0;reg**

This command steps 1 instruction from address 0, then displays the contents of the Z80 registers.

The result is:

```
00000  3e04        LD A,04
PC = 00002
reg a=04 f=08 bc=0000 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0002
```

Notice that the a registser contains 4. The prompt will change from U> to M> to indicate that the emulator is running in the monitor.

Continue stepping through the program and observe the results after each step.

Note 

You can recall the last command(s) by pressing ^ **r** (pressing the **CTRL** key, then pressing **r**). You can press ^ **r** instead of typing **s 1;reg**, to step through the rest of the program.

Type the commands in bold (or press ^ r), and watch the screen as the command and register results are displayed.

s 1;reg

```
00002  0608        LD B,08
PC = 00004
reg a=04 f=08 bc=0800 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0004
```

s 1;reg

```
00004  3c          INC A
PC = 00005
reg a=05 f=00 bc=0800 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0005
```

s 1;reg

```
00005  10fd        DJNZ 0004
PC = 00004
reg a=05 f=00 bc=0700 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0004
```

s 1;reg

```
00004  3c          INC A
PC = 00005
reg a=06 f=00 bc=0700 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0005
```

s 1;reg

```
00005  10fd        DJNZ 0004
PC = 00004
reg a=06 f=00 bc=0600 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0004
```

s 1;reg

```
00004  3c          INC A
PC = 00005
reg a=07 f=00 bc=0600 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0005
```

s 1;reg

```
00005  10fd        DJNZ 0004
PC = 00004
reg a=07 f=00 bc=0500 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0004
```

s 1;reg

```
00004  3c          INC A
PC = 00005
reg a=08 f=08 bc=0500 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0005
```

s 1;reg

```
00005  10fd        DJNZ 0004
PC = 00004
reg a=08 f=08 bc=0400 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0004
```

2-4 How To Use The Z80 Emulator

s 1;reg

```
00004  3c          INC A
PC = 00005
reg a=09 f=08 bc=0400 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0005
```

s 1;reg

```
00005  10fd        DJNZ 0004
PC = 00004
reg a=09 f=08 bc=0300 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0004
```

s 1;reg

```
00004  3c          INC A
PC = 00005
reg a=0a f=08 bc=0300 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0005
```

s 1;reg

```
00005  10fd        DJNZ 0004
PC = 00004
reg a=0a f=08 bc=0200 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0004
```

s 1;reg

```
00004  3c          INC A
PC = 00005
reg a=0b f=08 bc=0200 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0005
```

s 1;reg

```
00005  10fd        DJNZ 0004
PC = 00004
reg a=0b f=08 bc=0100 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0004
```

s 1;reg

```
00004  3c          INC A
PC = 00005
reg a=0c f=08 bc=0100 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0005
```

s 1;reg

```
00005  10fd        DJNZ 0004
PC = 00007
reg a=0c f=08 bc=0000 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0007
```

s 1;reg

```
00007  c30700      JP 0007
PC = 00007
reg a=0c f=08 bc=0000 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0007
```

To stop execution of the Z80, and put it in the reset state....

TYPE: **rst**

The Z80 emulator will remain in the reset state until you cause it to release from that state by entering the monitor, or running a program. To run the emulator from reset, just type **r**. To break the emulator into the monitor, just type **b**.

To put the Z80 emulator in reset, then have it begin executing in the monitor automatically....

TYPE: **rst -m**

The Z80 will reset, then begin executing in the monitor.

Display And Modify I/O Locations

You can observe data at the emulation processor I/O ports. You can also modify the data at those ports.

Note 

To transfer valid data to or from I/O ports, you must have the emulator connected to the target system. Also, if the emulator is in the reset state, you cannot modify the data at the I/O ports.

The data at your processor I/O locations may be different than those shown in the examples.

To display I/O data at address 400h in byte format....

TYPE: **io -db 400**

You will see:

```
00400   ff
```

Note

A 16-bit address may be used as an I/O port address.

To display I/O data in at address 800h in word format....

TYPE: **io -dw 800**

You will see:

```
00800   00ff
```

Note 

Specifying I/O word format has no meaning to the Z80, but is not flagged as an error. Z80 ports are strictly 8 bits wide. The two zeros preceding the address indicate you have specified word format.

To modify I/O data at location 400h to 20....

TYPE: **io 400h= 20**

To display multiple locations, and modify one location in a single command....

TYPE: **io 100h 101h 102h= 20**

You will see:

```
00100   00ff
00101   00ff
```

When you modify an I/O location, the result is not automatically displayed.

Display And Modify Z80 Registers

You can observe the Z80 emulator registers to verify exactly what is going on with your program while it is executing. You can also change the value of a register to see what effect it has on your program.

To display the contents of the primary Z80 registers....

TYPE: **reg**

The result is:

```
reg a=3a f=02 bc=0100 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=0acb
```

To display a single Z80 register by specifying its name....

TYPE: **reg a**

Only the content of the Z80 "a" register will be displayed.

The result is:

```
reg a=3a
```

To display the a register, modify the bc register, and display the set of alternate Z80 registers in a single command....

TYPE: **reg a bc= 0ffffh alt**

The result is:

```
reg a=3a
reg a'=00 f'=00 bc'=0000 de'=0000 hl'=0000
```

Note 

When you modify a register, the result is not automatically displayed.

To display just the set of alternate Z80 registers....

TYPE: **reg alt**

The result is:

```
reg a'=00 f'=00 bc'=0000 de'=0000 hl'=0000
```

To display just the set of interrupt Z80 registers....

TYPE: **reg int**

The result is:

```
reg i=00 iff2=00 imode=00
```

To display all of the Z80 registers....

TYPE: **reg all**

The result is:

```
reg a=3a f=02 bc=0100 de=0000 hl=0000 ix=0000 iy=0000 sp=0000 pc=7663 r=7d
reg a'=00 f'=00 bc'=0000 de'=0000 hl'=0000 i=00 iff2=00 imode=00
```

To modify the Z80 bc register to 1234....

TYPE: **reg bc= 1234**

NOW TYPE: **reg bc**

The result is:

```
reg bc=1234
```

To modify multiple Z80 registers in a single command....

TYPE: **reg a= 0 bc= 1234 hl= 5678 imode= 1**

To observe the result....

TYPE: **reg all**

The result is:

```
reg a=00 f=02 bc=1234 de=5678 hl=0000 ix=0000 iy=0000 sp=0000 pc=81b3 r=5b
reg a'=00 f'=00 bc'=0000 de'=0000 hl'=0000 i=00 iff2=00 imode=01
```

Set Up, Execute, And Display A Trace

When setting up a trace, you should first initialize the analyzer. Doing this ensures that the analyzer is starting from the default state.

To initialize the analyzer to the default state....

TYPE: **tinit**

The emulator will not display any messages.

Execute A Trace

If you want to trace program execution from the beginning of the program, start the trace before the program. The emulator can be running a user program when you start the trace.

Start the analyzer tracing.

TYPE: **t**

The emulator will display the message: "Emulation trace started". If you have modified memory with the short example program shown earlier, start that program running.

TYPE: **r 0**

If the prompt did not resemble "U> " before, it should now.

Display The Trace

To display the trace you just made of your program execution....

TYPE: **tl 0..30**

The result is:

```
Line   addr,H  Z80 Mnemonic,H                  xbits,H    count,R  seq
-----  ------  ------------------------------  -------  ---------  ---
    0   0000    LD A,04                         0000          ---   +
    1   0001        04  operand                 0000      0.440 uS   .
    2   0002    LD B,08                         0000      0.240 uS   .
    3   0003        08  operand                 0000      0.440 uS   .
    4   0004    INC A                           0000      0.280 uS   .
    5   0005    DJNZ 0004                       0000      0.400 uS   .
    6   0006        FD  operand                 0000      0.520 uS   .
    7   0004    INC A                           0000      0.760 uS   .
    8   0005    DJNZ 0004                       0000      0.400 uS   .
    9   0006        FD  operand                 0000      0.560 uS   .
   10   0004    INC A                           0000      0.760 uS   .
   11   0005    DJNZ 0004                       0000      0.400 uS   .
   12   0006        FD  operand                 0000      0.520 uS   .
   13   0004    INC A                           0000      0.760 uS   .
   14   0005    DJNZ 0004                       0000      0.400 uS   .
   15   0006        FD  operand                 0000      0.560 uS   .
   16   0004    INC A                           0000      0.760 uS   .
   17   0005    DJNZ 0004                       0000      0.400 uS   .
   18   0006        FD  operand                 0000      0.520 uS   .
   19   0004    INC A                           0000      0.760 uS   .
   20   0005    DJNZ 0004                       0000      0.400 uS   .
   21   0006        FD  operand                 0000      0.560 uS   .
   22   0004    INC A                           0000      0.760 uS   .
   23   0005    DJNZ 0004                       0000      0.400 uS   .
   24   0006        FD  operand                 0000      0.520 uS   .
   25   0004    INC A                           0000      0.760 uS   .
   26   0005    DJNZ 0004                       0000      0.400 uS   .
   27   0006        FD  operand                 0000      0.560 uS   .
   28   0007    JP 0007                         0000      0.240 uS   .
   29   0008        07  operand                 0000      0.440 uS   .
   30   0009        00  operand                 0000      0.320 uS   .
```

You could have just as well requested a default trace list by just typing **tl**. Typing **tl** multiple times would then allow you to see the entire trace in multiple viewings.

Stepping Through A Trace List

You can step the same program and trace after each instruction execution if you like. To do this, for example....

TYPE: **t**

The emulation trace is started.

TYPE: **s 1 0**

The result is:

```
00000  3e04        LD A,04
   PC = 00002
```

TYPE: **tl**

The result is:

```
Line   addr,H  Z80 Mnemonic,H                  xbits,H    count,R  seq
   -----   ------  ------------------------------  -------  ---------  ---
       0   0000    LD A,04                         0000          ---   +
       1   0001        04  operand                 0000      0.440 uS   .
       2
```

TYPE: **s 1**

The result is:

```
00002  0608        LD B,08
PC = 00004
```

TYPE: **tl**

The result is:

```
Line   addr,H  Z80 Mnemonic,H                  xbits,H    count,R  seq
   -----   ------  ------------------------------  -------  ---------  ---
       3   0003        08  operand                 0000      0.440 uS   .
       4
```

TYPE: **s 1**

The result is:

```
00004  3c          INC A
   PC = 00005
```

TYPE: **tl**

The result is:

```
Line   addr,H  Z80 Mnemonic,H                  xbits,H    count,R  seq
   -----   ------  ------------------------------  -------  ---------  ---
       3   0003        08  operand                 0000      0.440 uS   .
       4
```

Continue stepping and tracing if you desire.

Defining Logical Expressions

Equates are logical expressions. The **equ** command allows you to equate logical expressions to names that you choose.

You can add, observe, delete and modify logical expressions using the **equ** command. You can also put multiple equate commands on a single line. This is the syntax.

equ name= < value> - equate a name to a number or pattern

equ name- display defined equate

equ -d name- delete defined equate

equ -d *- delete all defined equates

equ *- list all defined equates

equ- list all defined equates

equ name1= < value> name2- define and modify equates

For the Z80 emulator, these equates are defined upon powerup:

```
equ busack=0xxxx1111y
equ dataread=0xxxx0100y
equ datawrite=0xxxx0101y
equ grd=11xx0x0xy
equ illegal=0xxxx0010y
equ im0=0xxxx1000y
equ im1=0xxxx1001y
equ im2=0xxxx1010y
equ input=0xxxx0110y
equ monitor=0xxxxxxxy
equ nmi=0xxxx1011y
equ opcode=0xxxx0000y
equ operand=0xxxx0001y
equ output=0xxxx0111y
equ refresh=0xxxx1100y
equ user=1xxxxxxxy
equ wrrom=1x1x0101y
```

These logical expressions are used to define values for the emulation analyzer label "stat".

You can define other types of logical expressions using the **equ** command.

For example:

To define a logical expression for running from the start of a program at address 100h....

TYPE: **equ start= 100h**

To make sure the start equate was added to the list of equates....

TYPE: **equ**

To start the program....

TYPE: **r start**

To use the start equate in defining a trigger pattern for the emulation analyzer....

TYPE: **tpat p1 addr= start**

This defines pattern 1 to be the address equal to the value of start.

To delete the start equate....

TYPE: **equ -d start**

To make sure the start equate was deleted to the list of equates...

TYPE: **equ**

For additional details on equates refer to the *Terminal Interface User's Reference.*

Note 

For additional details on using the analyzer, refer to the *HP 64700 Analyzer User's Guide.*

Determining How Much Of Memory Is Accessed

You can determine how much of a range of emulation memory is accessed by your program using the coverage (**cov**) command. Here is how it works.

First, clear the coverage memory....

TYPE: **cov -r**

To determine how much memory the simple loop program in locations 0 through 7 uses....

TYPE: **cov 0..0fh**

You will see:

```
percentage of memory accessed:  %  0.0
```

To start executing the program so that a coverage measurement can be made....

TYPE: **s 1 0**

You will see:

```
0000  3e04        LD A,04
   PC = 0002
```

TYPE: **cov 0..0fh**

You will see:

```
percentage of memory accessed:  % 18.7
```

Run the rest of the program....

TYPE: **r**

To see what percentage of memory is used now....

TYPE: **cov 0..0fh**

```
percentage of memory accessed:  % 62.5
```

For additional details on making coverage measurements, refer to the *HP 64700 Emulators Terminal Interface User's Reference.*

3

In-Circuit Emulation

What Is In This Chapter?

- How To Install The Z80 Emulator Probe
- Perform Z80 Emulation Functions In-Circuit
- In-Circuit Emulation Specifics
- Modifying Operation Of The Monitor Program
- High Speed CMOS Target System Interface

If you have never used an emulator in-circuit before, this chapter should be a great benefit to you. If you have used an emulator in-circuit, hopefully this chapter will still be of some help.

What Is In-Circuit Emulation?

In-circuit emulation is the process of connecting the emulator probe into the microprocessor socket of your target system, so that you can use the emulator features to create and debug your target system hardware and software.

By operating your emulator in-circuit, you can manipulate memory, I/O ports, and registers on the target system, allowing your development process to become easier and faster. When modify-

ing registers while running the emulator in-circuit, you are modifying the emulation processor's registers, because the HP 64753 emulator replaces the target system microprocessor.

You can map all of memory to the target system, allowing you to do all of your work in target system memory, such as copying programs from target system ROM to target system RAM to make changes and debug. You also can copy programs in target system ROM or RAM to emulation RAM so that programs may be analyzed or altered.

Default Configuration Item Definitions

The default emulator configuration items are defined for target system use, as most in-circuit emulation customers would typically need. If you find that some of the configuration items are not defined as required by your target system, you can redefine any of them to suit your particular target system needs.

Typing **cf** allows you to see all of these default configuration items.

```
clk=int  (select emulation or target system clock)
rrt=dis  (emulator not restricted to real-time runs)
qbrk=dis  (quickly break the emulator)
trfsh=dis  (trace on refresh cycles)
tbusack=dis  (trace on bus acknowledge cycles)
busreq=en  (enable or disable bus request)
int=en  (enable or disable interrupts)
nmi=en  (enable or disable non-maskable interrupt)
waitem=en  (enable or disable target system waits)
wrdata=dis  (enable or disable data writes to target system)
moncyc dis  (enable or disable target system recognition of
                emulator monitor cycles)
monbase=0  (location of the monitor program)
```

See chapter 1 for details about these configuration items.

How To Install The Z80 Emulator Probe

To install the emulator probe....

1. Power down the target system and emulator.

2. Remove any tape, foam, or plastic from the end of the emulator probe.

3. Examine all pins on the protective socket connected to the emulation probe to verify that they are straight.

4. If any of the pins on the probe are bent, carefully straighten them. If a pin breaks on the socket, replace the socket.

5. Locate pin 1 on the emulator probe. It is at the tip of the probe.

6. Locate pin 1 on your target system microprocessor socket.

7. Carefully match the pins on the emulator probe with the target system microprocessor socket.

8. Depending on what type of socket is in your target system, either press down on the emulator probe, or push the lever on the socket to make a stable connection between the emulator probe and target system Z80 microprocessor socket.

9. Power up the emulator.

10. Power up the target system.

Now the emulator is ready to operate in your target system.

Continue on to get started.

Perform Z80 Emulation Functions In-Circuit

Look at the emulator configuration by typing: **cf**

Note

You can change any of the configuration items. Only you can decide if the default configuration items are appropriate for your target system operation. For more information on the Z80 emulator configuration, see chapter 1.

If you want to delete all current memory map terms, map all other memory as target system RAM, and use the target system (external) clock, execute the in-circuit emulation macro....

TYPE: **incircuit**

You should see:

```
rst ; map -d * ; map other tram ; cf clk=ext
```

An Example Target System

Now, let's imagine a target system that contains a Z80 microprocessor, ROM (locations 0 to 7ffh), RAM (locations 800h to 0fffh), 8 input switches, 8 output LEDs, a counter timer circuit, and a hard reset switch, along with other control circuitry and buffers. Also, the target system input lines are hardwired to 0 through 7fh. Furthermore, the target system program generates only mode 2 interrupts.

Let's assume that the program is designed to turn on an LED each time it successfully completes a certain step. Eventually the LEDs are supposed to light up in a defined pattern, and keep repeating the pattern.

Here are some things you could do with this target system:

Observe part of the program in target system ROM....

TYPE: **m -dm 0..2fh**

Run the program in target system ROM....

TYPE: **r 0**

The LEDs would light up in the order defined by the program.

Stop target system execution, and cause the emulator to begin executing in the monitor....

TYPE: **b**

The prompt changes to "M> " to indicate that the emulator is executing in the monitor. Whenever the emulator breaks to the monitor, the target system stops running.

Start the program running from target system reset....

TYPE: **r rst**

If the target system contains a hard reset button, pressing it would now start the program running from reset. The target system will accept a "run from reset" command (**r rst**) while the emulator is running a user program, running in the monitor, or is in the reset state. After issuing a r rst command, when you press the reset button on the target system, it will continue running the target system program.

Copy the program into target system RAM so that you could make changes....

TYPE: **cp 800h= 0..065h**

This command indicates that the original program is located at locations 0 through 65 hexadecimal in target system ROM. The program will be copied to target system RAM starting at 800 hexadecimal (the starting address of target system RAM).

Step through the program that you just copied into RAM....

TYPE: **s 1 800h**

This command steps one instruction from the start of the program.

Continue stepping....

TYPE: **s 1**

When the program writes to certain locations (with an "OUT < location> , A" instruction), you would see an LED light up.

To trace activity of the program, start the program running, then....

TYPE: **t**

The message "Emulation trace started" would be displayed.

Display the trace of the program....

TYPE: **tl**

Store only trace activity of the interrupt mode 2 signal....

TYPE: **tsto stat= im2**

Check the content of the Z80 imode register....

TYPE: **reg imode**

The content would be 02 because the Z80 has been programmed to use interrupt mode 2.

Display and modify Z80 emulator I/O ports....

Note 

With the Z80 microprocessor, I/O port addresses 0 through 0ffffh are available for sending data to and receiving data from a target system. The Z80 microprocessor communicates with memory and the I/O ports individually.

To display multiple I/O locations communicating with the target system in byte format....

TYPE: **io -db 0 1 2 7fh**

You would see something like:

```
00000   ff
00001   ff
00002   ff
0007f   ff
```

If the input switches on the target system were set to 44, I/O locations 0 through 7fh would contain data of 44.

To modify I/O location 0 to 0....

TYPE: **io 0= 0**

When To Modify I/O Locations

A write to an I/O port indicates that the Z80 emulator holds valid data to be stored at the addressed I/O location. Therefore, you should send data to I/O locations when you want to cause an action in the target system.

When modifying an I/O location you need only to specify an I/O port address and supply a data value to send to the port.

The emulation processor must not be halted.

The target system must be ready to receive the data. If the /WAIT signal is active, the target system is not ready for the data to be transferred.

The emulation processor must acknowledge the target system bus request. Once it acknowledges the bus request, the target system can take control of the /RD, /WR, and /IORQ lines, and then transfer data.

In-Circuit Emulation Specifics

The HP 64753 Z80 Emulator can perform emulation in real-time or nonreal-time with a target system connected. If real-time performance of your target system is important, you should perform Z80 emulation in real-time.

Emulation may be required to run in real-time because running in nonreal-time with proper emulator operation may not always possible (such as with target systems that process interrupts and/or depend on a real-time clock for operation). Target systems like these cannot be emulated thoroughly if you do not specify real-time emulation. Therefore, you should be aware of the types of emulator commands that will cause the emulator to operate in nonreal-time.

In addition, you should understand the implications of using part or all of emulation memory for emulation. The use of emulation memory could affect real-time emulation, depending on the implementation of your target system.

Some guidelines for in-circuit emulation include:

Select the external clock during emulation configuration so that your target system program will execute relative to your target system clock.

If memory exists on your target system, map that memory as target memory (tram or trom) when you configure the emulator. This is important so that all activities you perform on memory are automatically done in the target system memory.

Emulation Memory and Target System Memory

The use of emulation memory and/or target system memory can significantly affect the operation of the emulator. Ideally emulation of a microprocessor should be done with as much of the final target system hardware as possible. Because this is not feasible at the start of the development cycle, the HP 64753 Z80 emulator provides the feature for emulation memory to replace target system memory during the project development stage.

The final target system memory may not have the same specifications as the emulator. Therefore, we recommend that you use the external clock, whenever possible, to assure synchronous operation between the emulator and your target system.

Modifying Operation Of The Monitor Program

When the emulator breaks from the user program it begins running the monitor program. The purpose of the monitor is to allow access to internal Z80 registers and to provide a path to access memory and input/output ports in the target system.

When the emulator stops running the user program, any support that is provided by the user program to the target system will also stop, including support of interrupts. This may be a problem, particularly with target systems that have real-time requirements.

Make Bus Cycles Visible

There are several ways that you can alter the operation of the monitor program to satisfy some specific requirements of your target system. The first step is to make the bus cycles that the Z80 performs while running the monitor, visible to the target system, by setting the monitor cycle configuration item.

For example, you would type: **cf moncyc= en**

Define Addresses

Your target system will require this if it has dynamic memories that must be refreshed by the Z80. You can define the range of addresses driven to the target system by setting the monitor base configuration item.

For example, you would type: **cf monbase= 7**

This command causes these addresses to be in the range 7000H through 7FFFH, while running in the monitor.

Reduce Monitor Program Access Time

A second way to modify the operation of the monitor is to reduce the amount of time that the monitor program runs when a temporary break occurs. A temporary break occurs when the emulator breaks to the monitor to display registers or to display target memory. An automatic return to running the user program follows the break. By setting the quick break configuration item (by typing **cf qbrk= en**) the amount of time spent in the monitor for a temporary break can be reduced from about 6 milliseconds to about 200 microseconds, depending upon processor clock speed.

Even though interrupts are not serviced while the emulator is running in the monitor, it may be possible to return the emulator to the user program in time to service interrupts satisfactorily.

Note 

This mode of breaking affects CMB operation because other emulators on the CMB will not break to the monitor when this emulator breaks in the quick mode.

Tailoring The Monitor For Target System Interaction

If the target system requires constant interaction, even while the emulator is running the monitor, the monitor program itself can be tailored by you. One example of a target system need of this type is a "watchdog timer" that will reset the target system if it is not written to periodically. In this case, you could modify the monitor program to write to the timer while it is running, in the same way that the user program writes to the timer. Another possibility is to turn off the timer when a break to the monitor occurs, and turn it on again when the emulator returns to the user program. Both of these requirements can be satisfied by adding user code to the monitor program.

Using The Load -g Command

The user code is supplied by you in the form of four possible Z80 assembly language subroutines. You then load this code by using the **-g** option with the **load** command. Refer to the *HP 64700 Emulators Terminal Interface User's Reference* for details about the **load** command.

Subroutine # 1: Reset Entry To Monitor

The first subroutine will be called once when the monitor program is entered from reset. This will occur when a reset command (**rst**) is followed by a break (**b**), run (**r**), or step (**s**) command, or with a reset to monitor command (**rst -m**).

Subroutine # 2: Break Entry To Monitor

The second subroutine will be called once when the emulator breaks from running the user program into the monitor.

Subroutine # 3: Monitor Loop

The third subroutine will be called once for each time the monitor program cycles through its main loop.

Subroutine # 4: Exit From Monitor

The last subroutine will be called once when the emulator exits the monitor and returns to the user program.

Note

It is not necessary that you provide all four subroutines. Any subroutine that is not included in the loaded code will be defaulted to a "return from subroutine" (RET) when the load is performed.

Restrictions

Code that you add to the monitor is limited by the following restrictions:

User Code Separate From Monitor Code

Code that you add to the monitor is completely separate from your user program. The user monitor code does not reside in the user address space, nor can it call or jump to any part of your program.

Location Of The Subroutines

The four subroutines must begin at the following addresses:

0300H - reset entry to monitor subroutine

0400H - break entry to monitor subroutine

0500H - monitor loop subroutine

0600H - exit from monitor subroutine

Note

Each subroutine must end with a return from subroutine instruction RET). The entire user monitor code must reside with the address range 0300H through 06FFH. The user monitor code must not jump to or access any monitor locations outside this range.

Note 

The addresses where the subroutines must be located are not affected by the setting of the "cf monbase" configuration question.

Communication With Target System I/O ports

The user monitor code can communicate with target system memory and I/O (input/output) ports. The following instructions must be used when this communication is desired. No other instructions will access the target system.

Table 3-1. Instructions Used With A Target System

INSTRUCTION	FUNCTION OF INSTRUCTION
LD HL,(nnnn)	Loads HL with target memory locations nnnn and nnnn+ 1.
LD (nnnn),HL	Writes contents of HL to target memory locations nnnn and nnnn+ 1.
LD r,(HL)	Loads register with target memory address pointed to by HL. "r" indicates register A, B, C, D, E, H, or L.
LD (HL),r	Writes register content to target memory pointed to by HL.
IN r,(C)	Loads register with input port value pointed to by the C register.
OUT (C),r	Writes register content to I/O port pointed to by C.

You Cannot Use Some Instructions

The following instructions must not be used anywhere in the user monitor code because they are reserved for control of the Z80 emulator. **Unexpected operation of the emulator will occur if you use any of these instructions:**

JP nnnn
RST n
LD A,(BC)
LD D,D
LDIR
LDDR

Registers Used By The User Monitor Code

The user monitor code may only use the following Z80 registers: A, flags, BC, DE, and HL

The content of these registers is not maintained by the monitor from one call to the user routines to the next. Any data storage required must share monitor memory addresses in the range 0300H through 06FFH with the user monitor code. Instructions used to access data stored in monitor memory must not use the instructions for target memory communication listed above.

A Stack Is Provided

A stack is provided so that you may include subroutines in the user monitor code. The stack is used only for the purpose of calling the user monitor subroutines, and for their operation. The stack size is limited to 64 bytes. The stack is always automatically initialized before a call is made to a user monitor subroutine.

No Access To Emulation Memory

The user monitor code cannot access emulation memory.

Caution 

If the user monitor subroutines do not adhere to the restrictions listed above, erratic operation of the emulator may result.

Creating/Loading The User Monitor Subroutines

You can develop the user monitor subroutines with the same tools that are used for the user application program. Any one of the subroutines, or all four, can be included in the same file. It is important that each subroutine be ORGed at its specified address (refer to the appropriate *Assembler Manual* for details about the **ORG** command).

The subroutines are then loaded into the emulator with the **-g** option to the **load** command. For example, if the user monitor code is in HP ASCII format and being loaded from the other port (not the command port), the command would be: **load -ghxo**

Note 

The **load** command will fail if the address specified in the file is not in the range 0300H through 06FFH.

Observe Monitor Operation

After the user monitor code has been loaded, the emulation analyzer can be used to monitor its operation. You can enable tracing of background monitor cycles by changing the clock specification.

For example, you would type: **tck -ub**

Refer to the *Terminal Interface User's Reference* for details.

Note 

If the emulator is reinitialized (with the **init** command), the user monitor code will be lost, and must then be reloaded.

High-Speed CMOS Target System Interface

The HP 64753 Z80 emulator is designed for both CMOS capability, and operation at clock speeds up to 10 MHz. To meet these requirements, high speed CMOS buffer circuitry is used to drive the address, data, and status signals on the emulator cable to the target system.

These CMOS buffers generate extremely fast rise and fall times. Some Z80 target systems may exhibit erratic operation with the Z80 emulator installed, such as target systems which do not have power and ground layers on their printed circuit boards. This may occur with a prototype that is wire-wrapped.

To minimize the effects of this problem, Hewlett-Packard suggests that you add the circuit shown in figure 1. Figure 2 shows one possible implementation of the circuit.

Consult your local Hewlett-Packard Sales and Service Office for more information.

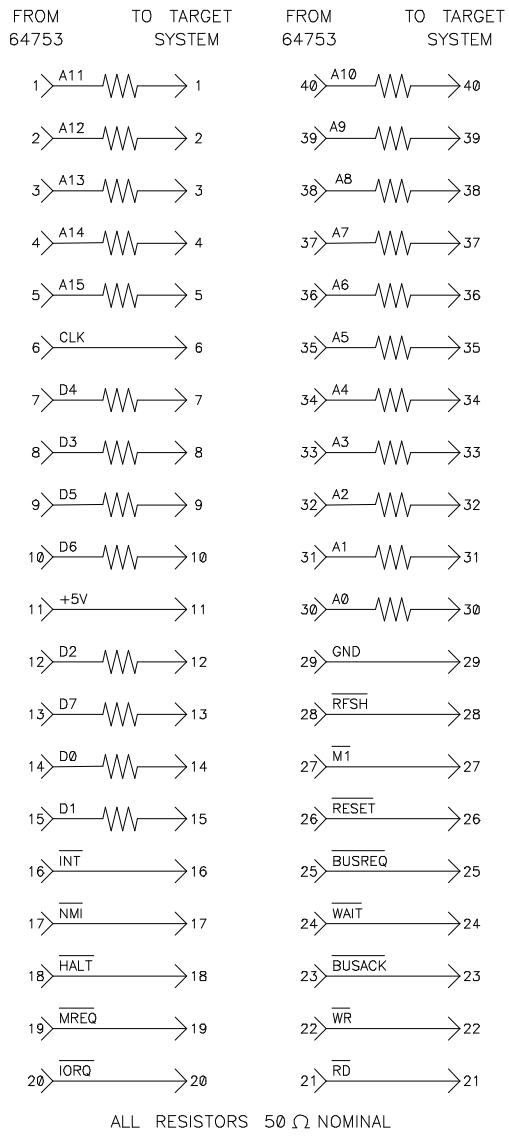


Figure 3-1. High-Speed CMOS Circuit Schematic

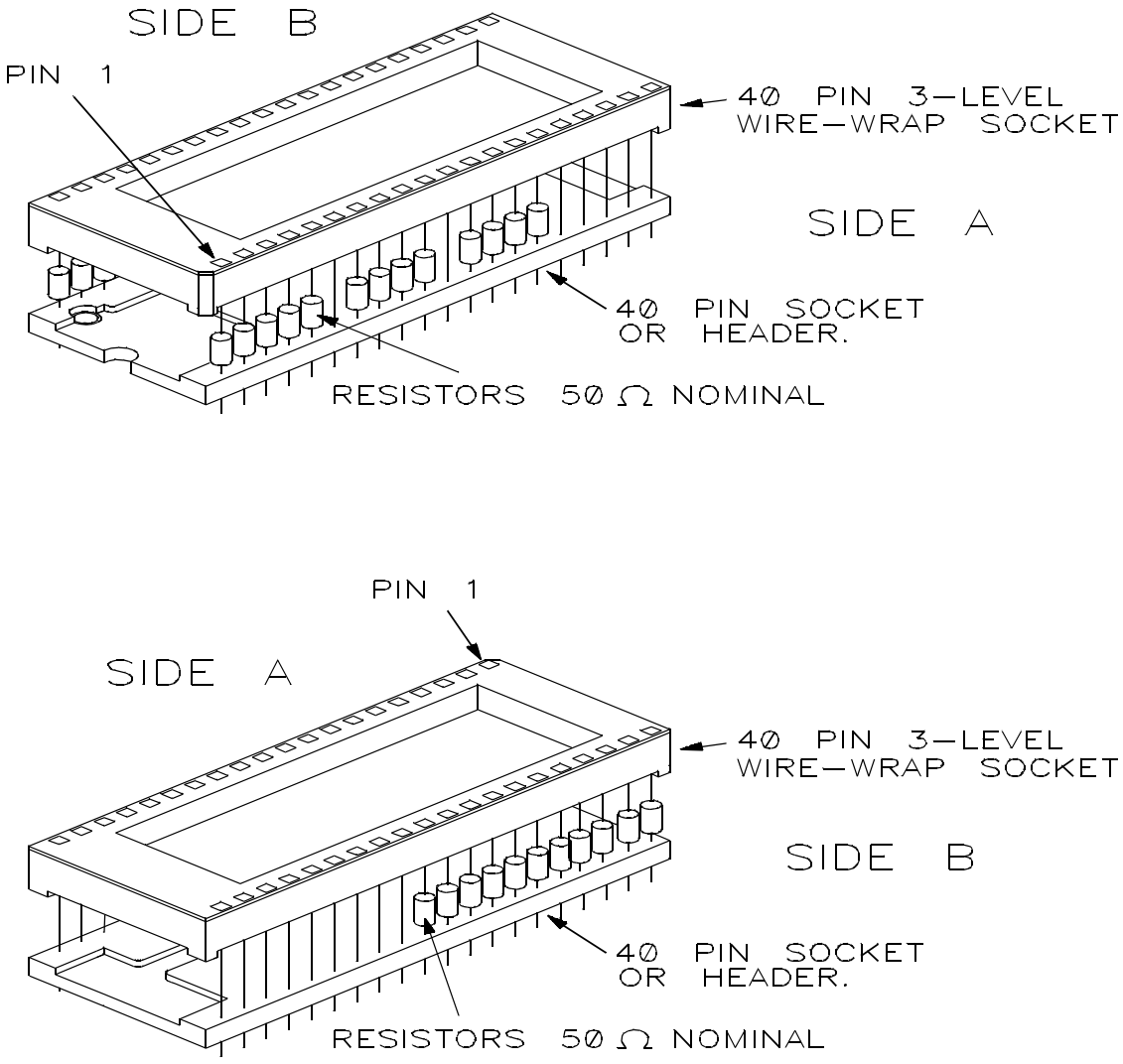


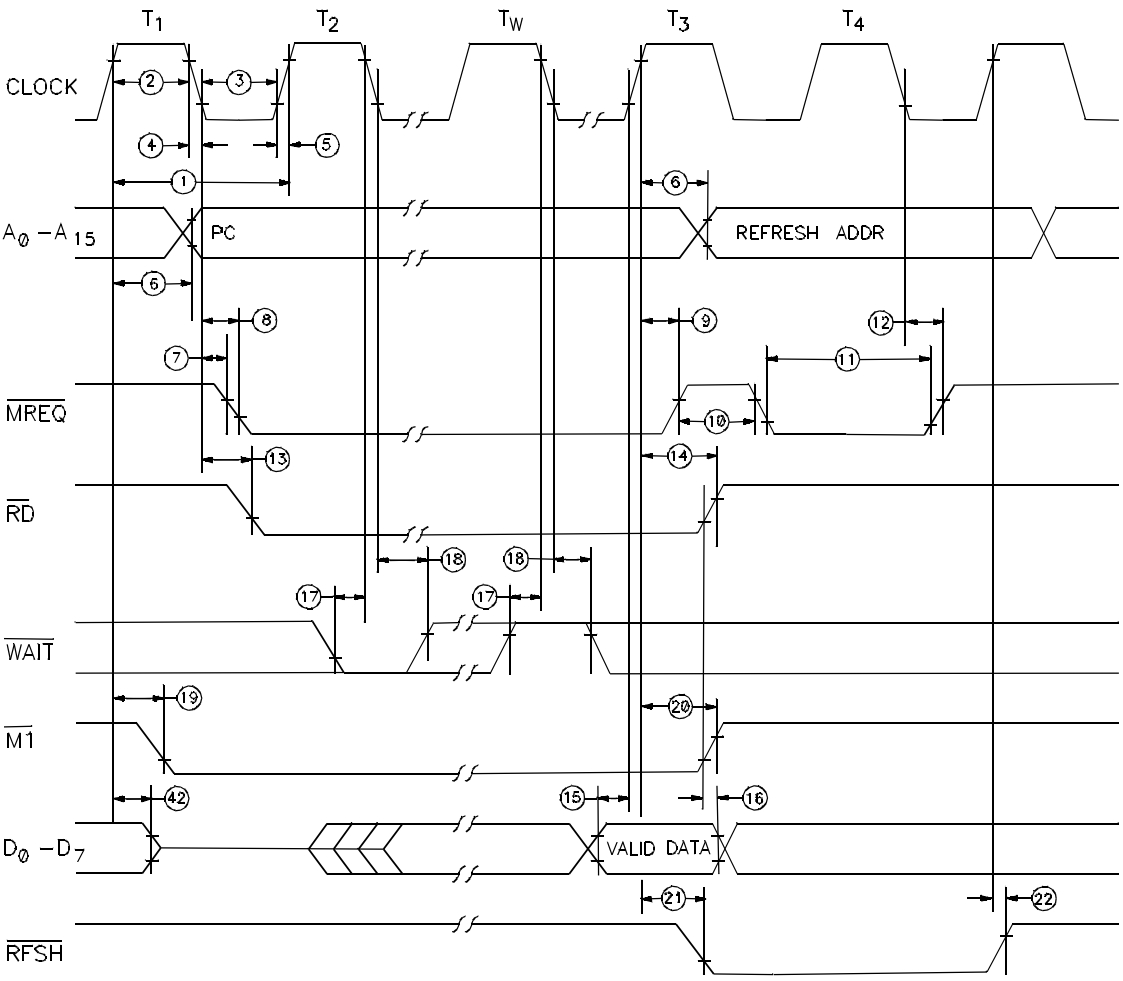
Figure 3-2. Example Build Of Circuit

A

Z80 Emulator Characteristics

- Z80 Signals
- Timing Characteristics Of The Z80 Emulator And Microprocessor
- Specifics On Z80 Emulator Interaction With A Target System

Timing diagrams A-1 through A-7 and table A-1 are:



NOTE: T_W – Wait cycle added when necessary for slow ancilliary devices.

Figure A-1. Opcode Fetch Cycle Timing

A-2 Z80 Emulator Characteristics

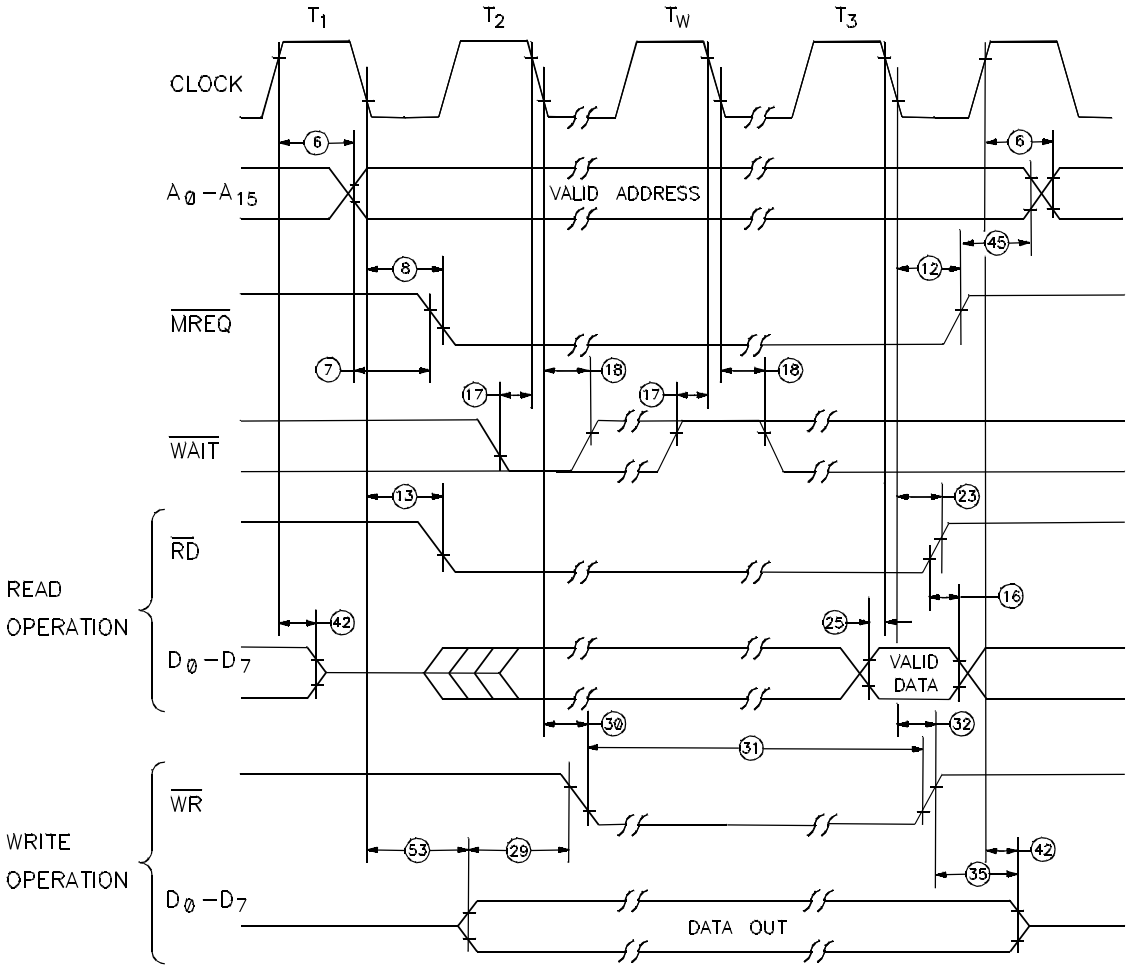
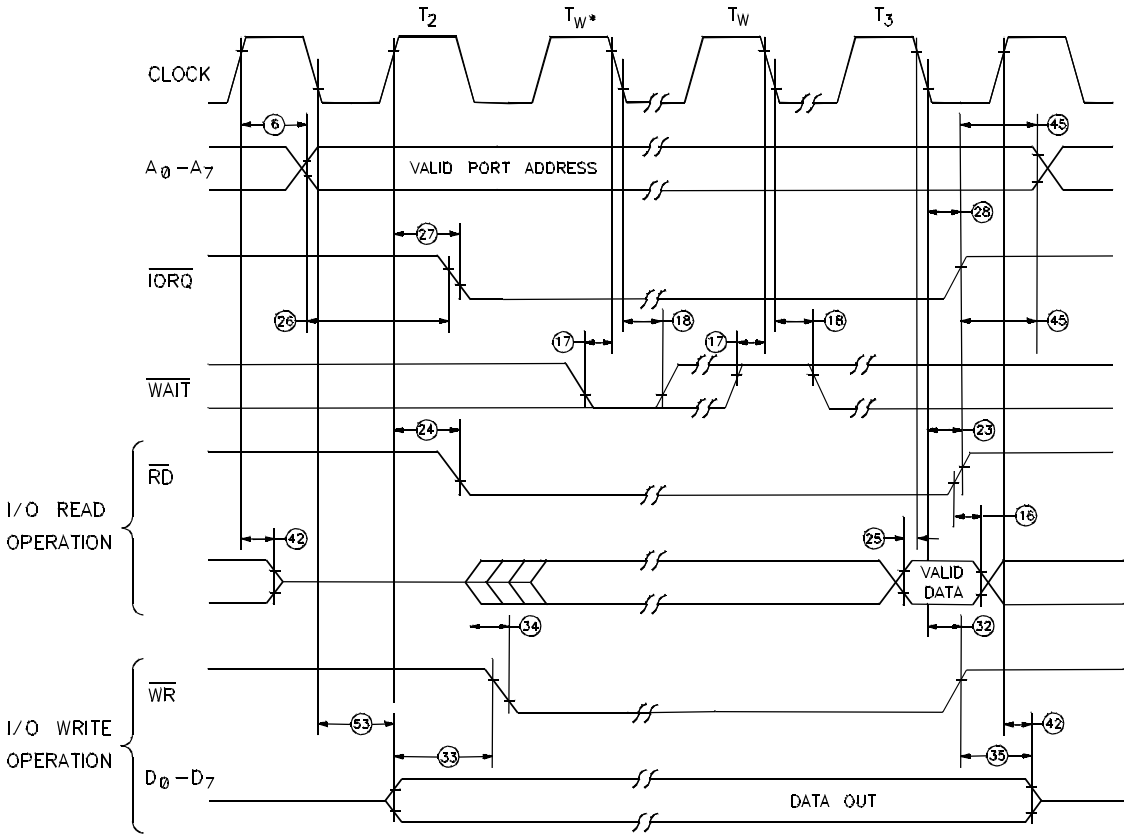


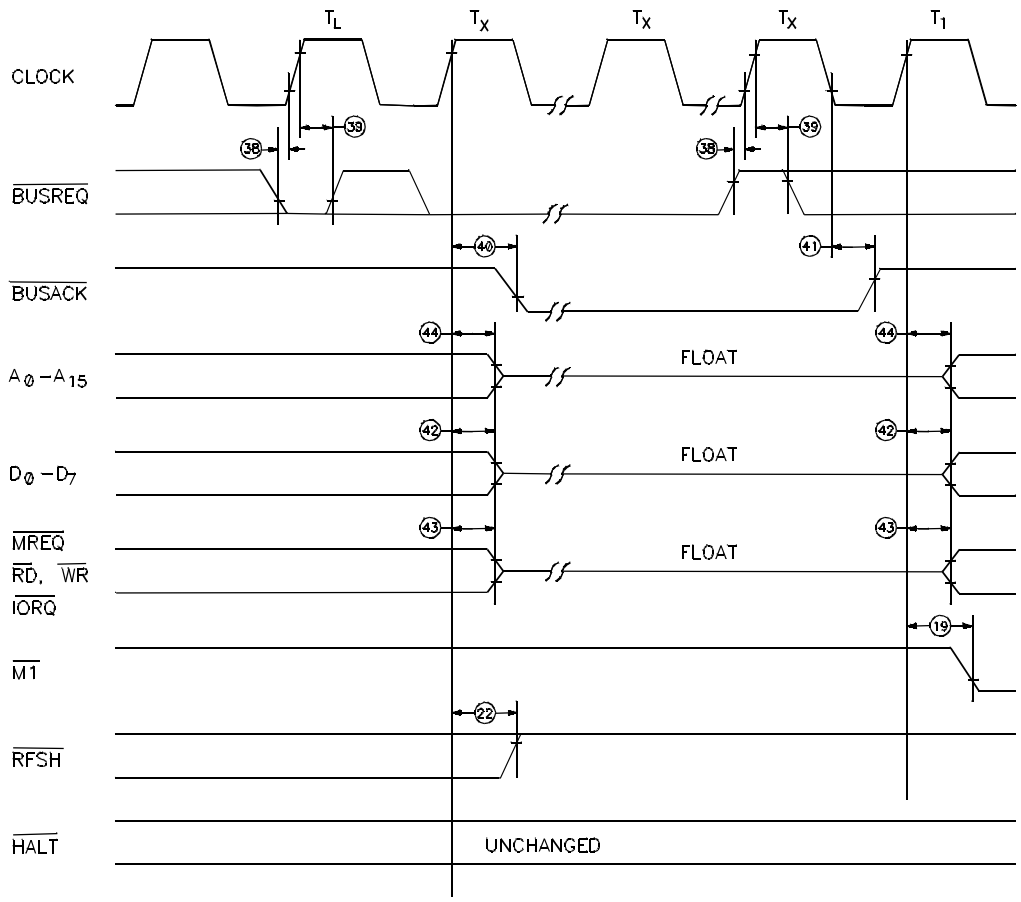
Figure A-2. Data Memory Read/Write Cycle Timing



NOTE: T_{W*} = One Wait cycle automatically inserted by CPU.

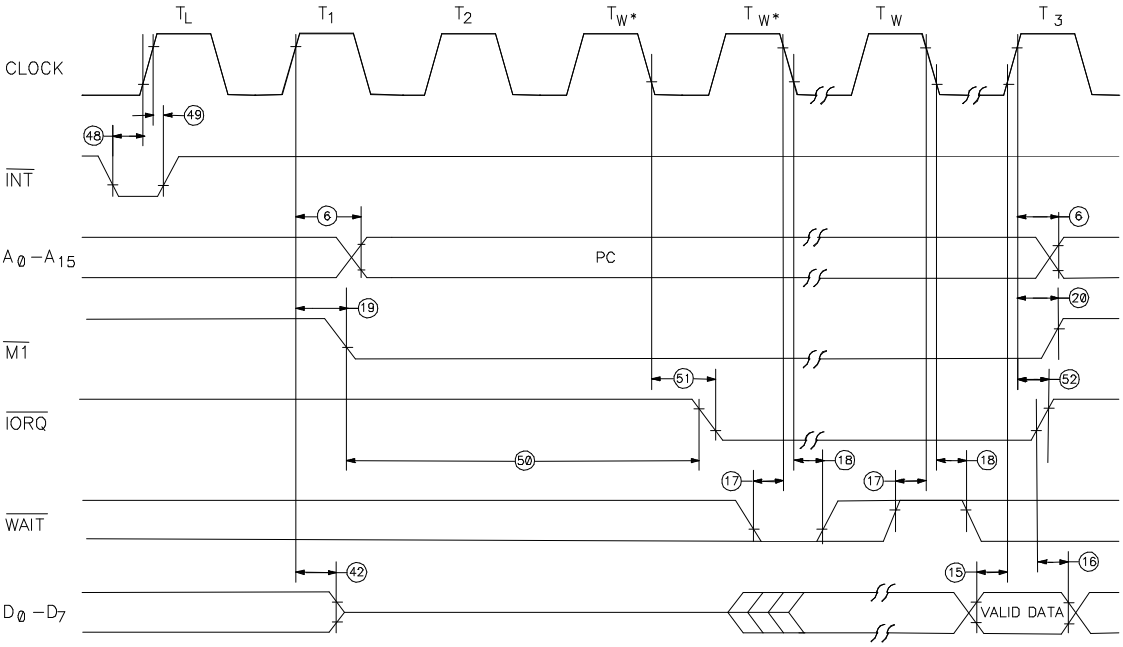
Figure A-3. Input/Output Read And Write Timing

A-4 Z80 Emulator Characteristics



NOTE: T_L = Last state of any M cycle. T_X = An arbitrary clock cycle used by requesting device.

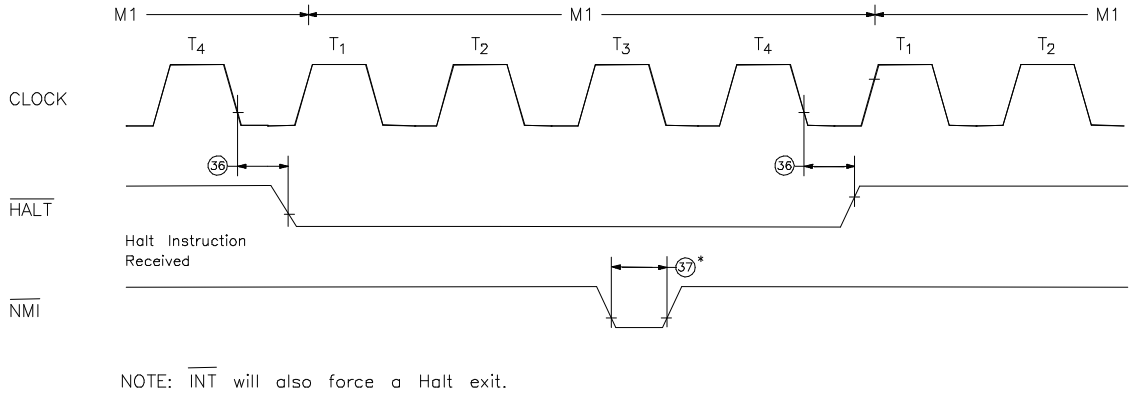
Figure A-4. Bus Request/Acknowledge Timing



NOTE: 1) T_L = Last state of previous instruction. 2) Two Wait cycles automatically inserted by CPU(*).

Figure A-5. Interrupt Request/Acknowledge Timing

A-6 Z80 Emulator Characteristics



NOTE: $\overline{INT}$ will also force a Halt exit.

Figure A-6. Halt Acknowledge Cycle Timing

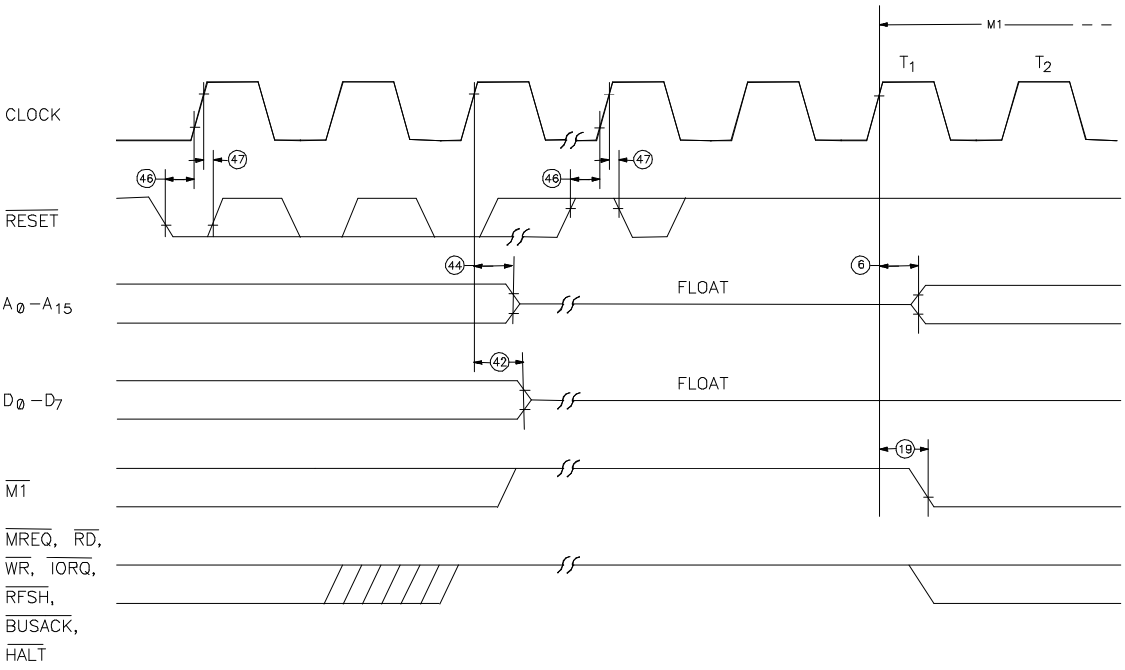


Figure A-7. Reset Timing

A-8 Z80 Emulator Characteristics

Note All times in table A-1 are measured at the target system end of the emulation probe cable with external clock selected.

All times are in nanoseconds.

Table A-1. Performance Characteristics

Number	Symbol	Parameter	64753A	
			min	max
1	TcC	Clock Cycle Time	100*	DC
2	TwCh	Clock Pulse Width (High)	45	DC
3	TwCl	Clock Pulse Width (Low)	45	DC
4	TfC	Clock Fall Time	-	10**
5	TrC	Clock Rise Time	-	10**
6	TdCr(A)	Clock rising to Address Valid Delay	-	110
7	TdA(MREQf)	Address Valid to LMREQ falling Delay	10	-
8	TdCf(MREQf)	Clock falling to LMREQ falling Delay	-	90
9	TdCr(MREQr)	Clock rising to LMREQ rising Delay	-	90
10	TwMREQh	LMREQ Pulse Width (High)	30*	-
11	TwMREQl	LMREQ Pulse Width (Low)	75*	-
12	TdCf(MREQr)	Clock falling to LMREQ rising Delay	-	90
13	TdCf(RDf)	Clock falling to LRD falling Delay	-	100
14	TdCr(RDr)	Clock rising to LRD rising Delay	-	90
15	TsD(Cr)	Data Setup Time to Clock rising	30	-
16	ThD(RDr)	Data Hold Time to LRD rising	0	-
17	TsWWAIT(Cf)	LWAIT Setup Time to Clock falling	50	-
18	ThWAIT(Cf)	LWAIT Hold Time after Clock falling	10	-
19	TdCr(Mlf)	Clock rising to LM1 falling Delay	-	100
20	TdCr(Mlr)	Clock rising to LM1 rising Delay	-	100
21	TdCr(RFSHf)	Clock rising to LRFSH falling Delay	-	125
22	TdCr(RFSHr)	Clock rising to LRFSH rising Delay	-	115
23	TdCf(RDr)	Clock falling to LRD rising Delay	-	90
24	TdCr(RDf)	Clock rising to LRD falling Delay	-	90
25	TsD(Cf)	Data Setup to Clock falling during M2, M3, M4, or M5 Cycles	30	-

Table A-1. Performance Characteristics (Cont'd)

			min	max
26	TdA(IORQf)	Address Stable prior to LIORQ falling	50*	-
27	TdCr(IORQf)	Clock rising to LIORQ falling Delay	-	85
28	TdCf(IORQr)	Clock falling to LIORQ rising Delay		90
29	TdD(WRf)	Data Stable prior to LWR falling	-20*	-
30	TdCf(WRf)	Clock falling to LWR falling Delay	-	90
31	TwWR	LWR Pulse Width	75*	-
32	TdCf(WRr)	Clock falling to LWR rising Delay	-	90
33	TdD(WRf)	Data Stable prior to LWR falling	-65*	-
34	TdCr(WRf)	Clock rising to LWR falling Delay	-	85
35	TdWRr(D)	Data Stable from LWR rising	5*	-
36	TdCf(HALT)	Clock falling to LHALT rising or falling	-	255
37	TwNMI	LNMI Pulse Width	60	-
38	TsBUSREQ(Cr)	LBUSREQ Setup Time to Clock rising	60	-
39	ThBUSREQ(Cr)	LBUSREQ Hold Time after Clock rising	10	-
40	TdCr(BUSACKf)	Clock rising to LBUSACK falling Delay	-	110
41	TdCf(BUSACKr)	Clock falling to LBUSACK rising Delay	-	110
42	TdCr(Dz)	Clock rising to Data Float Delay	-	110
43	TdCr(CTz)	Clock rising to Control Outputs Float Delay (LMREQ, LIORQ, LRD, and LWR)	-	110
44	TdCr(Az)	Clock rising to Address Float Delay	-	110
45	TdCTr(A)	LMREQ rising, LIORQ rising, LRD rising, and LWR rising to Address Hold Time	5*	-
46	TsRESET(Cr)	LRESET to Clock rising Setup Time	65	-
47	ThRESET(Cr)	LRESET to Clock rising Hold Time	10	-
48	TsINTf(Cr)	LINT to Clock rising Setup Time	65	-
49	ThINTr(Cr)	LINT to Clock rising Hold Time	10	-
50	TdM1f(IORQf)	LM1 falling to LIORQ falling Delay	200*	-
51	TdCf(IORQf)	Clock falling to LIORQ falling Delay	-	90
52	TdCf(IORQr)	Clock rising to LIORQ rising Delay	-	90
53	TdCf(D)	Clock falling to Data Valid Delay	-	145

A-10 Z80 Emulator Characteristics

*For clock periods other than the minimum shown in the table, calculate parameters using the following expressions. Calculated values above assumed TrC = TfC = 10 nS. All timings assume equal loading on pins with 50 pF. Timings are subject to change.

**When the clock frequency is 10 MHz, the maximum clock rise and fall time is 10 nS. With slower clock speeds, the rise and fall time can be up to 30 nS.

Notes to Z80 AC Characteristics

Number	Symbol	64753A
1	TcC	TwCh + TwCl + TrC + TfC
7	TdA(MR)	TwCh + TfC - 45
10	TwMRh	TwCh + TfC - 25
11	TwMR1	TcC - 25
26	TdA(IR)	TcC - 50
29	TdD0(WRm)	TcC - 120
31	TwWR	TcC - 25
33	TdD0(WRi)	TwCl + TrC - 120
35	TdWR(D0)	TwCl + TrC - 50
45	TdCT(A)	TwCl + TrC - 50
50	TdM1(IR)	2TcC + TwCh + TfC - 55

Notes

B

Z80 Emulator Specific Syntax

What Is In This Appendix?

- Z80 Specific Syntax Diagrams And Variables
- Descriptions Of The Syntax Options
- Examples Of Command Use

Note 

For details about common emulation command syntax, refer to the *HP 64700 Emulators Terminal Interface User's Reference.*

Z80 Specific Syntax Diagrams And Variables

Syntax diagrams unique to the Z80 emulator include:

CONFIGURATION ITEMS

I/O

Variables used in the syntax diagrams that are unique to the Z80 emulator include:

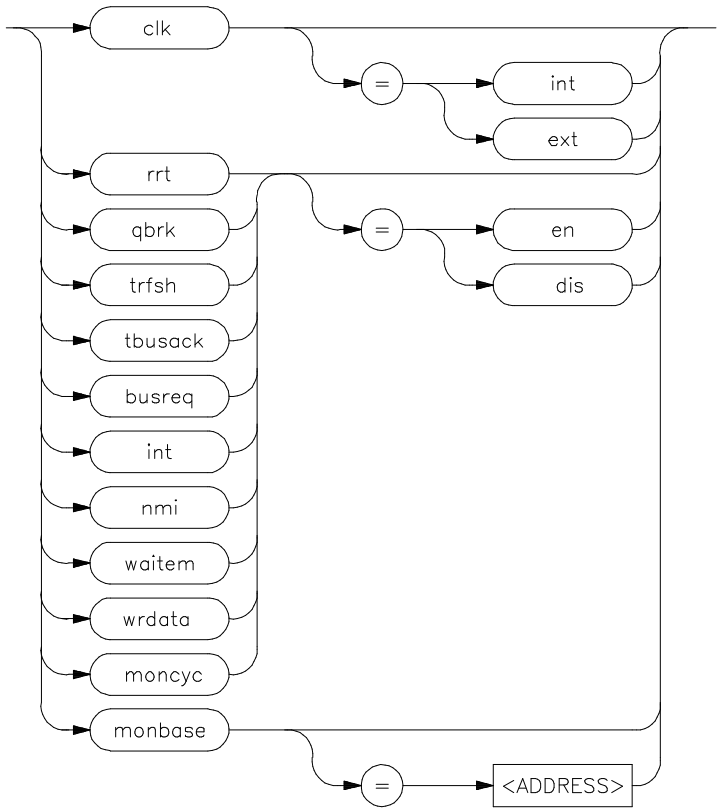
ADDRESS

DISPLAY MODE

REGISTER CLASS

Z80 Emulator Configuration Items

The Z80 configuration items allow you to set up the emulator in a way that best suits your system needs. You will likely use some of the configuration items more often than others. Most of the items allow you to set up the emulator to work properly with your target system. In addition, most of the configuration items are either enabled or disabled.



B-2 Z80 Emulator Specific Syntax

When connecting the emulator to a target system, you may be interested in modifying all of the configuration items. That all depends on your target system requirements. If your target system has a clock, we recommend that you set the clock to "external", so that operation of your program will be relative to the target system clock.

For example:

To select the target system clock, type: **cf clk= ext**

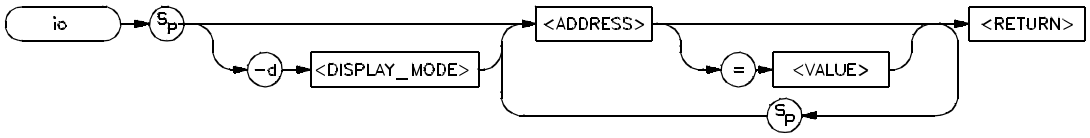
To allow the emulator to break into the monitor only when a reset, break, run, or step command is received, type: **cf rrt= en**

To cause the emulator to ignore a target system request for bus access, type: **cf busreq= dis**

To allow the emulator to write data to the target system during all read cycles from emulation memory, type: **cf wrdata= en**

I/O

The Z80 I/O command allows you to send data to and receive data from a target system (I/O device).



The Z80 processor addresses I/O ports separate from memory, and has two sets of instructions for I/O read and write operations. You can display and modify data at the I/O port addresses. When displaying the data at I/O ports, you can specify that the display format be in bytes or words.

For example:

To display I/O location 0 in word format, type: **io -dw 0**

Note 

The Z80 microprocessor ports are 8 bits wide. Therefore, using the "word format option" has no effect on the I/O display.

To display multiple I/O locations, type: **io 0 1 2**

To modify a single I/O location, type: **io 0ffffh= 44**

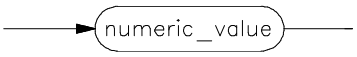
To modify multiple I/O locations, type: **io 0= 1 2= 1 3= 1**

Note 

The In-Circuit Emulation chapter in this manual also contains information on I/O.

Address

When you see the address variable in various syntax diagrams, it is unique for the Z80 emulator. The address variable indicates that you should type in an address in a form recognized by the Z80 emulator.



Valid Z80 address include 0 through 0ffffh. This is true for both memory and I/O addresses. When no base letter is specified, such as "o", the default is hexadecimal ("h").

For example:

To display memory at a single address, type: **m 100**

To observe a range of memory locations in hexadecimal byte format, type: **m -db 0..0ffh**

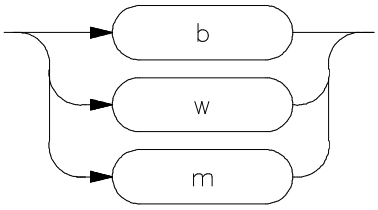
To set memory location 00ffh to 0, type: **m 0= 0ffh**

To display a block of memory starting at address 0000h, type: **m 0..**

To modify multiple memory locations to a pattern, type: **m 100h= 1,2,3,4**

Display Mode

The display mode variable indicates that you can display memory and I/O locations in various formats with the Z80 emulator.



You can display memory in bytes, words, or mnemonics.

You can display I/O locations in bytes or words.

For example:

To display memory locations 0 through 200 decimal in byte format, type: **m -db 0..200t**

To display memory locations 800 through 0fff hexadecimal in word format, type: **m -dw 800h..0fffh**

To display memory locations 800 through 810 octal in mnemonic format, type: **m -dm 800o..810o**

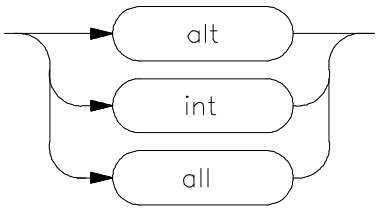
To display I/O location 7fh in byte format, type: **io -db 7fh**

Note

When memory is displayed in words, the low byte of the word that is displayed is from the specified address. The high byte is from the next higher addressed memory location. This is consistent with the way the Z80 loads a 16-bit register from memory.

Register Class

The register class variable indicates that each emulator has its own unique set of registers. For the Z80 emulator, registers are grouped into classes, including: primary registers, alternate registers, interrupt registers, and all of the registers.



For example:

To display the primary set of registers, type **reg** or **reg ***

To display the Z80 set of alternate registers, type: **reg alt**

To display the Z80 set of interrupt registers, type: **reg int**

To display all of the Z80 registers, type: **reg all**

You can specify a single register name, such as the hl register, by typing: **reg hl**

C

Z80 Error Messages

What Is In This Appendix?

This appendix contains a list of error messages specifically for the HP 64753 Z80 emulator. Each error message has its own unique error number, located in the left column. Following each message is information for how to recover from that state.

Z80 Unique Error Messages

140 **Legal values for iff2 are 0 and 1**

Both of the interrupt enable flip flops signal the Z80 interrupt status. Interrupt enable flip-flop iff2 stores the content of interrupt enable flip-flop iff1 during /NMI service. Because the values for iff1 can only be 0 (disables interrupts) or 1 (enables interrupts), the content of iff2 can only be 0 or 1.

141 **Legal values for imode are 0 through 2**

The maskable interrupt /INT has three programmable response modes available. The interrupt mode register indicates which of these modes the Z80 is programmed to respond to, when an interrupt occurs. You cannot modify the imode register to a value other than 0, 1, or 2.

Real-Time Error Messages

40 **Restricted to real time runs**

Because the emulator is restricted to real-time runs, you cannot perform the action you have just taken. Emulator features not performed in real-time include displaying, loading, modifying and storing target system memory, displaying or modifying registers and single stepping. If you try to perform any of these actions while the emulator is restricted to real-time runs, you will receive this message. All other features are performed in real-time.

Reset Error Messages

60 **Waiting for target system reset**

The emulator will not execute (or continue to execute) the target system program until you reset the target system. After you press a hard reset button on the target system program execution will continue.

61 **Emulator is in the reset state**

Because the emulator is reset, you cannot perform emulation functions that require the monitor program to run. This includes displaying or modifying processor registers, target system memory, or I/O ports. To recover from this error, type **b** to start the emulator executing in the monitor.

Monitor Error Messages

100 **No response from monitor**

The emulator will not accept your command until it is executing properly in the monitor. Try issuing the **b** command to break into the monitor. If that doesn't work, try issuing the **rst -m** command to first reset the emulator, then begin executing in the monitor.

102 **Monitor failure; no clock input**

When there is no clock input to the emulator, the monitor program cannot run.

104 **Monitor failure; bus grant**

The monitor cannot run because the emulator has granted the bus.

105 **Monitor failure; halted**

The emulation processor is waiting in a HALT state for either a /INT or /NMI before it resumes operation.

106 **Monitor failure; wait state**

The Z80 emulation processor is waiting for the target system to complete its data transfer.

142 **User monitor code must be in the range 0300H thru 06FFH**

The load address specified in the "download monitor" command is not in the specified range.

Unknown Or Fatal Errors

120 **Unknown emulator error**

The cause of the error is unknown. Reset the emulator.

Notes

Index