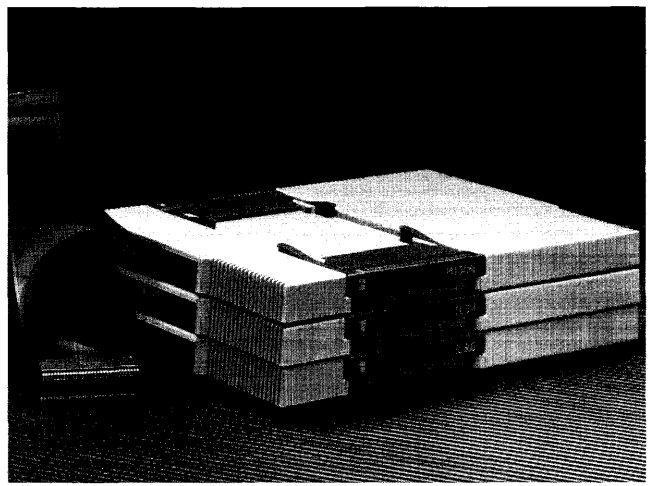
Applied
Microsystems
Corporation



EL 800
User's Manual
Z80 and HD64180
Microprocessors

Applied
Microsystems
Corporation

EL 800
User's Manual
Z80 and HD64180
Microprocessors

P/N 920-11576-02
October 1988

Table of Contents

Table of Contents

Preface

PREFACE

Unpacking the EL 800

As soon as you unpack your EL 800 emulator, examine all components for external damage. If you find any damage, file a claim with the carrier and notify Applied Microsystems Corporation. In the United States and Canada, call 206-882-2000 or 800-426-3925 and ask for Customer Service. Outside the U.S. and Canada, please contact your local sales office or representative.

After checking for external damage, verify that the following components are present:

— Base unit module

— Probe module and attached probe tip (the probe tip is packaged in a protective cardboard box)

— Male-to-male microprocessor socket

— *EL 800 User's Manual*

— RS-232 cable (9 to 25 pin)

— RS-232 AT adapter cable (25 pin to 9 pin)

— 2 floppy disks containing emulator control software

— 5 wires with clips: one black (ground wire), four red (for connecting to other equipment)

— Power supply: 110 vac or 220 vac
  - 110 vac power supply includes power cord
  - 220 vac power supply (international) does not include power cord

— Vertical support stand

— Optional accessories:
  - carrying case

Remove the EL 800 from the shipping carton and set it on a flat surface, preferably the work site. Save the protective cardboard box surrounding the probe-tip to use when you are not using the probe tip in a target. Save the rest of the packaging in case you have to ship or transport the EL 800.

Detailed installation instructions begin in Section 2.

Service

If the EL 800 unit needs to be returned for repairs, please follow these instructions.

In the United States and Canada,

1. Call 206-882-2000 or 800-426-3925 and ask for Customer Service. They will give you a return authorization number and shipping information.

Outside the U.S. and Canada,

1. Please contact your local sales office or representative for repair procedures.

After the expiration of the warranty period, service and repairs are billed at standard hourly rates, plus shipping to and from your premises.

Limited Hardware Warranty

Applied Microsystems Corporation warrants that all Applied Microsystems manufactured products and associated hosted control software will be free from defects in materials and workmanship from date of shipment for a period of one (1) year, with the exception of mechanical parts (such as probe tips, cables, pin adapters, test clips, leadless chip sockets, and pin grid array adapters), which are warranted for a period of 90 days. If any such product proves defective during the warranty period, Applied Microsystems Corporation, at its option, will either repair or replace the defective product. This warranty applies to the original owner only and is not transferrable.

To obtain warranty service, the customer must notify Applied Microsystems Corporation of any defect prior to the warranty expiration and make arrangements for prepaid shipment to Applied Microsystems Corporation. Applied Microsystems Corporation will prepay the return shipping to US locations. For international shipments, the customer is responsible for all shipping charges, duties and taxes. Prior to returning any unit to Applied Microsystems Corporation for warranty repair, a return authorization number must be obtained from Applied Microsystems Corporation's customer service department (see Service section).

This warranty shall not apply to any defect, failure or damage caused by improper use, improper maintenance, unauthorized repair, modification, or integration of the product.

Hardware Extended Warranty

Applied Microsystems Corporation's optional Extended Warranty is available for all hardware products for an additional charge at the time of the original purchase. The Extended Warranty may be purchased to extend the warranty period on mechanical parts normally restricted to 90 days to a total of one (1) or two (2) years and to extend the warranty on electrical parts and all other mechanical parts to two (2) years.

Hardware Service Agreements

Service agreements are available for purchase at any time for qualified Applied Microsystems Corporation manufactured products. The service agreement covers the repair of electrical and mechanical parts for defects in materials and workmanship. For information, contact your local Applied Microsystems Corporation sales office or representative.

Interference Warning

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. It is temporarily permitted by regulation and has not been tested for compliance with the limits of Class A computing devices pursuant to Subpart J of Part 156 of FCC Rules, which are designed to provide reasonable protection against such interference. Operation of this equipment in a residential area is likely to cause interference. It is up to the user, at his own expense, to take whatever measures may be required to correct the interference.

SECTION 1

Table of Contents

Introduction

INTRODUCTION

The EL 800 Series of microprocessor emulators provides a flexible, cost-effective approach to developing and debugging systems using 8-bit microprocessors and microcontrollers. During emulation, the emulator physically replaces the emulated microcontroller, and provides the ability to dynamically examine and modify CPU registers, memory and I/O. With it, you can stop a program and examine the history of events leading up to or surrounding a problem.

The EL 800 consists of a base unit, a choice of probe modules and optional expansion modules, and convenient, menu-driven emulator control software for the IBM PC and compatibles.

Probe modules are available for the Zilog Z80 family (Z80, Z80A, Z80B, Z80H, NMOS and CMOS versions) in the DIP package. They are also available for the 6, 8, and 10 MHz Hitachi HD64180 microprocessors with the R0, R1, and Z masks in both DIP and PLCC packages. For complete information on your chip, please refer to the reference manuals:

Zilog:	*Z80x-CPU Technical Manual*
Hitachi:	*HD64180 8-Bit High Integration CMOS Microprocessor User's Manual*

Additional chip specific information is available in Section 5 of this manual.

How to Use This Manual

This manual is your guide to using the Applied Microsystems Corporation's EL 800 Emulator for Zilog Z80H and Hitachi HD64180 microprocessors. It describes the use of the EL 800 emulator, and is not intended to be used as a general "how-to" guide to emulation, although you may find that it contains useful examples and procedures for debugging problems.

If this is your first time using the EL 800, read through the Introduction, Getting Started, and Tutorial sections.

Once you are familiar with the emulator, the *Operation* section is an alphabetical reference to the windows and commands within each window.

The comprehensive index and Appendix A (Command Summary) are useful for finding specific information in the manual.

The manual is organized as follows:

Section 1: Introduction introduces Applied Microsystems Corporation's EL 800 emulators. It introduces the features of the EL 800 and explains the uses of emulation throughout the development cycle. It also describes this manual's layout and general contents.

Section 2: Getting Started provides instructions for setting up and starting the EL 800 emulator and control software. Hardware and software installation instructions are included.

Section 3: Tutorial is designed to familiarize you with using your emulator. This section describes a sample session, including downloading a code file, setting breakpoints, and setting more advanced events.

Section 4: Hardware contains information on each module: emulator, probe and optional modules, as well as details on the serial port, maintenance and troubleshooting.

Section 5: Chip and Emulator Characteristics describes the unique features of your target microprocessor which affect emulation, and explains how the EL 800 addresses these unique features.

Section 6: Operation is an alphabetic reference for using each window.

The **Appendices** provide a handy reference for EL 800 commands, information on cables, object file formats, error messages, when certain information is stored, and the use of expressions in the control software. Appendix G provides a detailed guide to using the cross triggering capability of the EL 800 for multiprocessor development.

Appendix A: Command Summary
Appendix B: Serial Interface: Cable Information
Appendix C: Object File Formats
Appendix D: What Happens When ...
Appendix E: Error Messages
Appendix F: Using Expressions
Appendix G: Debugging Multiprocessor Systems

The EL 800 Emulator

The EL 800 emulator, used with a host computer, is a complete development system for software development and testing, hardware testing and hardware/software integration. The EL 800 has two basic parts: hardware which allows target access, target control and recording of target activity, and a set of software programs which allow software development and debugging, and provide a convenient human interface to the hardware functions.

Features include:

- Real time, transparent emulation at full system clock speed.
- Full symbolic debug capability.
- Two powerful breakpoint and event systems:
  - Basic Breakpoint System which provides up to one million address and range breakpoints.
  - easy to use, powerful Advanced Event System, which provides a state transition system for setting breakpoints and controlling the target.
- 8K x 48 bit high speed trace memory.
- Up to 1 MB of overlay memory.
- Convenient, menu-driven emulation control software.
- Quick, convenient switching between microprocessors.

Real-Time, Transparent Emulation

The EL 800 emulator runs at full target clock speed, with less than 5 ns signal skew, so you can trust that a program that works during the development process will actually work in real-time in the target.

No wait states are inserted and no interrupts are preempted for breakpoints. The emulator is transparent to the target system, so there are no restrictions on memory, software or interrupts. The latest surface mount and FET technology make it possible to have less than 5 ns timing corruption and to maintain transparent CPU driver levels.

Trace Memory

The 8K x 48 bit Trace Memory reveals the actual sequence of program instructions, providing a way to follow the events leading up to or around a problem. The trace data lets you selectively capture execution. Trace data can be captured in real time for clock cycle speeds up to 10 MHz for the HD64180 and up to 8 MHz for the Z80H, without slowing down the target system.

Powerful Breakpoint and Event Systems

There are two breakpoint/event systems in the EL 800: the Basic Breakpoint system, and a state machine Advanced Event System.

The Basic Breakpoint System provides up to 1 million address or range breakpoints. The number of breakpoints available depends on the number of overlay memory modules you have installed.

The powerful Advanced Event System is standard with all EL 800 emulators, and allows you to debug complex, hierarchical problems:

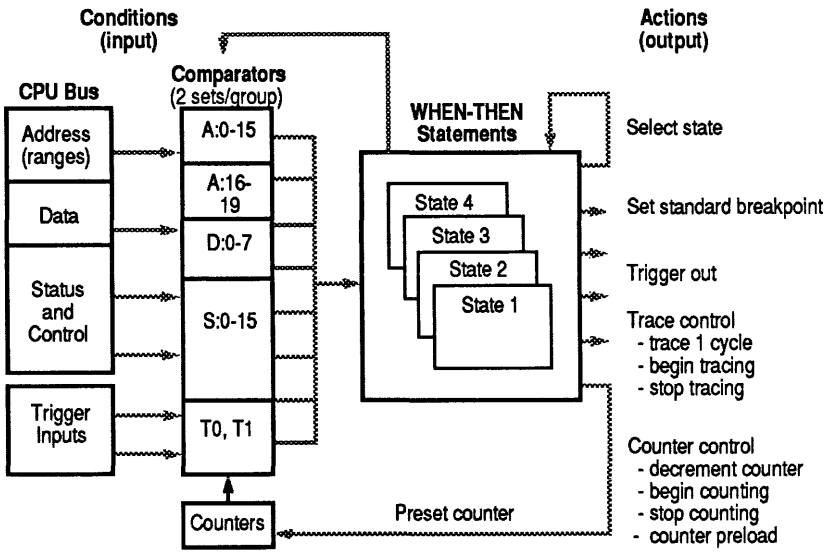
— Flexible comparators and logical combinations give you complete breakpoint control.

— When a condition is reached, you have more choices than just breaking emulation.

— Four nested levels of statements let you monitor recursive and reentrant events.

You define conditions as combinations of comparators, including data, status, address, counters and control lines. You have a choice of actions (or a combination of actions) when a condition is reached, such as breaking emulation, tracing one cycle, starting or stopping trace acquisition, controlling the counter, triggering another device and moving between event states.

Figure 1-1. The Advanced Event System



Complex Triggering Capability

The two trigger outputs can be used to:

— trigger a logic analyzer

— trigger another EL 800 or an ES 1800 emulator for debugging multiprocessor development projects

The two trigger inputs can be used to:

— gather additional input from your target board

— receive a trigger from another EL 800 or an ES 1800 emulator

On the EL 800 for the Z80 microprocessor, there are four additional address lines which can be used for four additional inputs from your target.

Easy Transition Between Target CPU's

To emulate a different microprocessor, you just need to unstack one probe module, replace it with a new probe module, load new host software onto your computer and download new shell code to your emulator.

User Interface

The emulation control software provides a window for each emulator function, such as configuration, trace, register access, and Advanced Event System control. You can easily switch between windows, or put several on the screen at once.

For windows that require input, you enter information on the command line (the bottom line of the screen). A help message showing valid choices and command syntax is always displayed to the right of the prompt, and comprehensive help for each prompt is available by typing a question mark (?).

You can control your entire development process through the EL 800: downloading, uploading, and emulation functions are all part of the control software, and convenient shell escapes let you escape to DOS or run your favorite editor, file viewer or make utility.

Symbolic Debugging

You can refer to trace data, events, breakpoints, memory and I/O locations by their assembly source symbolic name, so you don't need to use hex addresses. The emulator displays the symbolic names along with the physical values during memory disassemblies, trace displays, and emulation control.

Optional Modules

Optional modules allow you to conveniently add functionality to your EL 800 emulator. The entire EL 800 system is designed to be extendible for a variety of modules.

Overlay Memory

Overlay memory replaces all or a portion of your target memory, allowing easy integration of the program with target hardware as it becomes available. You can examine, modify, verify or move data in the target system and overlay.

The overlay memory module also provides a simple breakpoint system which can break on addresses and ranges with read, write and status qualifiers. The number of breakpoints is dependent on the size of overlay memory, with a maximum of 1 MB of additional breakpoints.

A choice of 64KB, 128KB, and 256KB overlay memory is available, with or without battery backup. You can stack up to four modules together, for a maximum of 1 MB of overlay. Overlay is mappable anywhere, with 1 KB resolution and 160 ns access time. If you want to use wait states to mimic a slow target, you can set the number of wait states from 1 to 4, and you can choose to execute the desired number of wait states for each mapped memory segment.

The following commands can be used with the overlay memory:

- map segments of memory*
- set up simple breakpoints
- display raw data or disassembled data
- examine, modify, move and fill memory
- download or upload code or data to or from your PC or compatible

* Memory may be qualified as target, overlay, read/write, read only, or invalid.

Emulation as a Tool

Emulation speeds the debugging process during embedded system design by providing the ability to interactively control and examine the state of the system at any chosen time.

The EL 800 provides three key emulation functions:

1. *Visibility:* The EL 800 emulator makes it possible for you to observe the bus cycle trace, and to gather information on addresses accessed, instructions executed, registers and timing. You can halt program execution at a program state you have predefined, and look at the data for that program state. Information is stored in buffers, making it easy to find the information you need.

2. *Control:* You can control your target through one of the EL 800's two breakpoint systems: the Basic Breakpoint System, or the Advanced Event System. With the Basic Breakpoint System, you can break on individual addresses or ranges with read or write qualifiers. With the Advanced Event System, you can track any program state, and choose a variety of actions, including trace control, counter control, setting breakpoints, switching event states or sending a trigger out signal once a program state is reached.

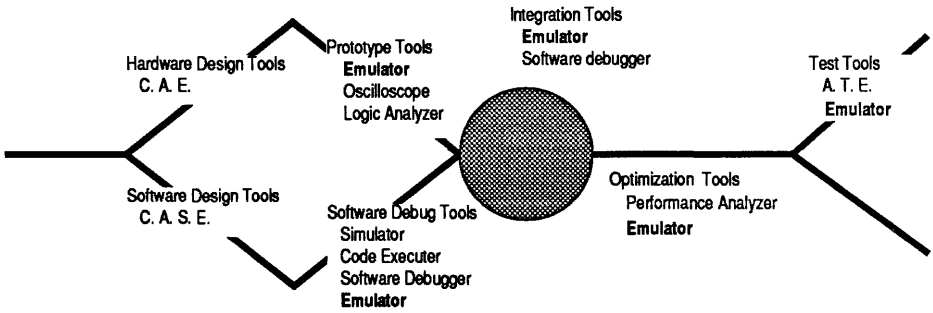
   The emulator and host software make it easy to manage development tasks, with easy download of files to target memory or to the emulator.

3. *Transparency:* When the EL 800 emulator is in use, it is virtually indistinguishable from the microprocessor in terms of code execution, logical, mechanical and electrical operation. With the CPU in the probe tip, and the latest surface mount and FET technology, there is less than 5 ns skew.

The Development Process

Each stage of developing a product requires different debugging techniques. The EL 800 is used at four stages of product development:

Figure 1-2. Tools Used During the Development Cycle



1. *When software is available, and no hardware is available.* At this stage, the EL 800 emulator can be used with the built-in test target to run your code. The overlay RAM in the emulator can be used in place of allocated memory space in the target. This RAM can be configured to act like ROM so that ROM code can be checked and modified before programming the actual chips.

2. *When prototype hardware is available.* As each board is complete, the emulator is typically first used to check for static problems around the CPU, such as the clock signal at the CPU, and then dynamic states such as whether signals are being transferred properly and bus contention problems.

   When hardware problems are discovered, an emulator and oscilloscope or logic analyzer can be used for troubleshooting, using the emulator to duplicate the problem and the oscilloscope or logic analyzer to help pinpoint the problem.

3. *When target hardware and software are available.* At the integration stage, the emulator is used to narrow down whether the problem is due to software or hardware. The emulator can step through the code and show whether the hardware or software is functioning as expected, making it easy to isolate the source of a problem. The emulator is capable of tracking deeply nested software bugs which take the majority of integration time.

4. *As part of testing.* Emulators are useful in production and production testing. The microprocessor socket is an excellent interface to most systems, since it usually has access to most of the hardware. Emulators can be used to checksum

ROM, test RAM, exercise I/O ports, etc., without the overhead of including test programs in the product. The EL 800 also has many scope loops available as target diagnostics.

Using emulation throughout the development cycle has three major advantages:

1. Emulators can help to isolate problems in software and hardware at each stage.
2. By using emulation during early development stages, you make sure that your product will work with an emulator, so that at the time-critical integration stage, you don't have to worry about whether your debug tools will run in your target.
3. Both software and hardware engineers are familiar with the same debug tool, and don't have to learn a new tool during hardware and software integration.

Emulation Steps

The specific way you will use your emulator depends on which step of the development cycle you're in, and whether your focus is on the hardware or the software. This section describes typical steps for

- a software engineer debugging code before hardware is available
- a hardware engineer debugging prototype hardware
- a hardware or software engineer debugging the product at integration

Debugging Prototype Hardware

1. Check power supply voltages in your target system to make sure you won't damage your prototype when you power it up.
2. Plug the probe tip in to the target board.
3. Turn on power to the prototype.
4. Turn on emulator power.
5. Try to run, using a target reset button if available.
6. Write a simple loop in emulator overlay memory to cycle through memory or run a diagnostic loop from the Diagnostics window.

7. Look at address, data and control lines with the emulator.
8. Check static signals around the CPU.
9. Check dynamic signals around the CPU.

Other tools useful in this process are logic analyzers, oscilloscopes, and timers. Emulators can be used to pinpoint where a problem is occurring, and can be used to trigger a logic analyzer at that exact point.

Debugging Software

1. Write the program modules.
2. Compile or assemble the modules using the appropriate cross compiler or cross assembler for your microprocessor.
3. Debug modules using a cross debugger, if available.
4. For code that requires your target to run, you must use a simulator to test the code before hardware is available.
5. For code that can work without your specific target, download the code from the PC to the emulator overlay memory and download the symbols into the control software.
6. Set up the control parameters. Tell the emulator the address at which to start executing code, define the program state(s) to break emulation or trigger another emulator or logic analyzer, and specify what information you want to see after a break takes place.
7. Emulate. Execute code within the conditions described in the previous step.
8. Observe. Examine trace memory to view the trace of code execution.

Debugging Integration of Software and Hardware

The steps of debugging during integration vary depending on the problems that you are solving. The following shows the first steps you might do.

1. Download code from the PC to the emulator overlay memory.
2. Set up the control parameters. Tell the emulator the address at which to start executing code, define the program state(s) to break emulation or trigger another emulator or logic analyzer, and specify what information you want to see after a break takes place.
3. Emulate. Execute code within the conditions described in the previous step.
4. Observe. Examine trace memory to view the trace of code execution.

SECTION 2

Table of Contents

Getting Started

GETTING STARTED

Introduction

This section explains how to set up your EL 800 emulator and target system. Once you have the hardware and software installed, Section 3 provides a tutorial on using the emulator.

Emulator Setup Requirements

The EL 800 emulator must be used with an IBM PC/XT, PC/AT or compatible computer:

- MS-DOS or PC-DOS 2.0 or higher
- 640K RAM minimum
- 1 hard disk drive
- 1 IBM compatible RS232 port (com1: or com2:)

Installation Steps

There are five steps to set up your EL 800 emulator. Each step is described in detail in this section. Please follow the installation steps in the following order. Operational details of the hardware can be found in Section 4.

1. Set up the EL 800 hardware. This includes stacking the modules, setting two sets of dip switches to indicate your baud rate and overlay memory configuration and connecting your PC to the EL 800 with the supplied cable(s). (pages 2-2 to 2-9)
2. Set up the EL 800 with either your target or the built-in test target. (pages 2-10 to 2-12)
3. Install the EL 800 software on your PC. This includes loading the software, setting up your directory structure and configuring the software. (pages 2-13 to 2-16)
4. Turn on power to the emulator and target. (page 2-13)
5. Invoke the EL 800 control software.

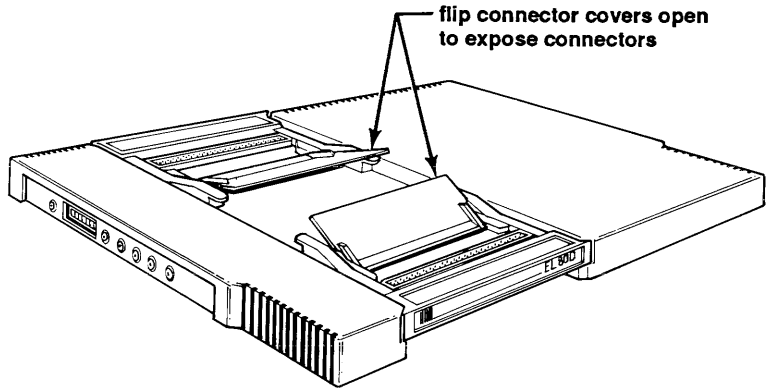
Step 1: Hardware Setup

Stack Modules

To stack the modules,

1. Flip both connector covers on the base module to the open position to expose the connectors (see Figure 2-1). Press the connector covers down so they lie flat against the base module.

Figure 2-1. Opening Connector Covers



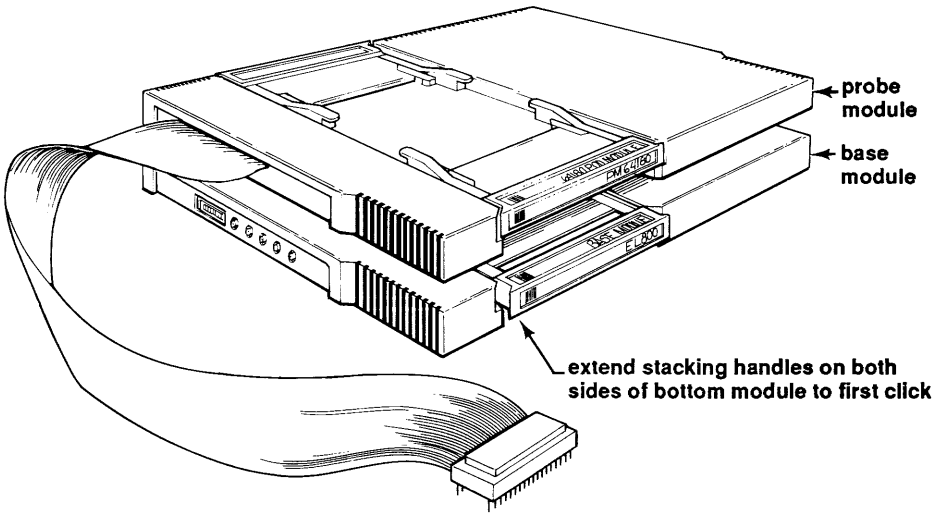
2. Pull out both grey handles on the base module to the first click. Each handle will extend about 1/3" from the module.

3. If you have not purchased any overlay modules, set the probe module on top of the base module. Press the modules together. When the modules are seated, push in both handles on the base unit, locking them together. No cables are necessary to connect the modules.

CAUTION

Forcing together misaligned modules or modules with the handles in the wrong position may bend the connector pins, making the emulator unusable.

Figure 2-2. Stacking Modules

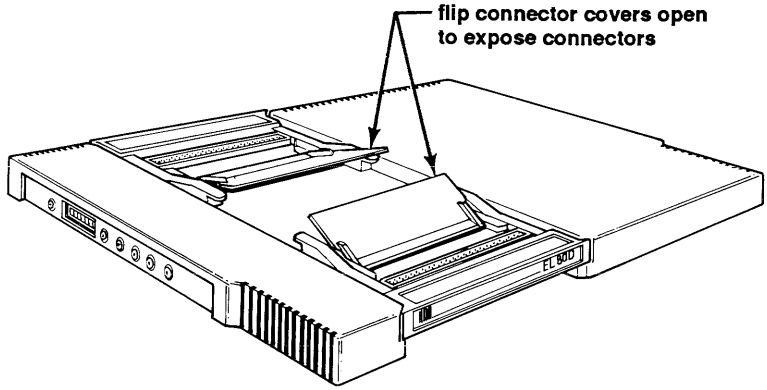


4. If you have purchased one or more overlay modules, stack them between the base and pod modules. All modules stack the same way:

   - pull out both handles on bottom module to first click (the modules cannot be connected with the handles fully extended or completely pushed in)
   - set top module on top of bottom module
   - press modules together (fingertip pressure should suffice)
   - when the modules are seated, push in both handles on bottom module to lock modules in place

   The order of the overlay modules does not matter. There are eight dip switches on the left side panel. Switches 1 and 2 (from the left) indicate the configuration of overlay modules you are using, as shown in Figure 2-3. Switches 3-8 are not use , and should be left on (1). No two overlay modules can have the same switch setting.

Figure 2-1. Opening Connector Covers



2. Pull out both grey handles on the base module to the first click. Each handle will extend about 1/3" from the module.

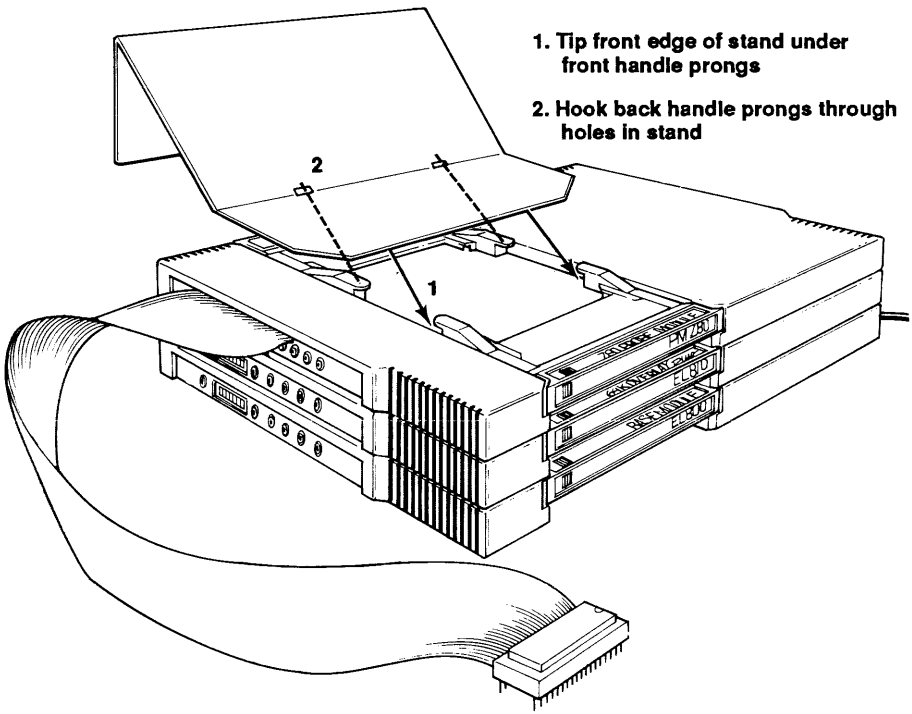
3. If you have not purchased any overlay modules, set the probe module on top of the base module. Press the modules together. When the modules are seated, push in both handles on the base unit, locking them together. No cables are necessary to connect the modules.

CAUTION

Forcing together misaligned modules or modules with the handles in the wrong position may bend the connector pins, making the emulator unusable.

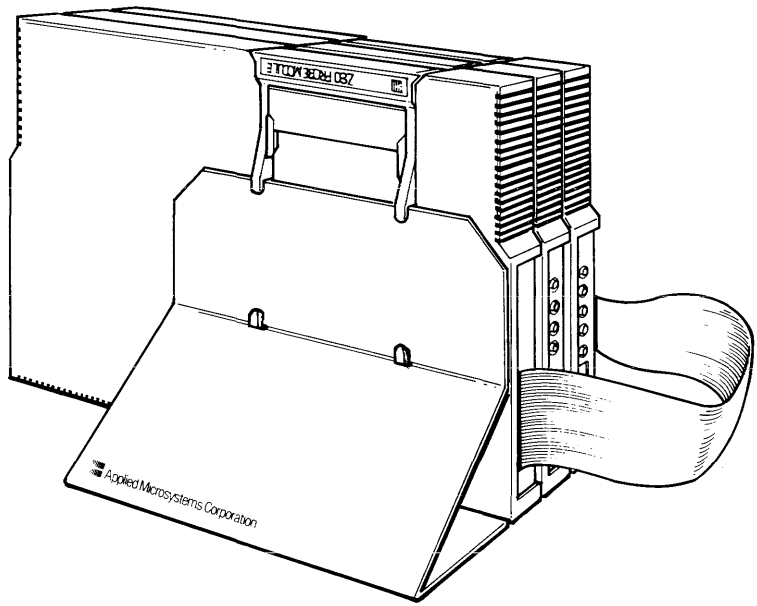
Note: When using the EL 800 with its built in target, you need to position the vertical stand so that the rubber feet of the EL 800 are **up**.

Figure 2-4. Attaching the Vertical Support Stand



Once the stand is secure, tip the EL 800 onto its side. Figure 2-5 shows a typical configuration of the EL 800 in the vertical position.

Figure 2-5. Typical Configuration: Vertical Operating Position



Emulator/PC Connection (Serial Interface)

Cables

The emulator is connected to the PC via an RS-232 cable. The EL 800 is shipped with two cables: one 9 pin to 25 pin cable, and one short 25 pin to 9 pin adapter. Use the cables required for your host computer, and connect your computer to the RS-232 port on the base unit.

Host Computer	*Cables to Use*
IBM PC XT or compatible	9 pin to 25 pin cable
IBM PC AT or compatible	9 pin to 25 pin cable *and* the 25 pin to 9 pin adapter cable

If you want to make your own cables, the cable pin connections for most PC host computers are shown in Appendix B.

Port and IRQ Setting

Configure the serial port on your PC as either *com1:* or *com2:*. If the PC has selection for RS-232 or current-loop, make sure that RS-232 voltage levels are selected. Usually, PCs are shipped with *com1:*, RS-232 selected. Appendix B contains diagrams for configuring IBM serial ports.

The EL 800 uses an interrupt driven serial receive port. If your PC serial adapter has jumpers for the interrupt request line, they must be set also. Normal IRQ settings are as follows:

Port	*IRQ Setting*
com1	IRQ4
com2	IRQ3
com3	IRQ5
com4	IRQ7

For some compatibles, the IRQ setting may vary, so check the manual for your PC and interface card.

Baud Rate

We recommend that you set your baud rate to 19,200 baud for the most efficient operation of your EL 800. The baud rate is set in two places: on the base unit and in the software (Step 4 of installation).

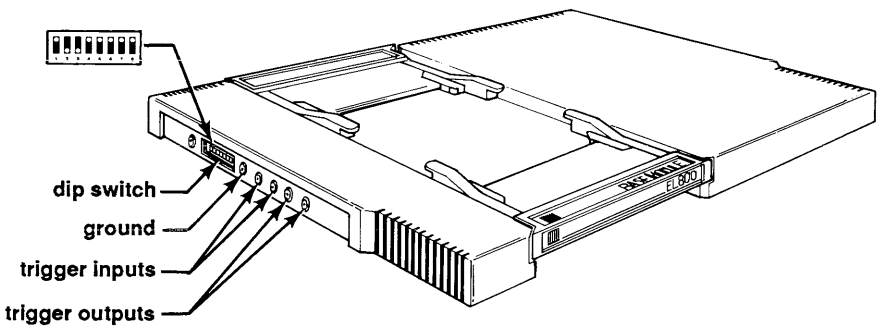
To set the baud rate on your base unit to 19,200, set up the dip switch as shown in Figure 2-6. All the available baud rates and switch settings are shown in the following table.

Baud Rate	*SW1*	*SW2*	*SW3*
19200	**1**	**0**	**0**
9600	0	1	0
4800	1	1	0
2400	0	0	1
1200	1	0	1
600	0	1	1
300	1	1	1

Note: 0 indicates off.

Figure 2-6. Base Unit, Showing Dip Switch set to 19,200

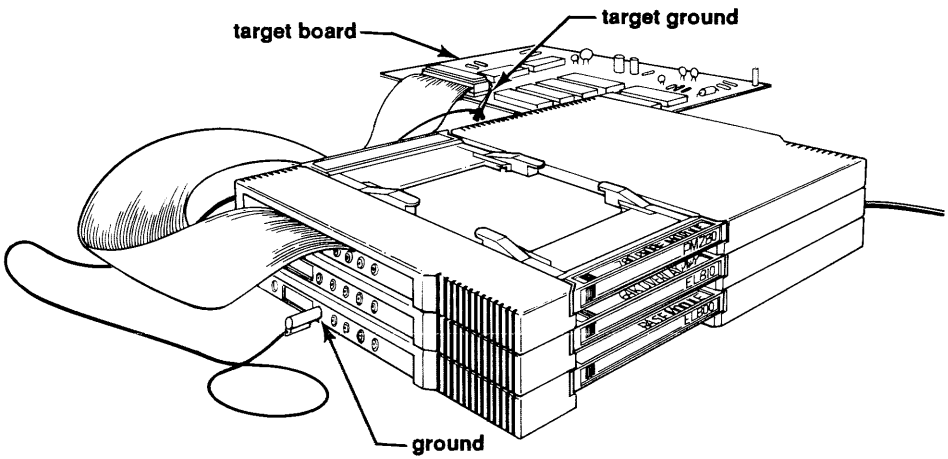
Note: The up switch position is equivalent to 1 in the above table, or ON.



Step 2: Using the Emulator With Your Target

Connect the ground on the base module to the ground on your target. If the target and emulator have different ground potentials, the emulator CPU in the probe tip may be damaged.

Figure 2-7: Ground Connection on Base Module



Plug your probe tip into the microprocessor socket on your target board, making sure to line up pin 1 on your target socket with pin 1 on the probe tip.

CAUTION

1. Plug in the probe tip to your target system *before* turning on power to your target or emulator.
2. Use standard static precautions when using the probe tip, as it is static sensitive.
3. Don't plug or unplug your probe tip with target power on.
4. Don't pull on the probe tip or cable.

Figure 2-8: DIP Probe Tip: Pin 1 Location

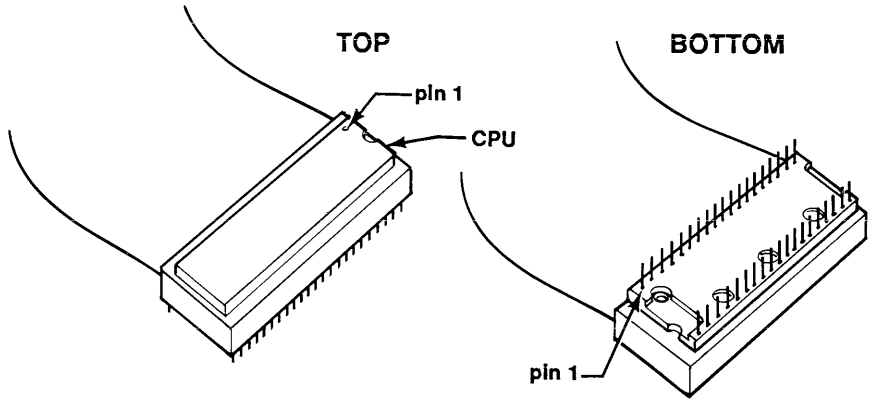
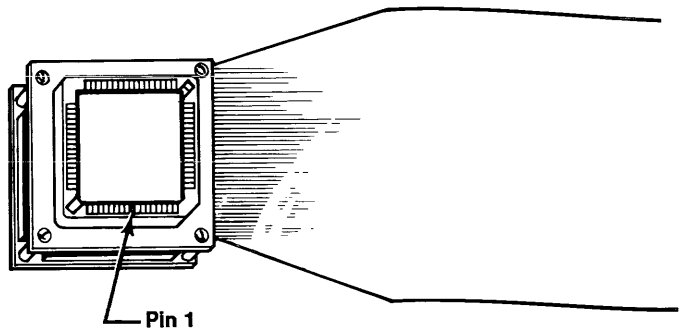


Figure 2-9: PLCC Probe Tip: Pin 1 Location



Step 2: Using the Emulator With the Test Target

The test target board is built-in to a pull-out drawer in the probe module. To use the test target board for software debugging:

1. Pull the board out from the top slot. (see Figure 2-10)
   (There is no release lever - just pull hard on the front corners of the board.)
2. Turn the board over.
3. Insert the board in the bottom slot. It goes in about 1": you'll hear a soft click when it is seated. (See Figure 2-11)
4. Insert the probe tip into the microprocessor socket on the board. (See Figure 2-11)

When you are not using the test target board, store it in the top slot.

Figure 2-10. Test Board Being Pulled Out of Storage

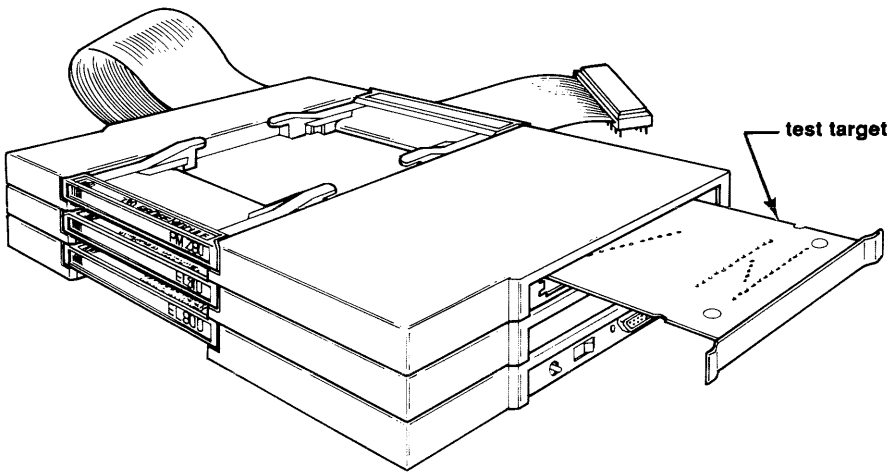
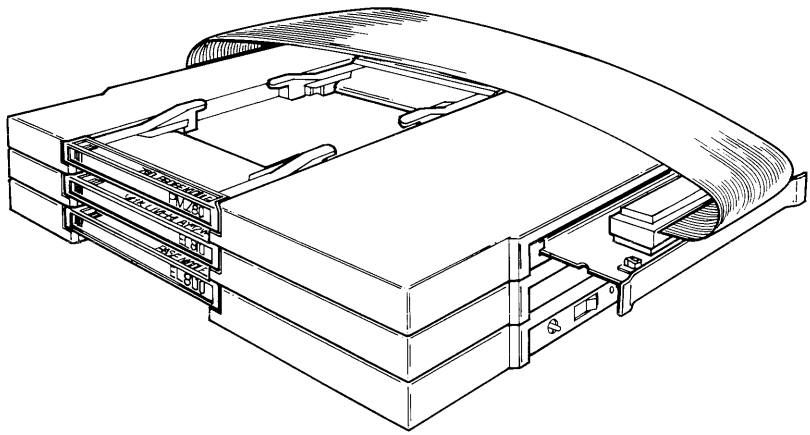


Figure 2-11. Test Board with Probe Tip Plugged In



Step 3: Software Installation: PC's and Compatibles

The EL 800 control software is shipped on two floppy disks formatted for either 360KB or 1.2MB double-sided, double-density soft-sectored disk drives. The label on your disks indicates the format.

There are four steps to installing your software:

1. Set up your directory structure.
2. Insert disk one and run the INSTALL program.
3. Set up your directory search PATH.
4. Invoke the software.

Directory Structure

While the EL 800 control software may be installed in any directory on your hard disk, we recommend that you reserve a directory by itself named **AMCTOOLS** for this software and any other utilities you may obtain from Applied Microsystems Corporation.

Run the Install Command

To install the control software, insert the disk into drive A. Make drive A the current drive by typing:

A:

Then, start the installation program by typing:

A: INSTALL

and follow the instructions.

The command **INSTALL** prompts you to:

- specify the name of the drive to install the programs on
- specify the directory to install the files in

The **INSTALL** command will then read in the appropriate configuration files from the distribution disks. You will be prompted to change disks when appropriate.

The following files are included in the distribution for the EL 800. Note that *???* is replaced by either **Z80** or **64180**, depending on which processor you are using.

README	Current release notes
???.**cfg**	Configuration database
el*???*.**exe**	Executable for the control software
el*???*.**hlp**	Help file
???.**lca**	Shell code part four: code loaded into gate arrays in base unit
???.**pod**	Shell code part three: code executed by pod processor in idle mode
*???***1.shl**	Shell code part one: operating kernel
*???***2.shl**	Shell code part two: applications software specific to processor
esxlate.pc	Error message translator
install.exe	Installation program executable file
install.fil	Configuration information for installation program
rlconfig.dat	Shell code file list
symtab.dbd	Databasee for symbol table
tutor.eth	Demonstration program for Section 3 tutorial, stored in Extended Tek Hex
tutor.sym	Symbol table for demonstration program
tutor.win	Window setup information for tutorial
h64z.lca	Special LCA file for Z-mask 64180 processors only (see the *Z-Mask Emulation with DIP Emulator* part of Section 5.)

Set Up Directory PATH

In order to use the EL 800 from other directories, your PATH must include the installation directory.

To find out the current PATH, type **PATH** at the DOS prompt. DOS will return with a string specifying the current path. If no path has been set, DOS will display "No Path".

To set a new PATH, type in the full path at the DOS prompt.

For example, you may be working on a program **MIFILE.ASM** in directory **\WORK**, have an assembler stored in directory **\ASMB**, and editor in directory **\UTIL**, the EL 800 software stored in directory **\AMCTOOLS**, and DOS stored in the directory **\DOS**. An appropriate PATH for this example would be:

path=\dos;\asmb;\util;\amctools

NOTE

Your PATH should be based on how the files are organized on *your* PC.

Once set, this path string becomes part of the DOS environment and instructs DOS to look first in the current directory, then in directory **\DOS**, then **\ASMB**, then **\UTIL** and finally **\AMCTOOLS** for a program when the program name is typed at the prompt. The EL 800 software uses the **PATH** to find its required files, such as the help file and configuration database.

The best place for the **PATH** command is in the **AUTOEXEC.BAT** file in the root directory. This causes the **PATH** command to be executed each time you boot up your PC.

See your DOS manual for more information about **PATH** and **AUTOEXEC.BAT**.

Step 4: Turn on Power

If you are using your own target:

1. Make sure the EL 800 emulator probe tip is plugged into your target *before* turning on power to the the emulator.
2. Turn on power to the target.
3. Turn on the power to the emulator.

If you are using the test target, turn on power to the emulator. The test target gets its power through the emulator.

The power switch for the emulator is on the right side of the base unit.

WARNING

Don't plug or unplug the probe tip with target power on.

Step 5: Invoke the EL 800 Control Software

To start the EL 800 control software, type:

EL*???*

where *???* is replaced by either **z80** or **64180**, depending on your target processor.

As the software loads, you'll see three status messages in the upper right corner:

Loading Configuration Parameters
Opening Emulator Interface
Clearing Advanced Event System

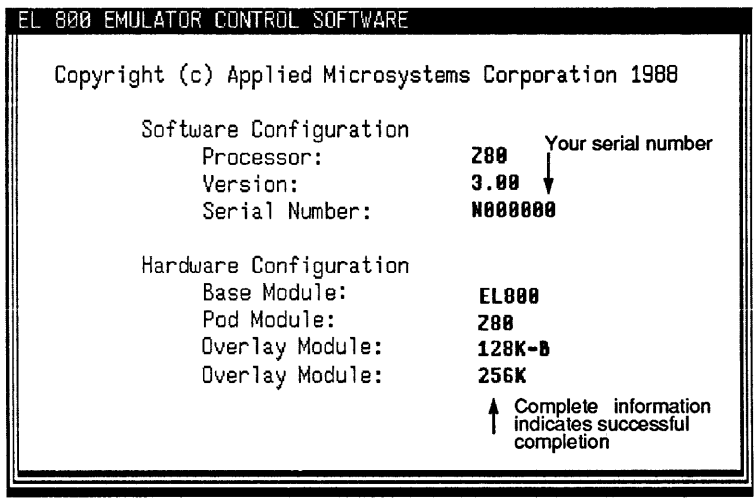
You may also see the message: **Booting Emulator, Please Wait...**

The first full screen you see (the Cover window) displays the software and hardware configuration. If you don't see any error messages, and the hardware information is displayed (Figure 2-12), you have successfully installed your EL 800 hardware and software. The rest of Section 1 covers troubleshooting startup problems, so you can now either try the tutorial in Section 3, or begin using the EL 800.

NOTE

If you are using an LCD monitor, please refer to page 6-58 for information regarding screen colors.

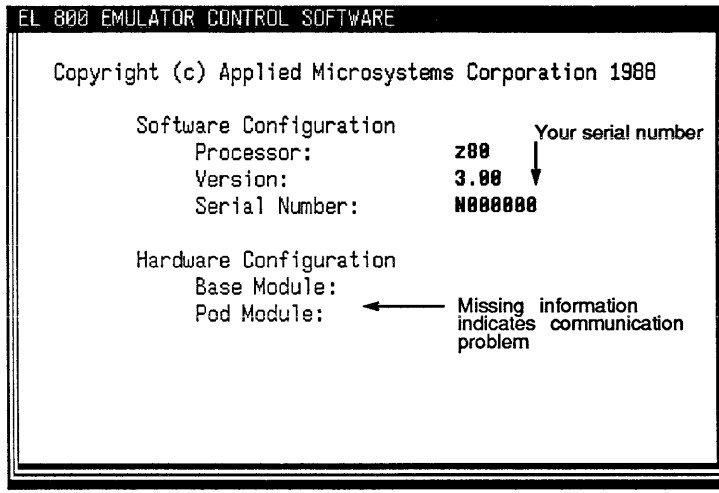
Figure 2-12. EL 800 Cover Window Showing Successful Communication



If you see an error message in the upper right corner of the screen, the software and hardware are not communicating. The chart on page 2-20 explains common startup error messages. For now, note the error message and press <esc> to clear the error message from the screen. The Cover Window will then be displayed.

Figure 2-13 shows the Cover Window if your hardware and software are not communicating properly.

Figure 2-13. EL 800 Cover Window Showing Communication Problem



There are 4 steps to getting your EL 800 hardware and software communicating:

1. Look up your error message in the table on the next page.
2. Do the actions specified.
3. Exit the software by typing **:X**<return>.
4. Start the software again.

Error Message	*Action Required*
No Target VCC; The Target Power is Off	Turn on power to your target. If you don't have a target, plug your probe tip into the test target.
Mismatch: probe, software	Your probe module must be for the same microprocessor as your shell code and executable EL 800 control software file.
Physical device sync failure	This message may have several causes. It indicates that the host computer has not been able to establish communications with the emulator.
	Check to see that the cables are attached, that emulator power is on.
	Make sure the baud rate is the same on the base unit and in the Communications Configuration Window. (page 2-21)
	Check that your cable is attached to the same com port on your PC as you have specified in the Communications Configuration Window. (page 2-21)
	Check your IRQ setting. The IRQ on your PC should match what you specified in the Communications Configuration Window. (page 2-21)
Boot Failed; Communications Link Not Found; Configuration Database Not Found; No Configuration File	One of your required setup files was not found. Check the list of files on page 2-15, and make sure that all are in the same directory as the *???*.exe file. These should all be either in your current directory or in your search path.
Configuration database file not found	Your *???*.cfg file was not found. Check the list of files on page 2-15, and make sure that all are in your search path.

If after checking and correcting the above conditions, you cannot establish connections between your PC and the emulator, please call Applied Microsystems Corporation at (800) 426-3925 (in WA, call (206) 882-2000).

Changing Port, Baud Rate and IRQ Info

To change the port or baud rate in the control software, start the software, and press <esc> to clear any error messages that appear. Press <return> from the Cover Window to get to the Main Menu, and then **C** to go to the Configuration window.

Type **C** to get to the Communications sub-window. Use the arrow keys to go to the Port Name choice. Make sure the Port Name, Baud Rate and IRQ are set appropriately for your PC. The baud rate must match what is set on the switch on the left side of the base unit of the EL 800 (see page 2-9).

Exit the control software:

<esc>	exit Configuration/Communications Window
:X	exit EL 800 control software
<return>	confirm exit

Then start the EL 800 control software again (**EL***???*). If the emulator and control software are communicating properly, the hardware/software Cover Window should have all the hardware information.

Using Resident Device Drivers

If com1: is used for the EL 800, we suggest that you do not use com2: (IRQ3) for Ethernet/NFS or other hardware drivers. Instead, use com4: (IRQ7) for NFS/Ethernet.

Table of Contents

Tutorial: 64180, Z80

TUTORIAL: 64180, Z80

This section provides a short tutorial to help familiarize you with the many powerful features of the EL 800.

Overview of Tutorial

The tutorial follows a hypothetical, but typical sequence you might follow when using your EL 800 to debug software before target hardware is available. As such, it requires either the use of Overlay Memory or a target with available RAM. The tutorial can be used with either the Z80 or 64180 version of the EL 800.

This tutorial guides you through:

- the basics of accessing the windows and commands
- downloading and viewing the demonstration code
- single stepping through the code
- using the Basic Breakpoint System
- using the Advanced Event System

Demonstration code is provided on the distribution disk, and is installed in the same directory as the executable control software by the INSTALL program.

Please note the following conventions:

bold type	Bold type indicates you should type the item as shown.
<key>	Angle brackets are used to enclose single key strokes such as **<space>**, **<return>** and **<PgUp>**.

Getting Started

The tutorial assumes one of two hardware configurations:

1. *Test target, Overlay Module*
   You should already have set up and installed your EL 800 hardware, including one Overlay Module, and have plugged your probe tip into the self-test target.
2. *Target RAM*
   You should already have set up and installed your EL 800 hardware without an Overlay Module, and have plugged your probe tip into a target with RAM space available. You must add the appropriate offset to all addresses specified in this tutorial.

To begin the tutorial, follow these steps.

1. Turn the emulator's power on.
2. Invoke the EL 800 control software, by typing **EL***???* (where *???* is either Z80 or 64180, depending on which probe module you are using).
3. You'll see a Cover Window which shows the current software revisions and lists the installed hardware modules. If you see any error messages, please see Section 2, and make sure you have successfully established communication between your EL 800 software and emulator.
4. Press **<return>** to go to the Main Menu.

Choosing any item on the Main Menu opens up a window and provides access to a group of related tasks. The system is designed so that several windows can be opened simultaneously.

Choosing Windows

To choose a window, use the arrow keys to highlight the name of the window and press return. To return from a window, press **<esc>**.

At the bottom of the screen is a *prompt*, which guides you through the available commands. This line is also where you enter commands to the emulator.

The Code File

If you are doing stand-alone emulation tasks (tasks performed without a target system), code must exist in the emulator's overlay memory. Typically, this code is downloaded from your host computer to the overlay memory.

If you are using your own target RAM, download the code from your host computer to the target RAM.

Mapping to Overlay

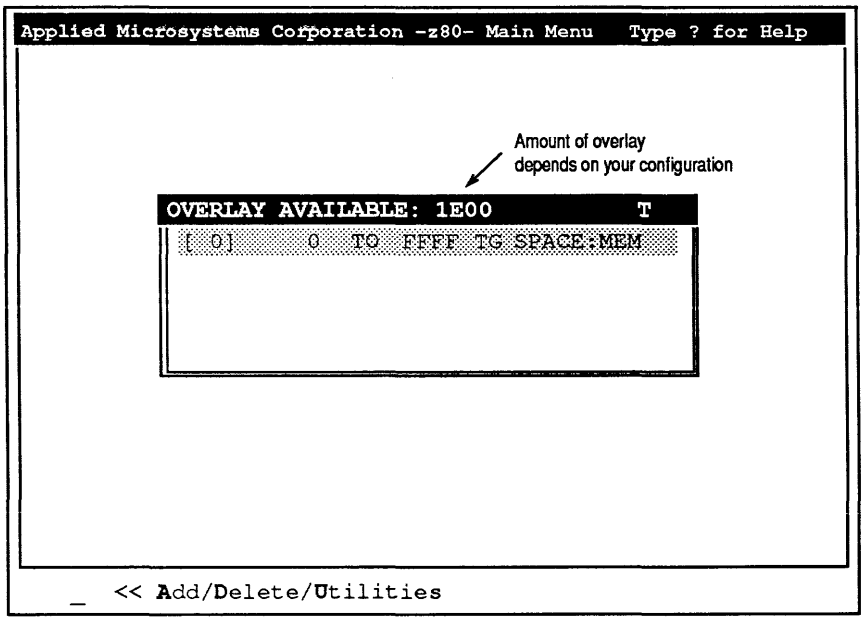
When code is downloaded to the emulator system, its destination can be either RAM on the target circuit that you have designed, or RAM in the emulator's overlay. For this tutorial, if you are using the self-test target, your desired destination is emulator overlay RAM. If you are using your target, the destination is your RAM.

To ensure that a download operation loads code where you want it, the address range of the code must be mapped either to overlay or to the target.

If you're using your target RAM, you don't need to map your code space, since the default is to map all available memory to target.

To map overlay to run in the self-test target, type **:O** to display the Overlay Window. You should see Figure 3-1 on your screen.

Figure 3-1. Overlay Window



All the tutorial code lies between addresses 0000 and 1040. (Note: if using target RAM, remember to add the offset for the location of your RAM).

To map this address to overlay, type **A**. The command line at the bottom of you screen should change to read:

Add << from (addr) , to (addr) [; type]

This tells you that the emulator is going to add a mapping specification, and that it expects the first entry to be the beginning address of the area mapped, terminated by a comma. Next, the emulator expects you to enter the ending address of the area mapped, terminated by a semicolon, and then it expects type information.

Type **0000** followed by a comma (,) and you should see the following command line:

Add 0000 to _ << to(addr) ; type

(If you make a mistake entering the address or any other information on the command line, use the **<backspace>** key to remove instructions, one at a time, from the command line. If you backspace all the way to the first character of the command

line, you can even remove the Add command.)

As you can see, your comma was replaced by the word "to", which is the natural language meaning of the comma. The list of necessary actions in the command line no longer displays the action you just performed (**0000** and **,**).

Type **1040;**. The new command line is:

Add 0000 to 1040; _ << Target/readWrite/Readonly/Illegal

If you're using the self-test target, you want to map your code space to overlay and not to your target circuit's RAM. There are three choices for mapping to overlay RAM: readWrite, Readonly and Illegal. In this case, you should define the new code space as read/write memory, which is selected by typing "W" (the upper case letter in "readWrite"). Type **W**. The command line now reads:

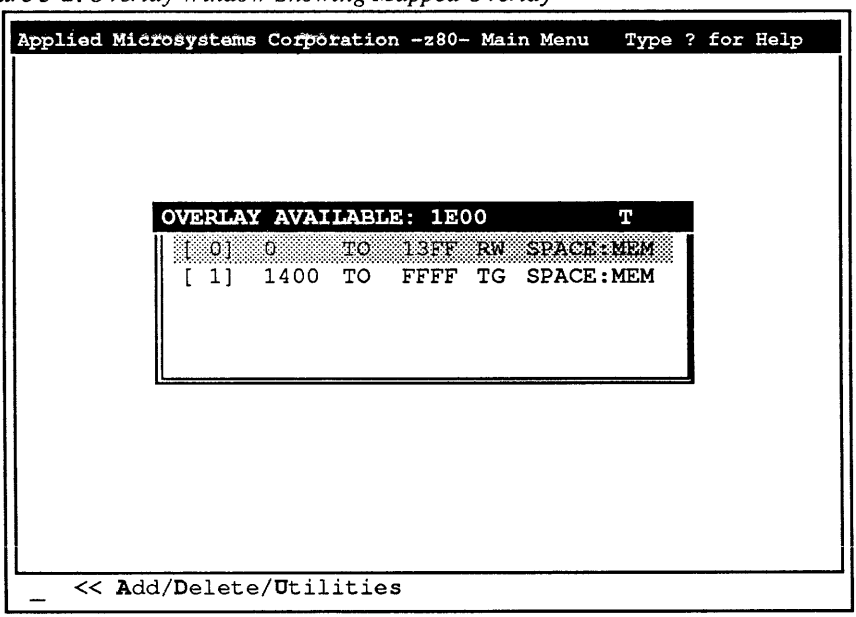
Add 0000 to 1040; readWrite _ << insert wait state(s) Y/N

You will not need wait states for this tutorial, so type **N**. Press **<return>** to confirm your mapping instruction.

(Note that the mapping conditionals "readWrite" and "no wait states" are defaults, and simply pressing **<return>** after entering the address range would achieve the same results as above.)

The Add/Delete/Utilities command line should reappear, and you should see Figure 3-2 on your screen.

Figure 3-2. Overlay Window Showing Mapped Overlay



Note that the upper address (13FF) of your newly-defined address space is slightly larger than the value you entered (1040). This is because the mapper controls memory in 1-Kbyte blocks, and will increase your entry to the next multiple of 1K.

Note also that the default mapping entry (originally 0000 to FFFF to target) has had your newly-mapped overlay space subtracted from it. Target space now starts where your overlay space ends, and extends to FFFF. This prevents you from mapping one memory space to two different RAMs.

Downloading the Code File

Now that you have defined the memory space for the code, you can download the code file.

Type a colon (:) to bring up the following command line:

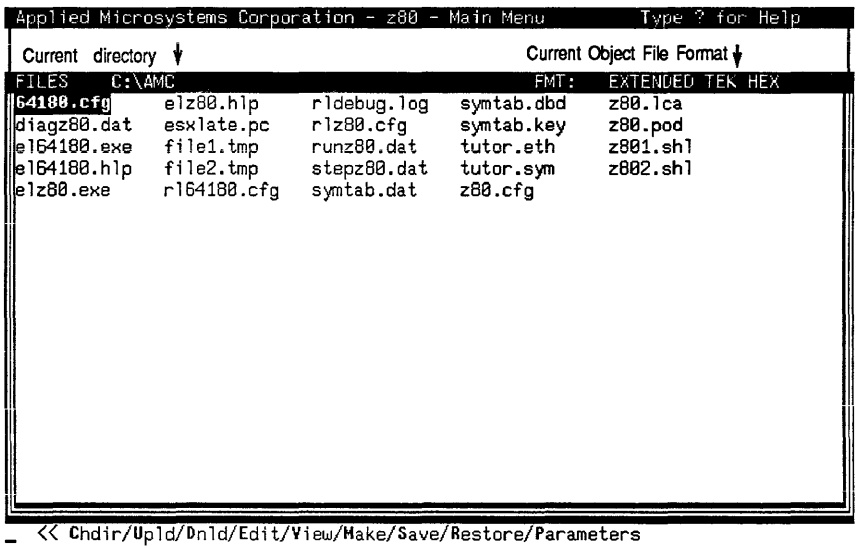
:_ <<A/B/C/D/E/F/M/O/R/S/T/W/X

This tells you, in an abbreviated form, that you may now enter any of the windows

shown in the Main Menu without actually returning to the Main Menu.

Since you need to do some file manipulation, type **F**, and you will enter the File Access Window, shown in Figure 3-3.

Figure 3-3. File Access Window



Notice that the screen displays a listing of the current directory. In this listing, you will find our tutorial code file TUTOR.ETH. The .ETH extension indicates that the file is stored in Extended Tek Hex format. This file includes code and symbols.

To download this file, type **D**. Your command line now reads :

Download _ << enter filename

To enter the requested file name, type **tutor.eth** and **<return>** (or you could move the cursor to the filename using the arrow keys, and then press **<return>**. The small status window in the screen's upper right will confirm that the emulator is downloading TUTOR.ETH. Your tutorial code is now located in overlay RAM, from 0000h to 1040h.

Examining Code in RAM

Now that the code is downloaded to overlay RAM or target RAM, you can verify that your code is in fact where you think it is.

There are several ways to do this. If you don't mind disassembling hex bytes in your head, you could examine your code by using the memory window (:M) to examine 0000h to 1040h. The Assembler/Disassembler Window provides an easier way to view your code.

Type **:A** to use the Assembler Window. You will be asked for a starting address. Although your code occupies 0000h to 1040h, the interesting code starts at address 1000h. Type **1000** and **<return>**. The Assembler Window should appear on your screen as in Figure 3-4.

Figure 3-4. Assembler Window Showing Memory Disassembly

```
Applied Microsystems Corporation -z80- Main Menu    Type ? for Help

ASSEMBLER                    SPACE: MEM
ZERO:       1000    3E00      LD    A,00
            1002    210F11    LD    HL,B2END [090F]
            1005    0610      LD    B,10
ZLOOP:      1007    77        LD    (HL),A
            1008    2B        DEC   HL
            1009    05        DEC   B
            100A    C20810    JP    NZ,ZLOOP [1007]
LOAD:       100D    3E07      LD    A,07
            100F    210711    LD    HL,B1END [0907]
            1012    0604      LD    B,04
            1014    08        EX    AF,AF'
            1015    D9        EXX
            1016    3E00      LD    A,00
            1018    210011    LD    HL,B1ST [0900]
            101B    08        EX    AF,AF'
            101C    D9        EXX
LLOOP:      101D    77        LD    (HL),A

_ << enter line to be assembled
```

In the leftmost column of your screen are the symbols ZERO:, ZLOOP:, and LOAD:. The four digits in the next column to the right are the addresses of the first byte in the instruction. In the next column you can see the hex bytes of the instruction. The mnemonic is in the next column, with the operands in the rightmost column. When a

label is used in the operands, the value of that label is shown to the right of the operands.

Understanding the Code File

Before you can perform meaningful emulation tasks with a piece of code, it is important to understand the structure and function of that code. This section explains the code file TUTOR.ETH.

A flowchart for TUTOR.ETH is shown in Figure 3-5, a listing is shown in Figure 3-6, and a functional description follows.

Figure 3-5. Flow Chart for TUTOR.ETH

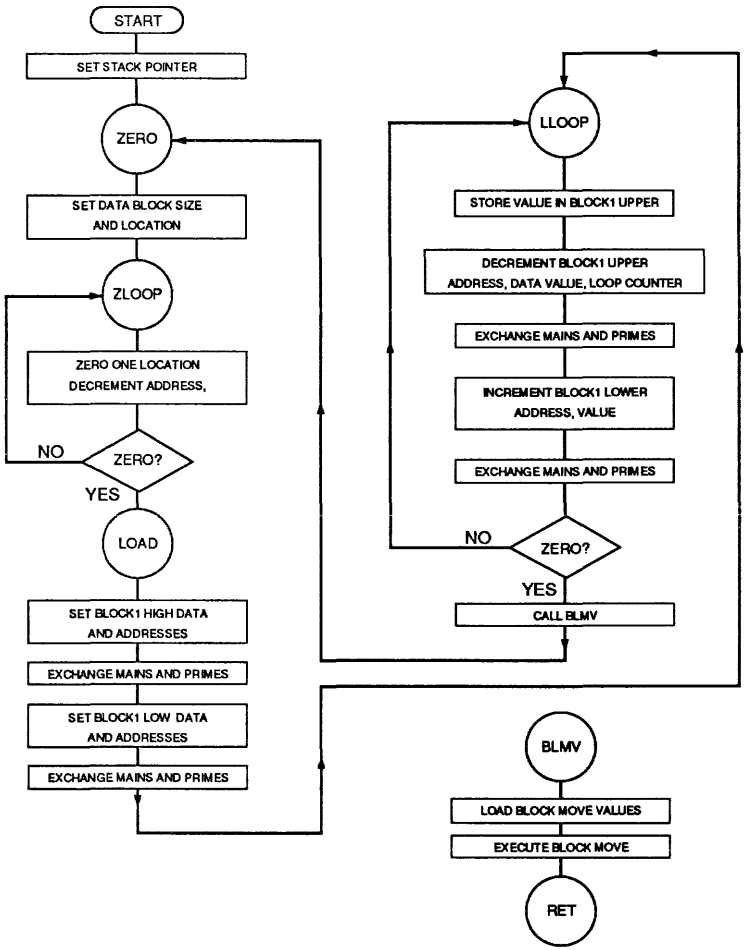


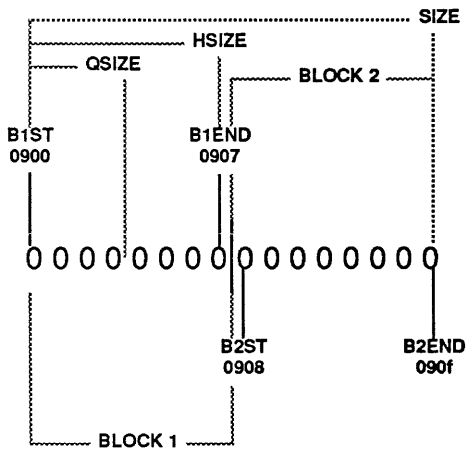
Figure 3-6. Listing for TUTOR.ETH

```
ERR  LINE  ADDR  OBJ

       1                                LIST   B,G
       2  0000
       3                          NAME   tutor
       4  0000
       5  0900                    blst   equ    0900h
       6  0907                    blend  equ    0907h
       7  0908                    b2st   equ    0908h
       8  090F                    b2end  equ    090fh
       9  0010                    size   equ    10h
      10  0008                    hsize  equ    08h
      11  0004                    qsize  equ    04h
      12  0007                    upper  equ    07h
      13  0000
      14                                org    0000h
      15  0000  31 00 09          start: ld     sp,0900h
      16  0003  C3 00 10                 jp     zero
      17                                org    1000h
      18  1000  3E 00             zero:  ld     a,0h
      19  1002  21 0F 09                 ld     hl,b2end
      20  1005  06 10                    ld     b,size
      21  1007  77                zloop: ld     (hl),a
      22  1008  2B                       dec    hl
      23  1009  05                       dec    b
      24  100A  C2 07 10                 jp     nz,zloop
      25  100D  3E 07             load:  ld     a,upper
      26  100F  21 07 09                 ld hl,blend
      27  1012  06 04                    ld     b,qsize
      28  1014  08                       ex     af,af'
      29  1015  D9                       exx
      30  1016  3E 00                    ld     a,0h
      31  1018  21 00 09                 ld     hl,blst
      32  101B  08                       ex     af,af'
      33  101C  D9                       exx
      34  101D  77                lloop: ld     (hl),a
      35  101E  2B                       dec    hl
      36  101F  3D                       dec    a
      37  1020  05                       dec    b
      38  1021  08                       ex     af,af'
      39  1022  D9                       exx
      40  1023  77                       ld     (hl),a
      41  1024  23                       inc    hl
      42  1025  3C                       inc    a
      43  1026  08                       ex     af,af'
      44  1027  D9                       exx
      45  1028  C2 1D 10                 jp     nz,lloop
      46  102B  CD 31 10                 call   blmv
      47  102E  C3 00 10                 jp     zero
      48  1031  01 08 00          blmv:  ld     bc,hsize
      49  1034  11 08 09                 ld     de,b2st
      50  1037  21 00 09                 ld     hl,blst
      51  103A  ED B0                    ldir
      52  103C  C9                       ret
      53  103D                           end

                                SYMBOL TABLE
B1END          0907     BLST           0900     B2END          090F
B2ST           0908     BLMV           1031     HSIZE          0008
LLOOP          101D     LOAD           100D     MEMORY       M 0000
NARG           0000     QSIZE          0004     SIZE           0010
STACK        S 0000     START          0000     UPPER          0007
ZERO           1000     ZLOOP          1007
```

The first line of the program, labeled START, is at address 0000h, the reset vector. The stack pointer is set here and the program jumps to 1000h, label ZERO.

The routine at ZERO sets the size and location of a data block in memory, as shown in Figure 3-7. This data block is divided into two sub-blocks, block1 and block2. The location of these blocks in memory are determined by the values of B1ST (Block 1 start), B1END (Block 1 end), B2ST (Block 2 start), and B2END (Block 2 end). SIZE is the number of bytes in the two blocks, HSIZE (Half-size) is the number of bytes in each block, and QSIZE (Quarter size) is one half of HSIZE.

Figure 3-7. Data Block



The routine at ZLOOP is executed SIZE times, writing a zero into each location in Block 1 and Block 2.

The routine at LOAD places address, data, and loop-count information in the main registers to write at the upper end of the upper half of block1. The main registers are then switched with the primes. Address and data information are placed in the prime registers to write at the lower end of the lower half of block1. Mains and primes are switched again.

The routine at LLOOP loads one block1 upper half byte and decrements the address and data value, while counting loop iterations. Mains and primes are switched and the block 1 lower half byte is loaded. Address and data are incremented and mains and primes are switched. This loop is executed QSIZE times, completing the block1 load.

BLMV (Block move) is called. This loads the register with values to move the contents of block1 to block2, using the LDIR instruction.

After the return from BLMV, the program jumps to ZERO and the procedure starts over.

Single-Stepping Through Code

The simplest control you can exercise over your code is to execute one line of code and stop execution. While stopped, you can examine trace, memory, registers, and/or event status to determine the effects of executing the instruction.

Let's use the single-step command to examine some code execution, and, at the same time, learn some of the emulator's features. To do this, type **:E** to enter the Emulate Window. Note that the use of :E from any window calls up the Emulate Window precisely as if you had used <esc> to return to the Main Menu and had then pressed E to enter the Emulate Window. This is called "tunnelling".

Since you will want to set your program counter to the beginning address of the tutorial code, and since you will also want to observe changes in the PC and other registers, enter **:R** to use the Register Window. If you are using the EL 800 Z80 probe module, the registers are divided into two groups, Primary and Alternate, for convenient display. The Primary window is shown in Figure 3-8, and the Alternate window is composed of only the "prime" registers.

If you are using the EL 800 HD64180 probe module, the registers are also divided into groups for convenient display, but six groups are used. The first two groups are identical to the above-described Z80 register groups. The newly-started Register Window displays the contents of the Primary registers. If you wish to examine or change the contents of registers not in the Primary group, use the arrow keys to highlight "PRIMARY". Use **<space>** to toggle through the register groups. For this demonstration, leave the Register Window displaying Primary.

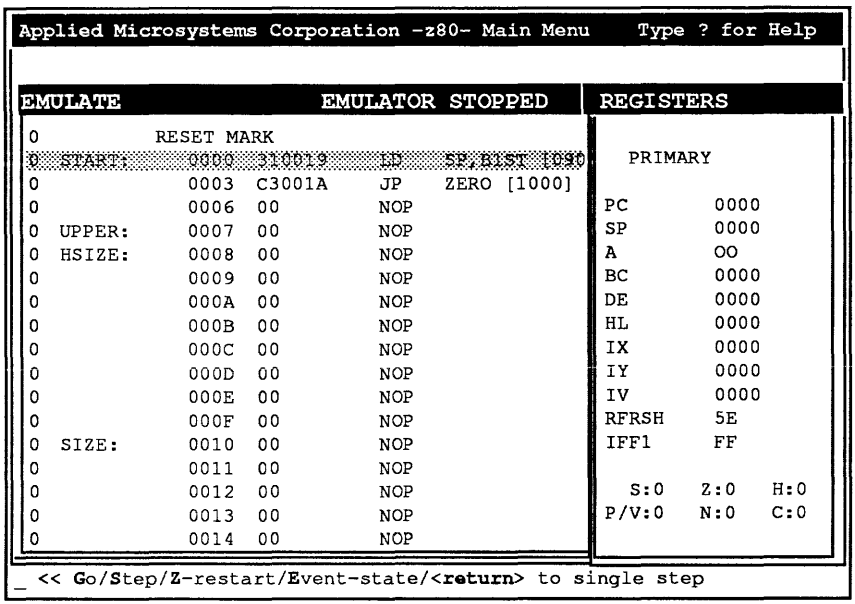
Note that while you are displaying two windows, Emulate and Register, only the Register Window is surrounded by double lines. The double lines means the Register Window is active, and can be controlled by you, while the Emulate Window is inactive, and cannot be controlled by you.

You should see Figure 3-8 on your screen.

Move the cursor down until the PC register is highlighted and enter **0**. The PC value should now be 0000.

Remove the Register Window by entering **<escape>** and notice the double lines now surrounding the Emulate Window.

Figure 3-8. Emulate Window With Register Window



The highlighted line of code in the Emulate Window will be executed if you choose the "**<return>** to single step" option, as shown in the command line at the bottom of the Emulate Window. Since this instruction loads a value of 900 into the stack pointer, you would expect pressing **<return>** to change the value of the stack pointer in the Registers Window to 900. Further, you would expect the highlight to advance to the next instruction, the JP ZERO at address 0003. You might also expect the value of the PC to change to that of the next instruction, 0003.

Press **<return>** to execute this instruction, and **<:R>** to call up the Registers Window to observe these changes. After you have examined the Register Window, enter **<escape>** to return to the Emulate Window. Although this invoking and removing of the Registers Window is extra effort right now, you will soon learn a method for keeping the Registers Window available with an active Emulate Window.

Single-stepping through the next few instructions should allow you to monitor several kinds of change. You will first see the PC value change to the value of the instruction ZERO:, where you will be able to watch the accumulator set to zero. You will see the instruction LD B,10 load the value 10 into the B register.

Press **<return>** to execute a single instruction, then **:R** to observe the effects of each instruction on the registers, then **<escape>** to remove the Registers Window, until the highlight is over the instruction:

```
ZLOOP: LD (HL),A
```

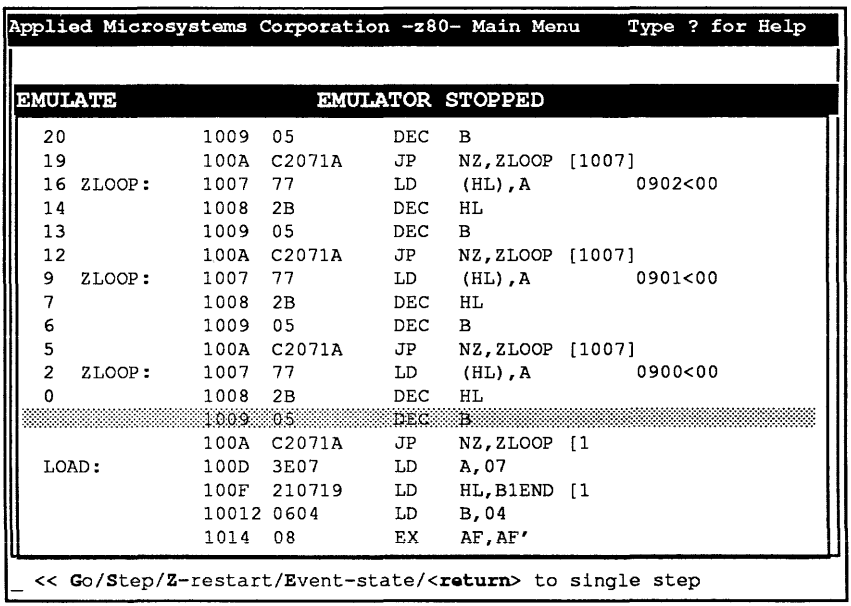
At this point you have entered the loop ZLOOP, which will be executed 10H, or 16D times. There are 4 instructions in ZLOOP:, ending with a jump-if-not-zero to ZLOOP:. This means that you will spend the next 64 single-steps in ZLOOP:, and nothing exciting will happen until you are almost finished with this loop.

The Step command is designed for this situation, allowing you to specify how many steps you would like to make with one instruction. Notice the status message in the top window border showing that the emulator is stepping.

Enter **S62 <return>**. The 62 single steps take about a minute to complete. Notice the status message in the top window border showing that the emulator is stepping.

When the stepping is complete, the status message in the window border will read "EMULATOR STOPPED" and you'll see Figure 3-9.

Figure 3-9. Emulate Window, Showing Result of the Step Command



There is a great deal of information contained in your Emulate Window. Learning to locate and interpret this information will make your emulation and development work more efficient. To that end, let's examine this screen.

Use the **<PgUp>**, **<PgDn>**, ↑ and ↓ keys to position the relative address number (those numbers in the extreme left-hand column in the Emulate Window) so that 120 is at the upper edge of your window. You will have to **<PgUp>** several times to get to this location. This line should be labeled START:, and represents the first instruction you entered in this session.

The contents of your screen, from relative address 120 to the uppermost relative address 0 (the line immediately above the highlighted cursor) are a record of every instruction you have executed since this session began. This record is the trace, a powerful emulation tool.

Examination of the trace can reveal much about the execution of your code. If you examine relative address line 112 in the trace, you will see that the constant B2END was loaded into register pair HL, and that the value of B2END is 90F.

You can see on line 107 that the accumulator contents were loaded into the memory location pointed to by the contents of pair HL, and that the absolute values of this instruction were 00, loaded into address 90F. This is shown in Figure 3-10.

If you page down this display until the highlighted line 0 is the 6th line up from the screen's bottom (the display will stop scrolling here), you will be back to the screen of Figure 3-9, and can derive more information from the display.

Figure 3-10. Emulate Window, Showing Trace

```
Applied Microsystems Corporation -z80- Main Menu    Type ? for Help

EMULATE                    EMULATOR STOPPED
 120  START:   0000  310009     LD    SP,B1ST  [0900]
 117           0003  C30010     JP    ZERO  [1000]
 114  ZERO:    1000  3E00       LD    A,00
 112           1002  210F09     LD    HL,B2END  [090F]
 109           1005  0610       LD    B,10
 107  ZLOOP:   1007  77         LD    (HL),A          090F<00
 105           1008  2B         DEC   HL
 104           1009  05         DEC   B
 103           100A  C20710     JP    NZ,ZLOOP  [1007]
 100  ZLOOP:   1007  77         LD    (HL),A          090E<00
  98           1008  2B         DEC   HL
  97           1009  05         DEC   B
  96           100A  C20710     JP    NZ,ZLOOP  [1007]
  93  ZLOOP:   1007  77         LD    (HL),A          090D<00
  91           1008  2B         DEC   HL
  90           1009  05         DEC   B
  89           100A  C20710     JP    NZ,ZLOOP  [1007]
  86  ZLOOP:   1007  77         LD    (HL),A          090C<00

_ << Go/Step/Z-restart/Event-state/<return> to single step
```

The instructions above the highlighted instruction, on lines 0 to 20, represent instructions actually executed.

Some information is also available about the future of code execution. The highlighted line, for instance, will be the next instruction to be executed.

The five instructions under the highlight are simply the next instructions in memory, and are not necessarily to be executed next, or even in the displayed order. If the instruction below the highlight, JP NZ,ZLOOP, causes a jump to ZLOOP, the instruction immediately below it, LD A,07, will not be executed now.

This raises an interesting question: will the instruction JP NZ,ZLOOP cause a jump

to ZLOOP when it is executed?

Loading Custom Window Configuration

To answer this question, it would be convenient to watch the Register window while emulating. Unfortunately, when the Emulate window is called up, it covers the Registers window.

Fortunately, the EL 800 provides a window-customizing utility, which allows you to define the size and location of active windows. This re-definition may be saved with the :Files/Save/Windows command. In fact, a good solution to the present problem has been worked out and saved in the file TUTOR.WIN.

To use the window configuration of TUTOR.WIN, type **:F** to enter the Files window, **R W TUTOR.WIN <return>** to restore the TUTOR.WIN window configuration. When a window is opened now, its size and location will be defined by TUTOR.WIN.

To invoke a viewable Register window and an active Emulate window, enter **:R :E**.

Now, if you were to enter **<return>** to single-step and execute the instruction DEC B, you should be able to see the contents of register B in the Registers Window, presently 01, decremented to 00. Because of this, you should also see the Z (zero) flag at the bottom of the Register window set. The Z flag should display "1" after the single-step.

Enter **<return>** to single-step. Because you can see zero flag is set, you should not expect the highlighted instruction, the next instruction to be executed, to cause a jump to ZLOOP.

Enter **<return>** and verify this.

Setting Basic Breakpoints

The Basic Breakpoint System requires that at least one Overlay Module be installed. Please skip this section if you do not have an Overlay module.

A basic breakpoint is a way of halting code execution, or breaking, when a specified address is encountered.

It would be useful if we could examine the execution of our demonstration code TUTOR.ETH at the end of each iteration of the memory-loading loop LLOOP, and examine the progress of the loading process. The basic breakpoint is a convenient tool for this.

Enter **:B** to use the Break/Event Summary Window, from which you can use the Basic Breakpoint System or the Advanced Events System. Enter **B** to select the Basic Breakpoints Window.

Our goal is to set a breakpoint which will become active, causing a break, when the last line of code in LLOOP is executed. This can be described as a READ at address 1028.

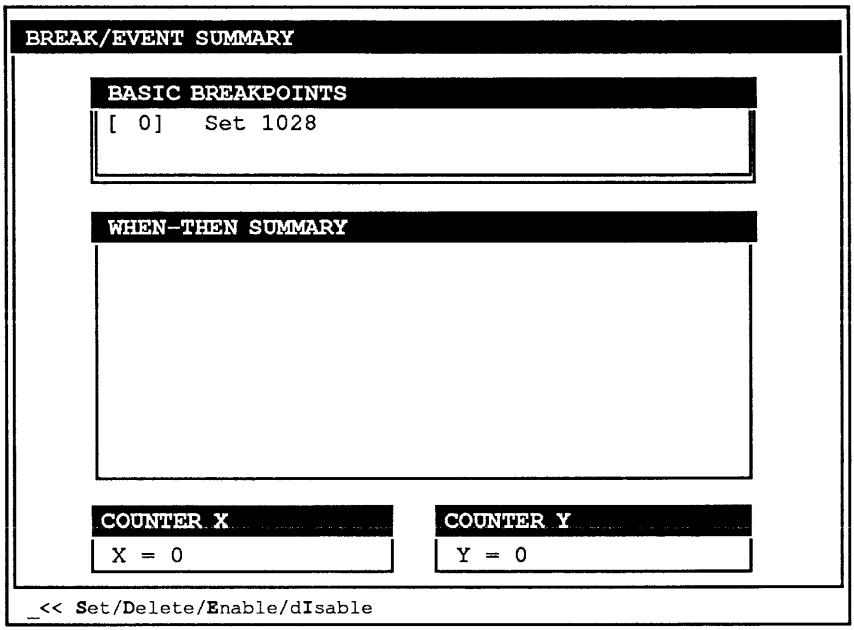
Enter **S** for SET, and observe the following command line:

Set _ << (addr) [, (to addr)]

The Basic Breakpoint System is now expecting an address.

Enter **1028** to enter the address, **<return>** to enter the breakpoint, and **<escape>** to deselect the Basic Breakpoints window. You should see Figure 3-11. The emulator will now break each time a read or write is performed at address 1028.

Figure 3-11. Basic Breakpoint Window With One Breakpoint Set

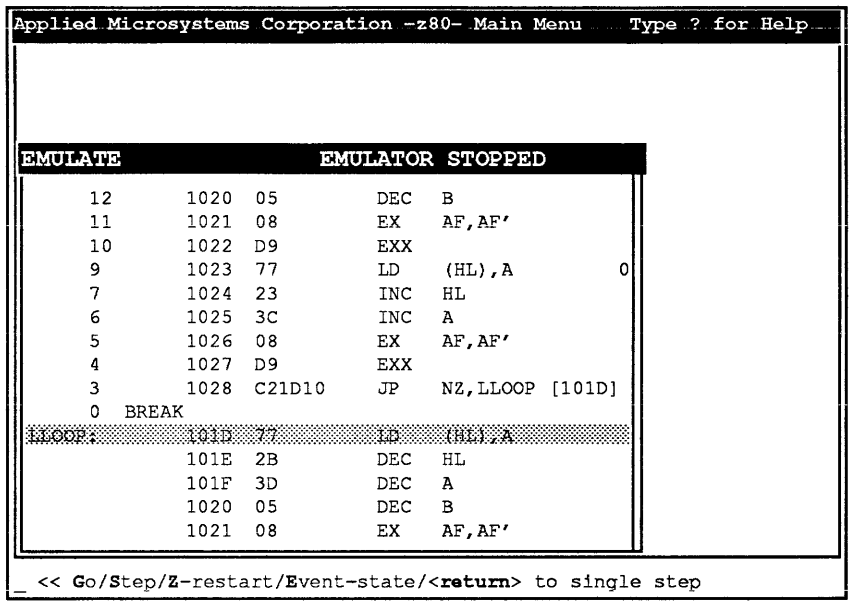


Enter **:E** to use the Emulate Window, and notice that it is exactly as you left it.

Enter **G** to GO, and **<return>** to indicate that you want to run your code without changing the starting PC value.

When the new trace is written to the Emulate Window, you can see that the instruction at 1028 was read and executed, and emulation was broken. The highlighted cursor is on the next instruction, LLOOP, as shown in Figure 3-12.

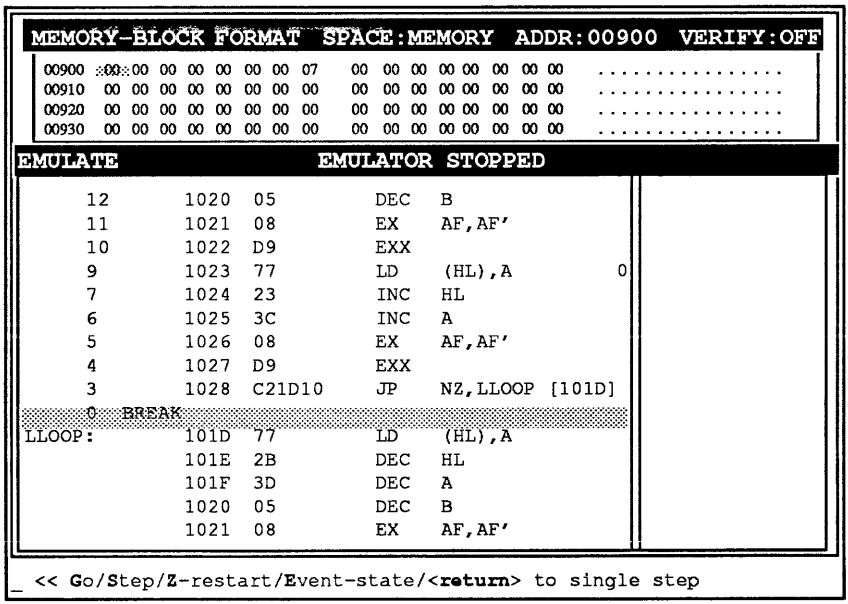
Figure 3-12. Emulate Window after Breaking at Breakpoint



The function of the loop LLOOP is to load one high and one low byte pair into Block 1, which extends from address 900 through 907. For verification, let's examine the memory location Block 1.

Enter **:M** to use the Memory Window, enter **900** to indicate which part of memory you wish to examine, and enter **<return>** to confirm the entry. You should see Figure 3-13. The Memory window is another window resized by restoring the file TUTOR.WIN.

Figure 3-13. Memory Mode Window



You can see that 07 has been written to 907, and you can predict that 00 has been written to 900. It seems logical to predict that another go-until-breakpoint operation should write another pair of bytes into Block 1.

Let's verify this. Enter **:E** to bring up an active Emulate window. Enter **G <return>** to run to your breakpoint and examine the Memory window. Another pair of bytes has been written to Block 1: 06 is written to 906, and 01 to 901.

This procedure can be repeated as often as you like.

Clear Basic Breakpoints

At this point, let's clear the Basic Breakpoint setup so that it doesn't get in our way when we set up an advanced event. Type **<escape>** to close the Memory Window, **<escape>** to close the Emulate Window and you'll see the Break/Event Summary Window again. Type **B** to select the Basic Breakpoints window, **D** to delete, and either **0** or , *****, and **<return>** to confirm the clearing of breakpoint 0. Enter **<escape>** to deselect the Basic Breakpoints window.

Setting Advanced Events

While the Basic Breakpoint System exercise you just completed is sufficiently powerful to handle many situations, many users will need greater power and flexibility. The Advanced Event System lets you break and perform other actions based on a wide variety of conditions.

For example, you might experience difficulties with the block-move functions performed by the LDIR instruction in the routine BLMV in your test program TUTOR.ETH. It would be helpful to examine the read and write cycles of the LDIR instruction, noting particularly the data on the bus, while viewing the contents of the source and destination locations in memory.

Further, you might want to gather this information for only one iteration of the BLMV routine and then halt code execution to examine your information. In addition, it would be nice if the only information gathered (traced) were the BLMV information, so that you would not need to wade through execution information irrelevant to your problem.

The Advanced Events System provides tools to qualify execution and trace this specifically. See Sections 6 and 7 for further information.

Advanced Event System Overview

Before going into the details of setting up this example, this section provides a conceptual overview, and describes the steps involved in using the Advanced Event System.

The Advanced Event System is based on WHEN-THEN statements:

WHEN *conditions* THEN *actions*

The conditions are logical combinations of address, data, status, count limits, and trigger inputs. The actions are combinations of breaking emulation, trigger out, trace control, counter control, and switching states. There are four independent sets of condition inputs, one set for each of the four state windows.

There are six steps to setting up the Advanced Event System:

1. Conceptualize the progression of generalized program conditions you wish to isolate, dividing these conditions into groups (states) whose contained

conditions will be simultaneously active.

2. Decide how each condition will be expressed, and what events will be caused.
3. Load the condition comparators with all conditions necessary for each state.
4. Write the WHEN-THEN statements, making sure that the state-to-state transition statements are included.
5. Reset or clear the state of the Advanced Events System to a beginning state.
6. Execute your program, using the Go command in the Emulate window (:EG)

Conceptualizing the Set Up

While developing WHEN-THEN statements to solve an emulation problem, it can be handy to write your WHEN-THEN statements in a kind of pseudocode. Once you have expressed the WHEN-THEN statements in pseudocode, it is easy to enter the statements into the Advanced Event State windows.

When-Then Description Pass One

For example, to solve the above problem of verifying possible block-move problems in BLMV, the WHEN-THEN statements might read:

WHEN the first line of code is executed, THEN turn the trace off.
OR
WHEN the routine BLMV is called, THEN turn trace on.
OR
WHEN the last instruction in the routine BLMV is read, turn trace off and halt code execution (break).

When-Then Description Pass Two

The next step in describing the WHEN-THEN statements should include some values. For example, the first line of code executed in TUTOR.ETH is the LD SP,900 at address 00. There are several ways to describe this line of code uniquely, but the most straightforward description is the line of code at address 00. This can be described by the following:

WHEN the operation is a fetch and the address value is 00,
THEN turn the trace off.

The second WHEN-THEN statement should should turn the trace on when the program calls the routine BLMV. Since this line of code, at line 102B, is the only CALL instruction in the test code, and since the hex value of the CALL instruction is CD, this can be described by the following:

WHEN the data is CD and the operation is a fetch,
THEN turn trace on.

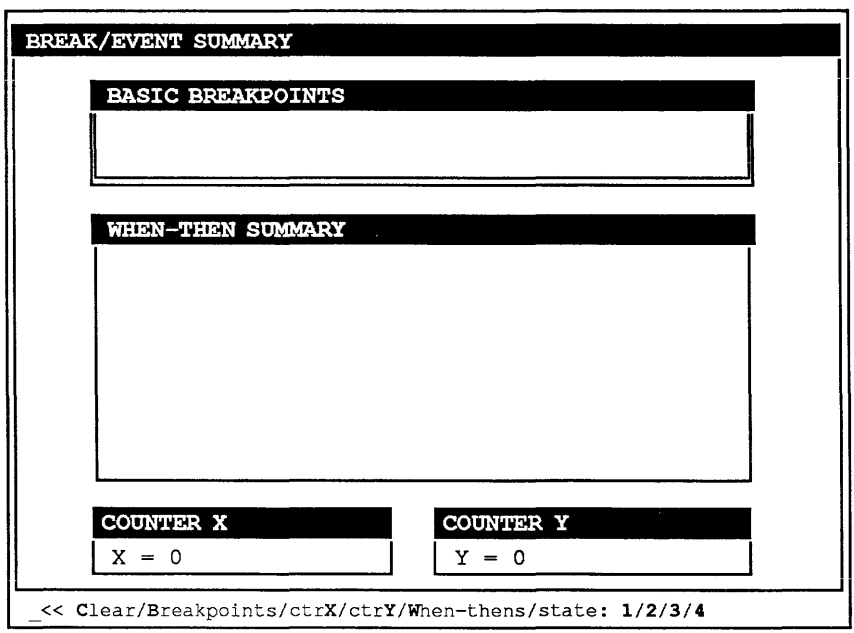
For the third WHEN-THEN statement, we will use the fact that the last line of the routine BLMV is the only RET instruction in the program. The hex value of an RET instruction is C9. This condition can be described as follows:

WHEN the data is C9 and the operation is a fetch
THEN turn the trace off and break.

Entering When-Then Statements and Comparators

Enter **:B** to use the Break/Event Summary Window, shown in Figure 3-14.

Figure 3-14. Break/Event Summary Window



Note that no WHEN-THEN statements are currently present in the WHEN-THEN expression summary.

Since we decided that our proposed WHEN-THEN statements would be active in state 1, enter **1** to use the Event System State 1 Window.

To implement our first WHEN-THEN statement, "WHEN the operation is a fetch, and the address value is 0000, THEN turn trace off", it is necessary to understand that the emulator determines that a fetch has been recognized when a status comparator set to recognize a fetch has been satisfied. Similarly, an address value of 0000 is recognized by satisfying an address comparator set to recognize the hex word 0000.

Since your State 1 Window has address comparators A and B, data comparators E and F, and status comparators R and S, you have a way to enter your conditions for WHEN-THEN statement #1.

Enter **A** to use the Address Comparator A sub-window. Note the command line at the bottom of your screen. It prompts you to "Set/Delete" a comparator entry; enter **S** to set. Again on the command line, you are asked for an address value (16-bit, if you have not selected a 20-bit address bus in the Emulator Configuration window). Enter **0000** and **<return>** to set address comparator A to recognize the hex word 0000. Enter **<escape>** to deselect Address Comparator A.

Now enter **R** to use the Status Comparator R sub-window, and **S** to set an entry and view your choices. The status condition you want is fetch1, which has no upper case letters, but does include a number, 1. Enter **1** and notice that fetch1 has been selected, as indicated by the highlighting of the word "fetch1" in this sub-window. Enter **<return>** to enter "fetch1" into the comparator, and enter **<escape>** to exit this sub-window and set status comparator R to fetch1.

Enter **W** to select the When-Then Statements sub-window.

Now you are ready to enter the WHEN-THEN statement, which will appear in the command line at the bottom of your screen.

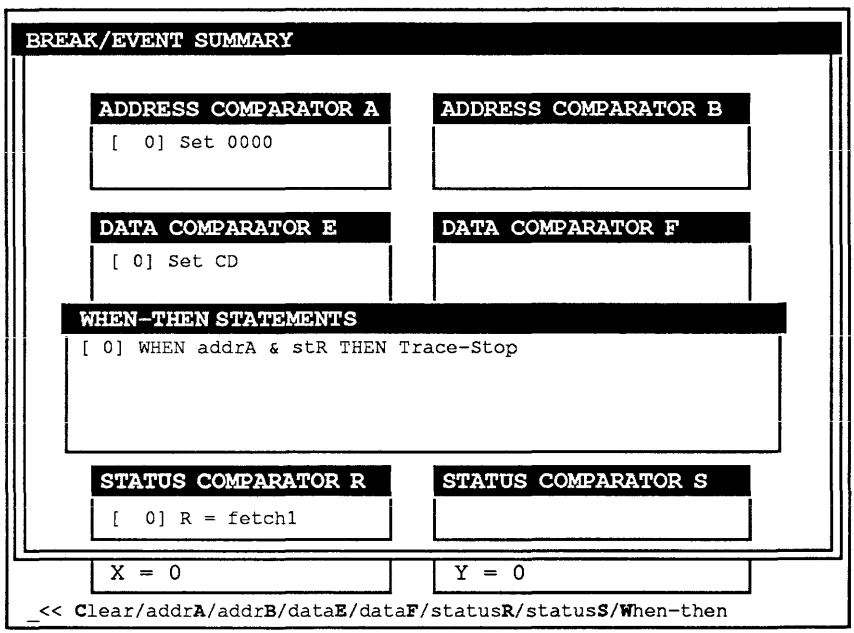
Enter **W** and see "WHEN" appear. Enter **A** and see "WHEN addrA" in your command line. Enter (with no leading or trailing spaces) **&** to indicate a logical ANDing process, **R** to specify status comparator R, **T** to specify THEN. You should see "WHEN addrA & stR THEN" in your command line.

The rest of your command line lists the options open to you, including "Trace". Enter

T to select trace, and notice your command line options again. Because you want this WHEN-THEN statement to turn trace off, enter **S**. You should see "WHEN addrA & stR THEN Trace-Stop". This is the desired statement, so enter **<return>** to accept the statement, and **<escape>** to deselect the When-Then Statements sub-window.

With statement #1 set up, your State 1 screen should appear as shown in Figure 3-15.

Figure 3-15. Advanced Event State 1



For WHEN-THEN statement #2, you need to have the hex value of the CALL instruction in a data comparator, meaning data comparator E.

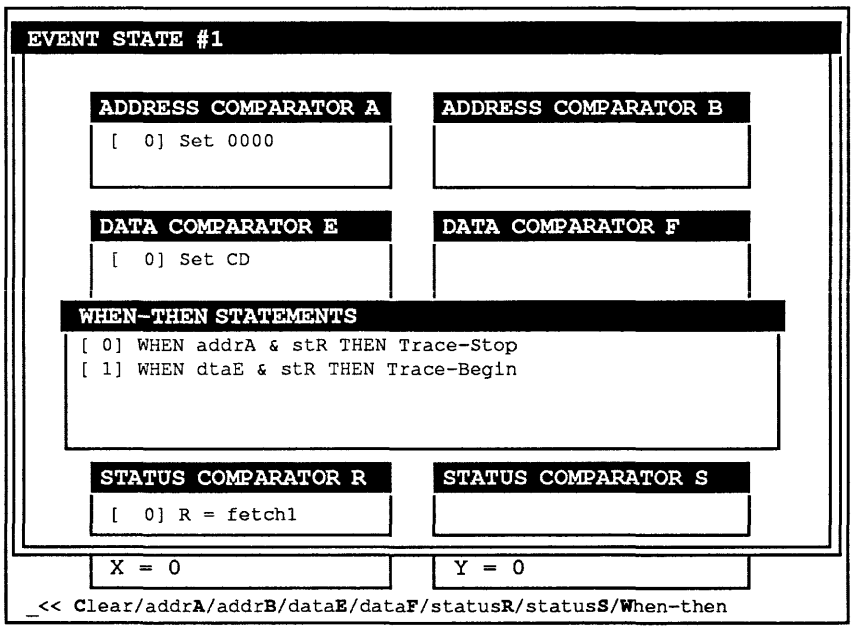
Enter **E** to select the Data Comparator E sub-window, and **S** to set a comparator entry. Enter **CD <return>** to load data comparator E with CD, and **<escape>** to deselect comparator E.

As before, enter the WHEN-THEN statement. The key stroke sequence is **W W E & R T T B <return> <escape>**. You should now have, in the main State 1 window, the following:

WHEN dtaE & stR THEN Trace-Begin

With statements one and two set up, your State 1 screen should appear as shown in Figure 3-16.

Figure 3-16. State 1 Window with Two When-Then Statements



Statement #3 is created by loading data comparator F with the value C9 (the hex value of an RET instruction), and writing the statement "WHEN dataF and stR THEN Trace-Stop Break". This causes a trace-off and a break every time an RET instruction is executed. As in the previous statement, this is a special case, with only one RET occurring in the test code.

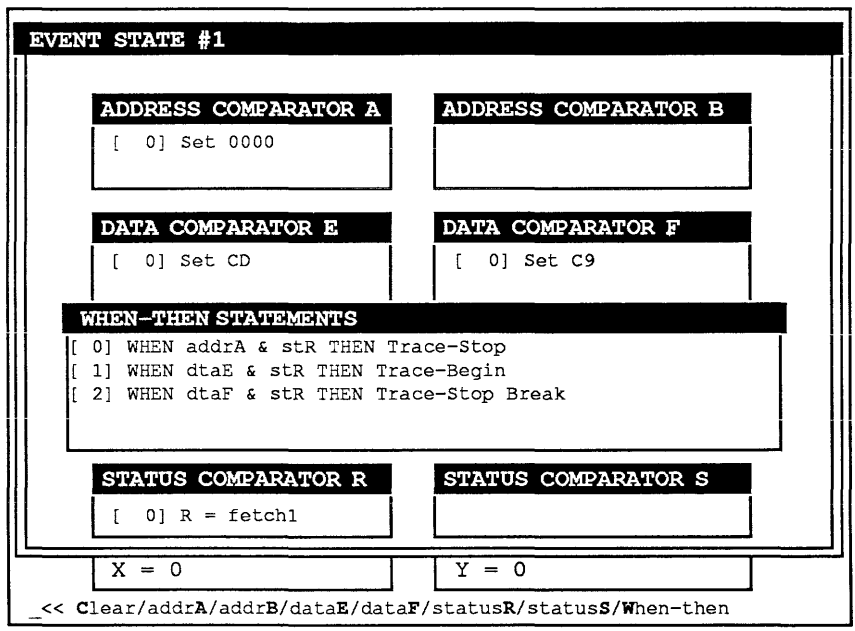
The key stroke sequence to load the comparator is **F S C9 <return> <escape>**. Enter this.

The key stroke sequence to select the When-Then window, create the WHEN-THEN statement, and deselect the window is: **W W F & R T T S B <return> <escape>**. Enter this. You should see the following line in the State 1 window.

WHEN dataF & stR THEN Trace-Stop Break

With statements one, two and three set up, your State 1 screen should appear as shown in Figure 3-17.

Figure 3-17. State 1 Window with Three When-Then Statements



This is the last of the WHEN-THEN statements, so enter **<escape>** to exit the State 1 Window and return to the Break/Event Summary Window.

You have now set up your emulation events with everything needed to satisfy your original conditions. Notice that the Summary window shows your three WHEN-THEN statements.

Emulating With the Advanced Events System Setup

When you execute code with the setup you have just entered, you will want to observe the effects of execution as interpreted by the trace function of the Emulate window, and you will also want to observe the Block 1 and Block 2 locations in memory, using the Watch window.

Invoke the Watch window by entering **:W**.

For convenience of display, you will set up the watch window to watch the memory area from 0900 to 090F in four blocks of four bytes. Blocks of 0900-0903, 0904-0907, 0908-090B, and 090C-090F will be used. To enter these blocks in the Watch window, location, data type, and display format must be defined. (A complete guide to the expression formats used here may be found in Appendix F, "Using Expressions".)

For instance, a pointer to the starting byte of the first watched block might be described as "(byte *)900". Further, you would like to examine four bytes in this block, and display them as hexadecimal integers, described as "4 x". The complete specification would read "(byte *)0900, 4 x".

To enter this specification into the watch window, enter **A** to add a specification. This will open the addition window, which will ask you to enter an expression. Enter the expression described above, for the first watched memory block, **(byte *)900, <space> 4 <space>**. When the statement is complete in the addition window, confirm it and enter it into the Watch window by entering **<return>**.

Now enter the next three memory watch specifications:

A (byte *)904, <space> 4 <space>.
A (byte *)908, <space> 4 <space>.
A (byte *)90C, <space> 4 <space>.

Enter **<escape>** to deselect the Watch window.

Enter **:E** to open the Emulate window.

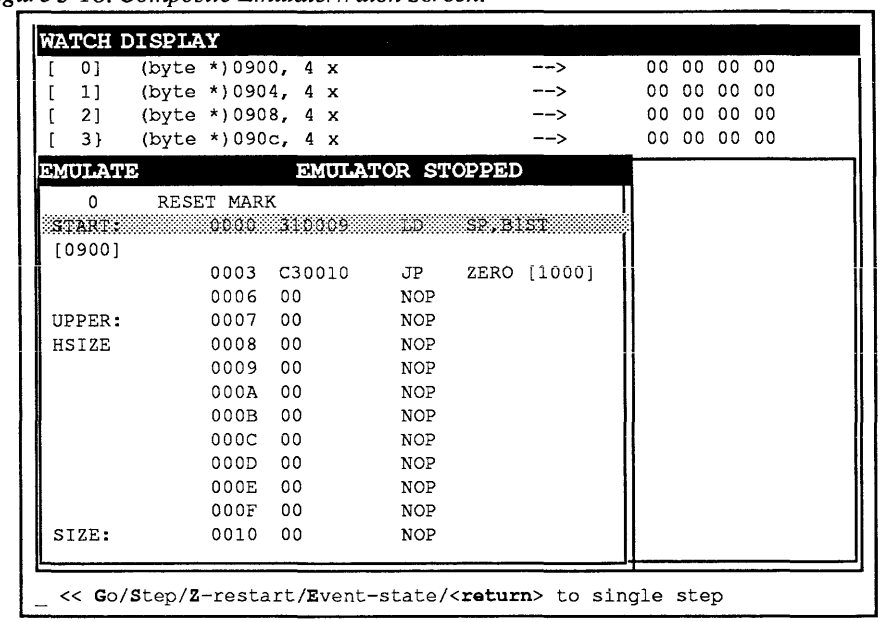
Enter **E** to open the Event State sub-window. This sub-window displays the current state of the event system, and may be used to evaluate your progress through an elaborate emulation experiment.

Note the "<< Clear state variables" message in the command line. "Clear", in this sense, resets all your state variables to a standard starting position, in state 1, with both counters stopped, with trace turned on, and with the counters loaded with the values you entered in the Break/Event Window. In normal operation, it is wise to clear the state variables before beginning emulation.

To clear the state variables and leave the Event State sub-window, enter **C <return> <escape>.**

You should see the screen of Figure 3-18.

Figure 3-18. Composite Emulate/Watch Screen.



Before you begin emulation within your new Advanced Events System, note that the Emulate and Watch windows are sized and located such that both are completely visible. These specifications are from the file TUTOR.WIN, your custom window configuration file. Note also that your Emulate window is "clipped" on the right side, also an artifact of your custom window configuration.

You may recall that the right margin of the Emulate window was moved to uncover the Register window. In the following example, register information will not be needed, but the bus activity information trimmed from the Emulate window will be

needed.

To move the Emulate window's right margin to the screen's right margin, for the purpose of displaying bus activity information, enter **<F1>** and see the command line:

<resize> _ << Arrow keys: Left Right Up Down

Because the Emulate window is the active window (with the double-line margin) it is the Emulate window that will be resized by the arrow keys. Using the → and ← keys, move the right margin of the Emulate window to the right margin of the screen.

Before you begin emulation within your new Advanced Events System constraints, you should Restart the emulator's target processor. Enter **Z <return>G <return>** to Restart and begin execution. Note that the breakpoint will be encountered before the "EMULATION STOPPED" annunciator at the top of the Emulation window is updated to show emulation running.

After running to the breakpoint, you should see the screen of Figure 3-19.

Figure 3-19. Emulate/Memory Screen with Trace.

```
WATCH DISPLAY
[  0]   (byte *)0900, 4 x                  -->       00 01 02 03
[  1]   (byte *)0904, 4 x                  -->       04 05 06 07
[  2]   (byte *)0908, 4 x                  -->       00 01 02 03
[  3]   (byte *)090c, 4 x                  -->       04 05 06 07
EMULATE                EMULATOR STOPPED
   33          103A  EDB0      LDIR              0900>00  0908<00
   29          103A  EDB0      LDIR              0901>01  0909<01
   25          103A  EDB0      LDIR              0902>02  090A<02
   21          103A  EDB0      LDIR              0903>03  090B<03
   17          103A  EDB0      LDIR              0904>04  090C<04
   13          103A  EDB0      LDIR              0905>05  090D<05
    9          103A  EDB0      LDIR              0906>06  090E<06
    5          103A  EDB0      LDIR              0907>07  090F<07
    1          103C  C9        RET
    0    BREAK
               102E  C30010    JP    ZERO [1000]
BLMV:          1031  010800    LD    BC,HSIZE [0008]
               1034  110809    LD    DE,B2ST [0908]
               1037  210009    LD    HL,B1ST [0900]
               103A  EDB0      LDIR
               103C  C9        RET
_ << Go/Step/Z-restart/Event-state/<return> to single step
```

This screen contains the essential information concerning the block-transfer function of routine BLMV. Each of the lines from #33 to #5 shows one pair of read-from-source and write-to-destination cycles of the LDIR instruction. Notice that the address read from or written to is shown at the right side of the Emulate window, as is the data value on the bus during that read or write.

In the four-line Watch window, the contents of the block-move source addresses (0900 to 0907) and the block-move destination addresses (0908 to 090F) are displayed. The values shown in Figure 3-19 are what you would predict.

This brings up a subtle, but important, point. Since every execution of BLMV should leave the Watch window displaying the information in Figure 3-19, it is not possible to know, from this setup, if this iteration of BLMV loaded memory correctly, or if memory has not been changed since the last time BLMV loaded the memory correctly.

The program sets the contents of memory locations 0900 to 090F to 00 before it loads 0900 to 0907 and block-moves that data to 0908 to 090F. If you were to insert an additional break instruction in the Advanced Event System that broke emulation immediately after the watched memory location is set to zero, the Go command would cause the emulator to break alternatively after zeroing, and then after load-and-transfer. This would allow verification of memory modifications at every Go.

Since no trace specifications would be added with the new WHEN-THEN statement, code inside the routine BLMV will still be the only trace acquired.

The Additional When-Then Statement

You need to describe a condition for this breakpoint that occurs after memory zeroing (this occurs in ZLOOP) and before LLOOP, where the memory load process starts. In addition, you have one address comparator left in Advanced Events System State 1.

All that is necessary is to pick an instruction between the end ZLOOP and the body of LOAD, and specify that instruction's address, ANDed with fetch1, and you will have described the necessary condition. The instruction at the label LOAD, LD A,UPPER matches these criteria, and will be used.

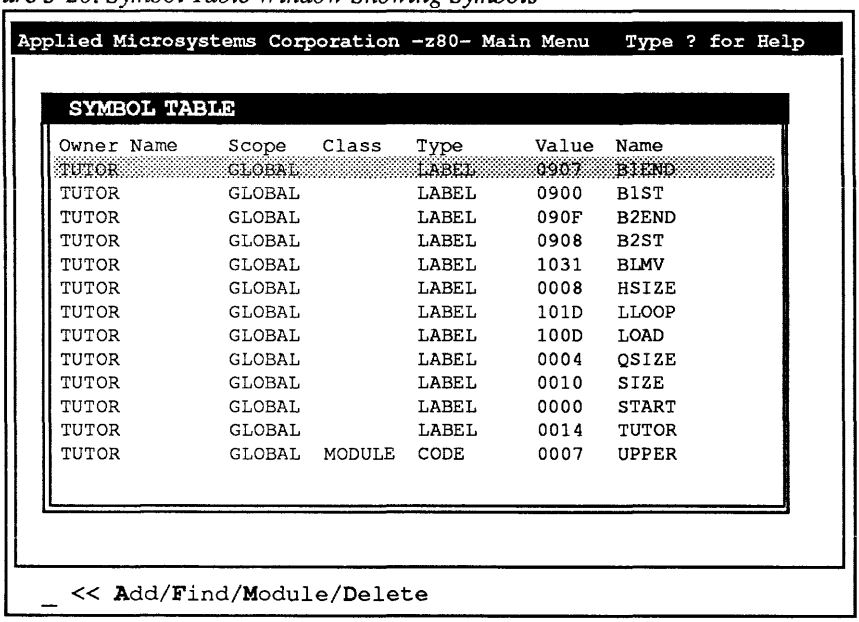
You could load the address of this instruction by entering the address value 100D into address comparator B, but there is an easier and more flexible way to enter this

number.

If, at the appropriate point in assembling the When-Then statement, you enter a space instead of a number, the expression analyzer is started. The expression analyzer allows you to use symbols from your code instead of the numbers those symbols represent. In this case, the starting address of the block-move source-load routine corresponds to the symbol LOAD. Note the upper case letters here, because the expression analyzer is case-sensitive.

If you want to check what symbols are in your symbol table, enter **:S** to display the symbol table. You can see LOAD, with a value of 100D, in Figure 3-20.

Figure 3-20. Symbol Table Window Showing Symbols



Enter **:B1B** to return to the Advanced Event state 1 window, and select the Address Comparator B subwindow.

To load address comparator B with 100D, enter
S <space> LOAD **<return> <return> <escape>.**

To create the When-Then statement, enter **W W B & R T B <return> <escape>.** You should see, as When-Then statement #4, WHEN addrB & stR THEN Break.

Execution and Observation

Go back to the composite Emulate/Watch window by entering **<escape> <escape> <escape>.**

Reset the processor and Go to the first breakpoint (**Z <return> G <return>**).

You should see the screen of Figure 3-21, with the displayed memory locations set to zero, and only the BREAK traced.

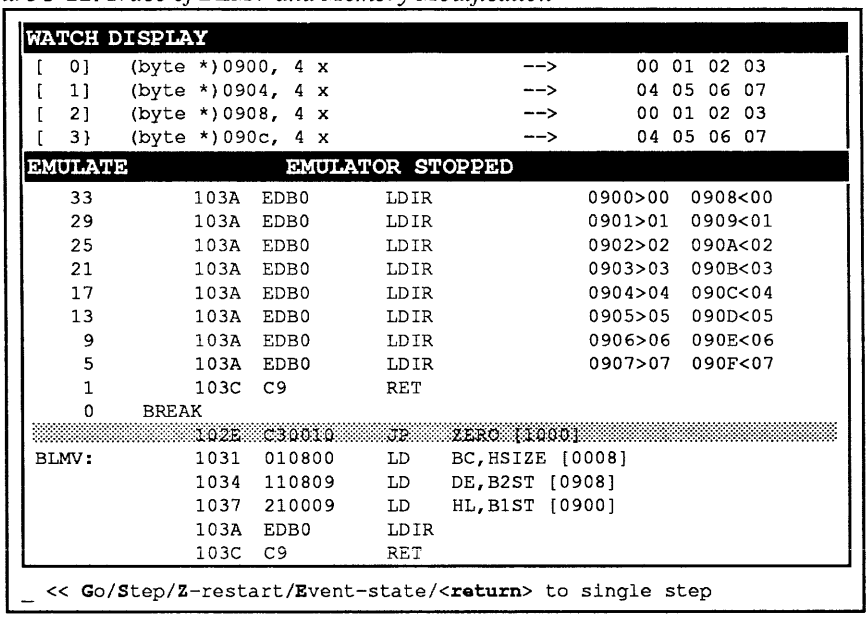
Figure 3-21. Memory Locations Set to Zero

```
WATCH DISPLAY
[  0]   (byte *)0900, 4 x                    -->      00 00 00 00
[  1]   (byte *)0904, 4 x                    -->      00 00 00 00
[  2]   (byte *)0908, 4 x                    -->      00 00 00 00
[  3}   (byte *)090c, 4 x                    -->      00 00 00 00
EMULATE                EMULATOR STOPPED
   27         103A  EDB0       LDIR              0902>02  090A<02
   23         103A  EDB0       LDIR              0903>03  090B<03
   19         103A  EDB0       LDIR              0904>04  090C<04
   15         103A  EDB0       LDIR              0905>05  090D<05
   11         103A  EDB0       LDIR              0906>06  090E<06
    7         103A  EDB0       LDIR              0907>07  090F<07
    3         103C  C9         RET
    2    BREAK
    1  Illegal Instruction
    0    BREAK
              100E  21079      LD    HL,B1END [0907]
              1012  0604       LD    B,04
              1014  08         EX    AF,AF'
              1015  D9         EXX
              1016  3E00       LD    A,00
              1018  210009     LD    HL,B1ST [0900]
_ << Go/Step/Z-restart/Event-state/<return> to single step
```

Go to the next breakpoint (**G <return>**.

You should see the screen of Figure 3-22, which traces the BLMV execution, and displays the modified memory after memory load and block-move.

Figure 3-22. Trace of BLMV and Memory Modification



If you had acquired a great deal of trace, and needed ways to move around in a large trace file, you could go to the Trace window with a :T. While you would see essentially the same information as in the trace portion of the Emulate window, you could request, by line number, a desired line in trace memory and display it in context. In addition, you could examine the trace instructions bus cycle by bus cycle (raw trace).

A More Complex Advanced Events Setup

The Advanced Event System exercise you just finished was designed as an introduction to the Advanced Events System, and did not seriously explore the logical possibilities of the EL 800's Advanced Event System. The following exercise is designed to expose you to more of the system's capabilities.

As you did in the previous exercise, assume a hypothetical problem. Assume that data corruption occurs occasionally on accesses to the memory area Block 1 (0900 to 090f). Assume further that the corruption only affects data bit D2, and that only logical-low values of D2 are corrupted. Also, the problem only occurs after the

target circuit has been "exercised" by many thousands of bus cycles. This problem is so occasional that a large number of memory accesses must be traced to have a reasonable chance of observing the problem, so very selective trace is needed.

Conceptualizing the Setup

As before, a sort of "when-then pseudocode" will be used to define the problem for the Advanced Events System. The development of this definition follows:

Turn the trace off, run code for a long time, then trace some large number of bus cycles where Block 1 memory is accessed and the data read or written has data bit D2 low, then break.

When-Then Description Pass One

State 1:

WHEN the first line of code is executed, THEN turn the trace off.
OR
WHEN the program loops back to the label ZERO THEN
decrement a counter with a large number in it.
OR
WHEN the counter contents are zero THEN change to state 2.

State 2:

WHEN the address is between 0900 and 0907 AND
the data is some value with bit D2 = 0 and the
other bits set to DON'T CARE AND
it's a memory access THEN trace one bus cycle AND
decrement a different counter with another large
number in it.
OR
WHEN the second counter contents are zero THEN break.

When-Then Statement Pass Two

State 1:

WHEN the operation is a fetch and the address is 0000
THEN turn trace off.

WHEN the operation is a fetch and the address is label ZERO
THEN decrement counter X.

WHEN counter X equals zero
THEN change to state 2.

State 2:

WHEN the address is between 0900 and 0907 AND
the data is 00 with a DON'T CARE mask of 04 AND
memory request (MREQ) is active
THEN trace one bus cycle and decrement counter Y

WHEN counter Y equals zero
THEN break.

Entering When-Then Statements and Comparators

To use the Break/Event window, enter **:B**.

To clear existing advanced events conditions, enter **C <return>**.

To enter State 1 window, enter **1**.

To set status comparator R to detect the fetch1 condition for the first when-then statement, enter **R S 1 <return> <escape>**.

To set address comparator A to detect address 00 for the first when-then statement, enter **A S 0000 <return> <escape>**.

Enter the first statement, "WHEN addrA and stR THEN Trace-Stop" (**W W A & R T T S <return> <escape>**).

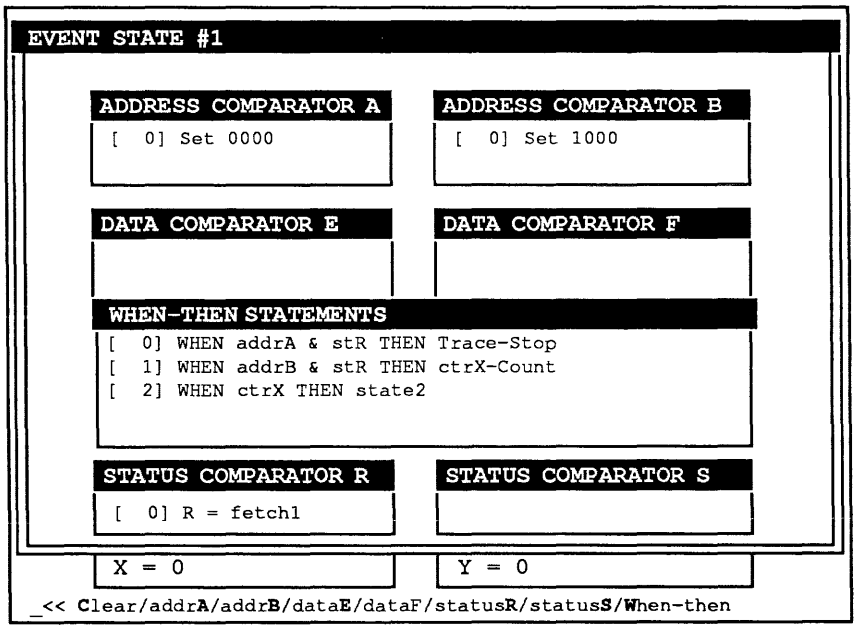
To set address comparator B to detect the address label ZERO for the second when-then statement, enter **B S <space> ZERO <return> <return> <escape>**.

Enter the second statement, "WHEN addrB & stR THEN ctrX-Count" (**W W B & R T X C <return>**).

Enter the third statement, "WHEN ctrX THEN st2" (**W W X T 2 <return> <escape>**).

You should see your three When-Then statements and comparator contents in the State 1 as shown in Figure 3-23.

Figure 3-23. Completed State 1 Window



To set the State 2 conditions, move to the State 2 window (**<escape> 2**).

To set address comparator A to detect the range 900-907 for the first statement in State 2, enter **A S 900 , 907 <return> <escape>**.

To set data comparator E to detect the byte 00 with a DON'T CARE mask of 04 for the first statement in State 2, enter **E S 00 ; 04 <return> <escape>**.

To set status comparator R to recognize the MREQ status for the first statement in State 2, enter **R S M <return> <escape>**.

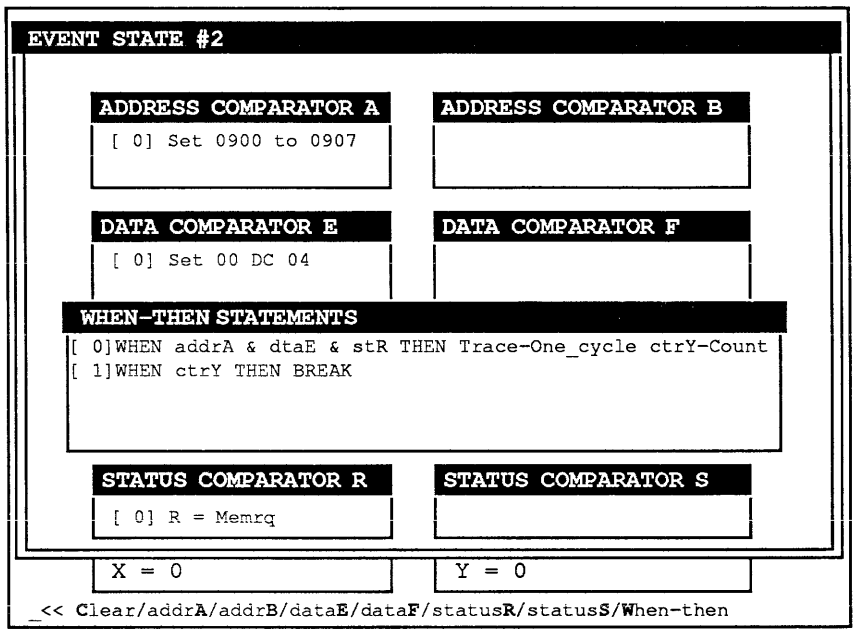
Enter the first State 2 statement, " WHEN addrA & dtaE & stR THEN Trace-One_cycle ctrY-Count" (**W W A & E & R T T O Y C <return>**).

Enter the second State 2 statement, "WHEN ctrY THEN Break" (**W W Y T B <return> <escape>**).

You should see your two When-Then statements and comparator contents as in

Figure 3-24.

Figure 3-24. Completed State 2 Window



Loading the Counters

Before you can execute code with this setup, you must first load the Advanced Event system's counters. We decided earlier that Counter X must be loaded with some very large number, and the counter is a 16-bit counter, so you will load the Counter X with the largest number possible, FFFF.

Counter Y will determine how many bus cycles will be traced, so 3FFF seems (arbitrarily) sufficient.

To load these values, go to the Break/Event window (**<escape>**) and load Counter X with FFFF (**X FFFF <return> <return>**. Load counter Y with 3FFF (**Y 3FFF <return> <return>**.

At the bottom of the window, you should see "X = ffff", and "Y = 3fff".

Emulating with the Advanced Events System Set Up

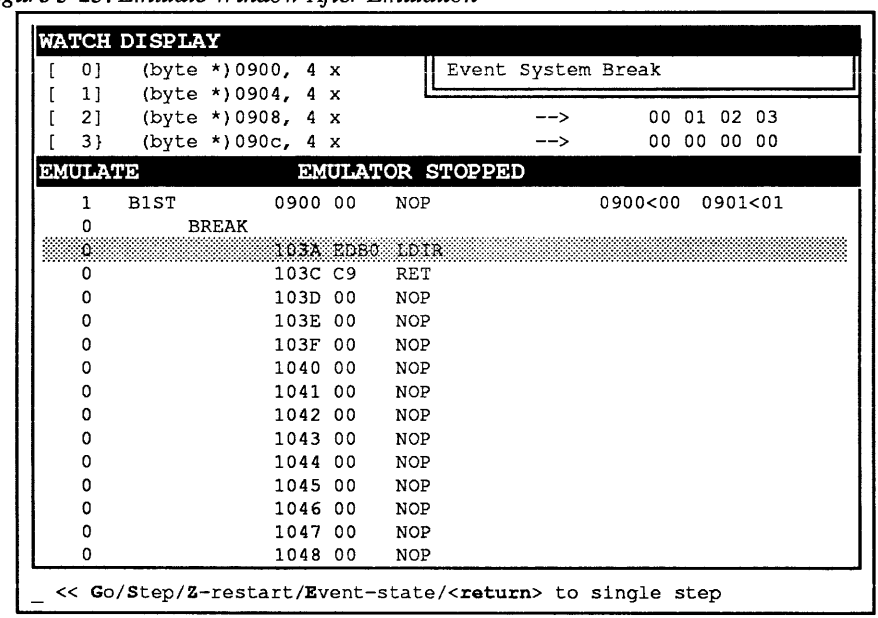
Enter **:E** to open the Emulate Window, and **E** to open the Event State sub-window.

Enter **C <return> <escape>** to initialize the state variables, and leave the Event State sub-window.

To begin emulation, Restart and Go (**Z <return> G <return>**.) Notice the "EMULATOR RUNNING" annunciator at the top of the Emulate window. Since the emulator has to execute 64K (FFFF) iterations of the test program, execution will take about 30 seconds, with a 2.5MHz clock.

When the emulator halts, notice the "Event System Break" window in the upper right, shown in Figure 3-25.

Figure 3-25. Emulate Window After Emulation



Examine the information in your Emulate Window. Is it what you expected? When you ran the previous experiment, the Emulate Window displayed, from top to bottom, a traced record of the code execution, the next instruction to be executed with a highlight, and a sequential code listing.

You appear to have no trace, and have instead only a "BREAK" statement, seen in Figure 3-25.

Whenever the results of an emulation experiment are unexpected, or appear to be wrong, it is time to re-examine the code and the specifications in the Break/Event Window.

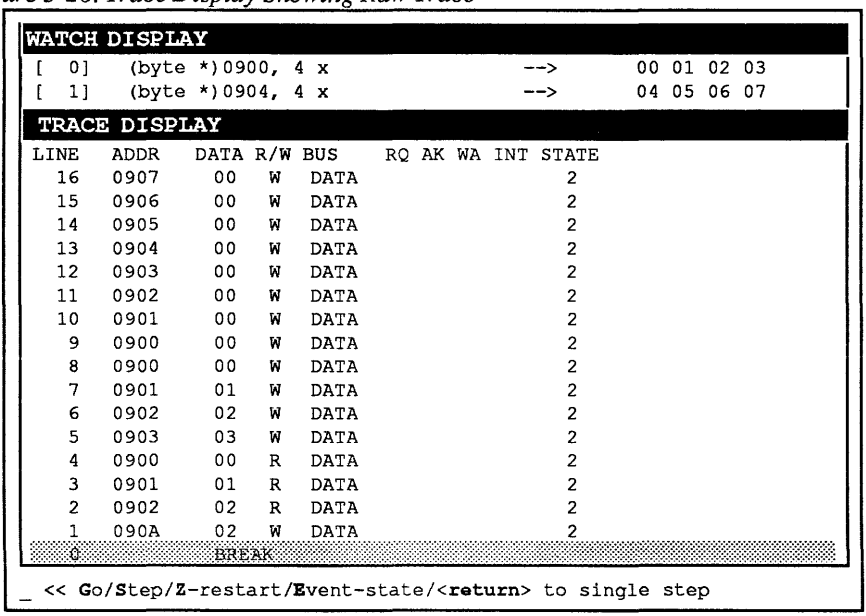
In this particular case, the problem seems to be absence of trace, and yet you entered the statement "WHEN addrA and dtaE and stR THEN Trace-One_cycle ctrY-Count". The address you installed in address comparator A certainly is the address range of Block 1 and 2, and examination of memory at this address range verifies that the memory reads and writes have been successfully performed.

The solution is in interpretation; the emulator did, in fact, trace one cycle at each access to the memory space Block1 and 2. The emulation screen displays only disassembled trace, raw data converted into its equivalent instructions, complete with mnemonics and labels. What you have traced, however, is single bus cycles extracted from multi-bus-cycle instructions; these isolated cycles are not capable of being disassembled.

This does not mean that you can't examine this trace; it means only that the traced information is not available in the Emulate Window. Viewing the Trace Window will allow you to examine the history of your code's execution.

Enter **:T** to use the Trace Display. You will see Figure 3-26.

Figure 3-26. Trace Display Showing Raw Trace



```
WATCH DISPLAY
[  0]   (byte *)0900, 4 x                    -->       00 01 02 03
[  1]   (byte *)0904, 4 x                    -->       04 05 06 07
TRACE DISPLAY
LINE   ADDR   DATA R/W BUS    RQ AK WA INT STATE
  16   0907     00   W  DATA                  2
  15   0906     00   W  DATA                  2
  14   0905     00   W  DATA                  2
  13   0904     00   W  DATA                  2
  12   0903     00   W  DATA                  2
  11   0902     00   W  DATA                  2
  10   0901     00   W  DATA                  2
   9   0900     00   W  DATA                  2
   8   0900     00   W  DATA                  2
   7   0901     01   W  DATA                  2
   6   0902     02   W  DATA                  2
   5   0903     03   W  DATA                  2
   4   0900     00   R  DATA                  2
   3   0901     01   R  DATA                  2
   2   0902     02   R  DATA                  2
   1   090A     02   W  DATA                  2
   0            BREAK
_ << Go/Step/Z-restart/Event-state/<return> to single step
```

The leftmost column is the trace line number column, which contains the position of that line in the trace buffer, with the first line having the highest line number, and the last line, where emulation broke, having the number 0. The next columns contain the contents of the address bus and data bus during the traced cycle, and whether the transaction was a read or write.

The column titled BUS interprets status information and, for the Z80, describes whether the traced cycle was a memory request, fetch, interrupt acknowledge, data, or I/O request. If the emulator is configured for the HD64180 processor, the BUS column may also display DMA and sleep cycle conditions.

The next four columns display status bit information, indicating if the BUSRQ-, BUSACK-, WAIT-, or INT- line have been asserted in this bus cycle.

The rightmost column in the raw trace display shows the break/event state of the emulator during the traced bus cycle. Note that, in this demonstration, all the traced cycles are in break/event state #2, as you might expect from your break/event setup.

Examining Trace

Before you use the cursor and page control keys to scroll through your trace, consider what you expect to find traced. The only cycles traced will be accesses to memory space Block 1. These cycles will be of three types.

One type is the set-to-zero cycles of ZLOOP, where all eight cycles of each ZLOOP are traced because D2 is always low in 00. Trace lines 9-16 are examples of this type of cycle. Notice that all these cycles are writes.

Another type is the loading of values from 00 to 07 in locations 900 to 907, where only those cycles with data values having D2 = 0 (less than 04) are traced. Trace lines 5-8 are examples of this, and are all writes.

The third type of cycle is the read portion of the LDIR instruction, and, as above, will only be traced for data values less than 04. Trace lines 2-4 are examples of this, and are all reads.

Summary of Tutorial

In this tutorial, you've gone through a typical debug sequence: downloading a sample piece of code, looking at the code in overlay memory, stepping through the code, and using the Basic Breakpoint and Advanced Event Systems to isolate potential errors. This should give you a good idea of how the EL 800 can be used for debugging software and hardware.

Take a few minutes now and look through the windows the tutorial did not cover. The unused windows were the Configuration window and the Diagnostics window. Type **:C** to enter the Configuration window and look through the communications, emulator setup and system processes windows.

Type **:D** to enter and examine the Diagnostics window. These RAM tests, scopeloops, and reset pulses are to aid you with your hardware debugging tasks.

To go back and view any other window, type **:** for the list of window letters, then the letter representing that window.

For more detail on any window and information on resizing and moving windows, see the alphabetical descriptions of each window in Section 6, Operation.

SECTION 4

Table of Contents

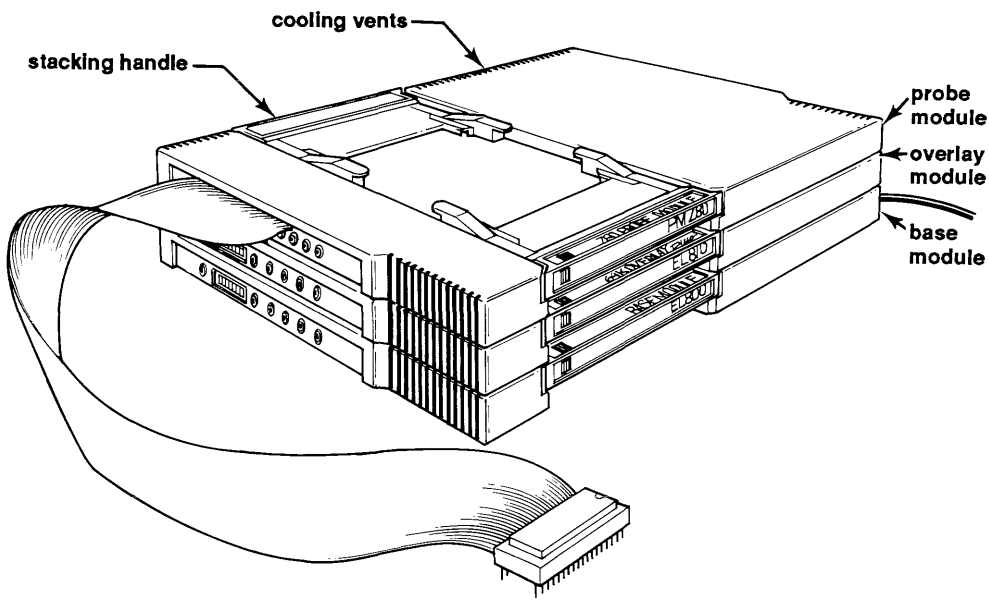
Hardware

HARDWARE

The hardware for the EL 800 consists of a base module, a probe module, a choice of optional modules, a power supply and miscellaneous accessories. The modules stack on top of each other, so no connection cables are needed.

This section describes each module, explains how to stack and unstack the modules, and provides information on maintenance of the EL 800.

Figure 4-1. EL 800 Typical Configuration



The components of the EL 800 are the following:

— Base unit module

— Probe module and attached probe tip

— Optional overlay memory modules

— *EL 800 User's Manual*

— RS-232 cable (9 to 25 pin)

— RS-232 AT adapter cable (25 pin to 9 pin)

— 2 floppy disks containing emulator control software

— 5 wires with clips: one black (ground wire), four red (for connecting to other equipment)

— Power supply: 110 vac or 220 vac
  - 110 vac power supply includes power cord
  - 220 vac power supply (international) does not include power cord

— Vertical support stand

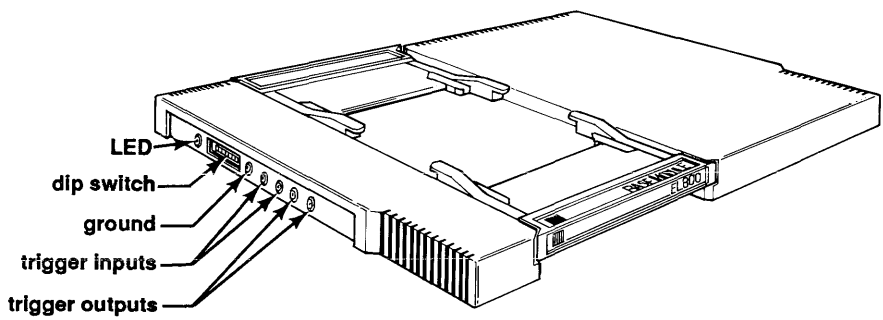
— Optional accessories:
  - carrying case

Base Module

The base module provides the 8K x 48 bit trace memory and the Advanced Event System, and is the main controller for the EL 800. It has connections for trigger input and output, and a serial port. The base module has a battery backup for RAM, so you don't need to download code each time you power up.

There are 5 input/output connectors, 8 dip switches and a green LED on the left side of the base module.

Figure 4-2. The Base Module: Left Side



The connectors, LED's and dip switches on the left side are described below:

Ground	The ground should always be connected to your target via the black clip wire provided. If the target and emulator have different ground potentials, the emulator CPU in the probe tip may be damaged. To avoid this problem, you should connect your target ground to the base module ground before you plug the probe tip into your target.
Trigger Output	Two external trigger outputs enable you to look at signals or signal duration on an oscilloscope or use other capture equipment. These are labeled "OUT 1" and "OUT 2." The red clip wires provided can be used for these connections.

Trigger Input

Two external trigger inputs provide input from logic outside the microprocessor. These are labeled "IN 1" and "IN 2." The red clip wires provided can be used for these connections.

Dip Switches

The dip switches on the base unit control the baud rate of the serial port. Only the first three are used; leave the others set to 0. The setting on this must match the setting in the Configuration/Communications window in the EL 800 control software. We recommend using the EL 800 at 19200 baud.

Baud Rate	*SW1*	*SW2*	*SW3*
19200	**1**	**0**	**0**
9600	0	1	0
4800	1	1	0
2400	0	0	1
1200	1	0	1
600	0	1	1
300	1	1	1

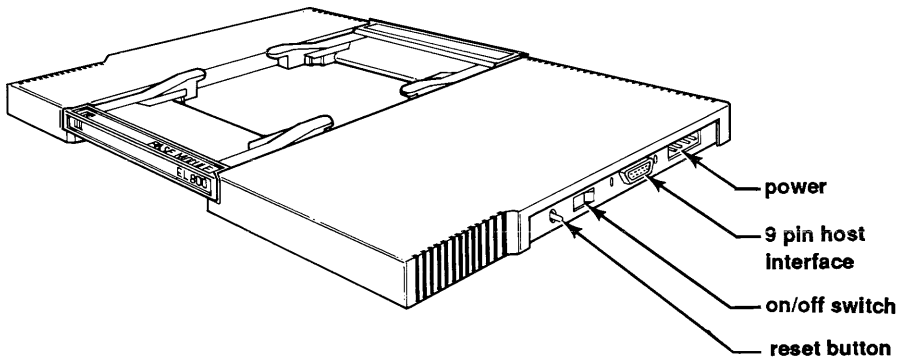
Note: 0 indicates off.

Status LED

The LED lights when the unit is turned on and the self-tests have completed successfully.

The right side of the base module has a reset button, an on/off switch, a serial port and the power connection.

Figure 4-3. The Base Module: Right Side



The connectors, and switches on the right side are described below:

Serial port

The EL 800 has one RS-232C serial interface on the base module.

The baud rate can be set from 300 - 19,200 baud. It must be set in two places: the dip switch on the left side of the base unit (see page 4-3) and in the Configuration/Communications window.

Two cables are provided: a 9 pin to 25 pin cable, and a short 25 pin to 9 pin cable. Complete cabling information can be found in Appendix B.

Reset button

This button is used to reset the emulator

— before starting the EL 800 control software (optional)

— if the emulator stops communicating with the control software. This may happen if the stack loses its place, bus contention leads to execution of incorrect data, or interrupts are incorrectly used in your code.

Power connector

The base module is the only module which needs to be plugged in. A power supply is included with the EL 800. There are two versions available: 110 vac or 220 vac.

The 110 vac version includes a power cord. If you are using the 220 vac version, you will have to supply your own power cord.

Figure 4-4. Power Supply (110 vac)

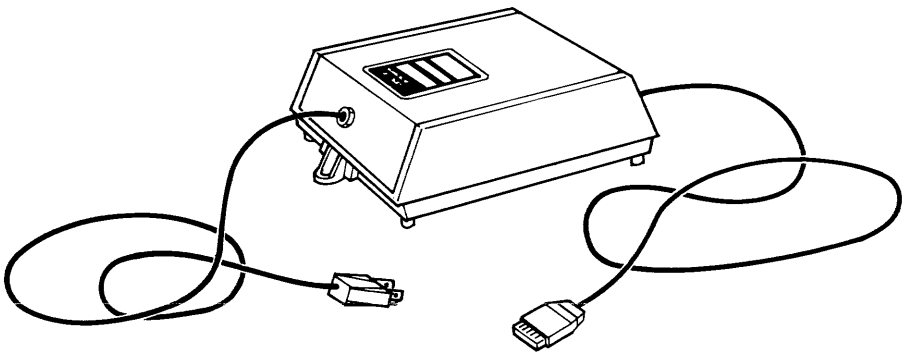
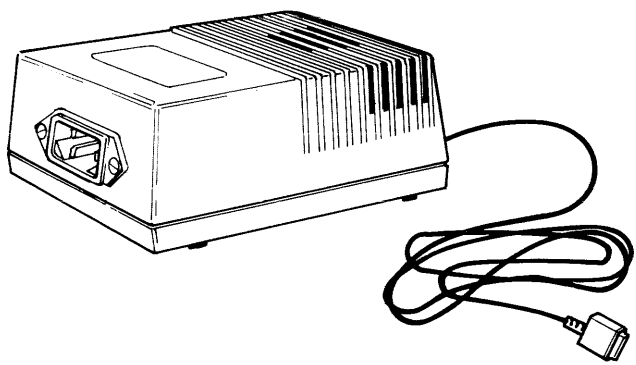


Figure 4-5. Power Supply (220 vac)

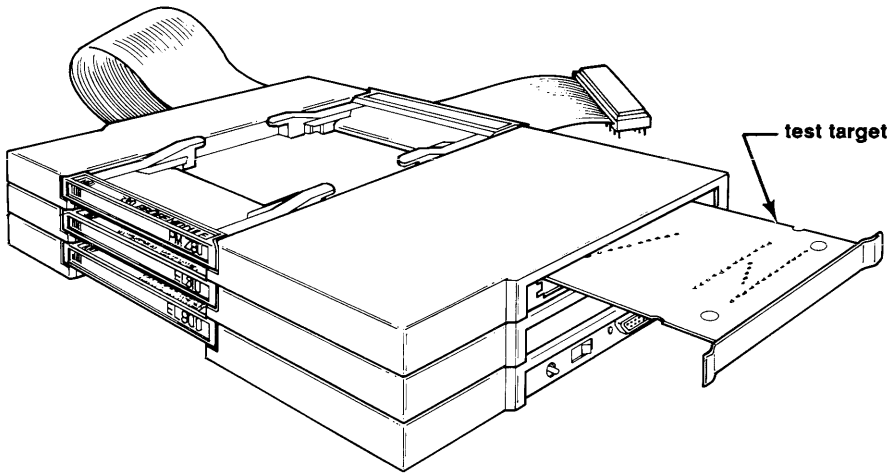


Probe Modules

Each probe module consists of the unit and a probe tip. There is a pull-out board stored inside the probe module which can be used as a test target for debugging software before your target is available.

The probe tip is the small assembly that plugs into the target system microprocessor socket. With the 64180 EL 800, you may use either the DIP probe tip or the PLCC probe tip (see Section 5 for specifics). The Z80 probe tip is available only in the DIP configuration.

Figure 4-6. Probe Module Showing Test Target Board Pulled Out



Probe Tip Use

Please note the following "DOs" and "DON'Ts."

DO

1. Note the location of pin 1 on the probe tip. Make sure to line up pin 1 on the probe tip, with pin 1 on your target socket.
2. Use standard static precautions when using the probe tip, as the probe tip is static sensitive.
3. Use the male-to-male microprocessor socket provided to avoid damaging pins on your probe tip (DIP package only).
4. Use your EL 800 with the vertical stand.

DON'T

1. Don't plug in or unplug the probe tip with the target on.
2. Don't pull on the probe tip or cable.

Test Target Board

The test target board is stored in a pull-out drawer in the probe module.
To use the test target board for software debugging:

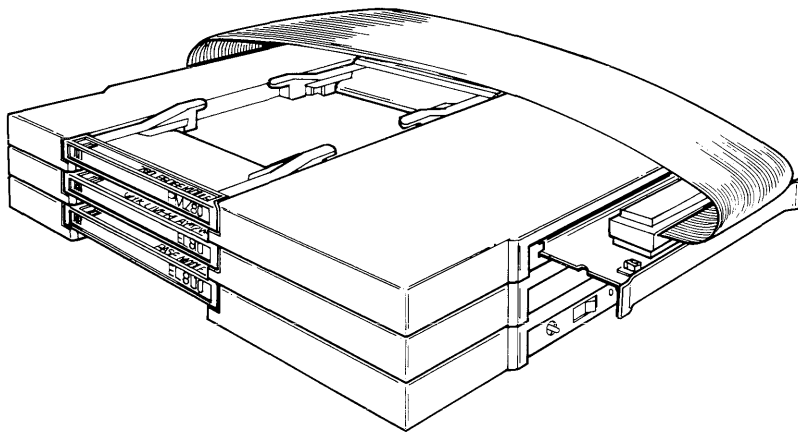
1. Pull the board out from the top slot (see Figure 4-6)
   (There is no release lever - just pull hard on the front corners of the board.
2. Turn the board over.
3. Insert the board in the bottom slot. It goes in about 1": you'll hear a soft click when it is seated. (See Figure 4-7)
4. Insert the probe tip into the microprocessor socket on the board. (See Figure 4-7)

When you are not using the test target board, store it in the top slot.

Target Diagnostic Tests

The EL 800 control software includes ten diagnostic tests you can run on your target hardware. These diagnostics include scope loops and memory tests. Please see the "Operation" section for more information.

Figure 4-7. Test Board with Probe Tip Plugged In

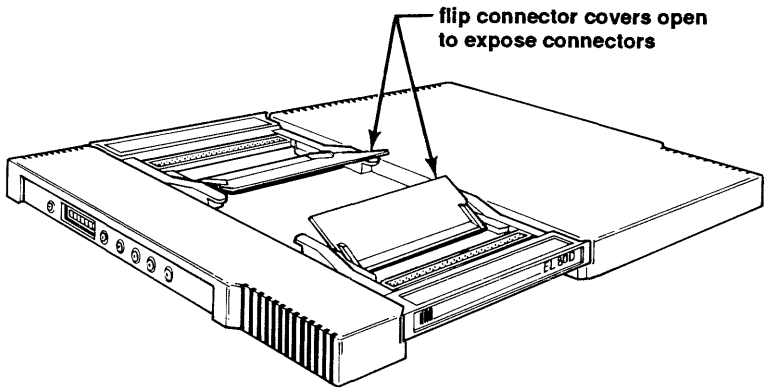


Stacking and Unstacking the Modules

To connect two modules together:

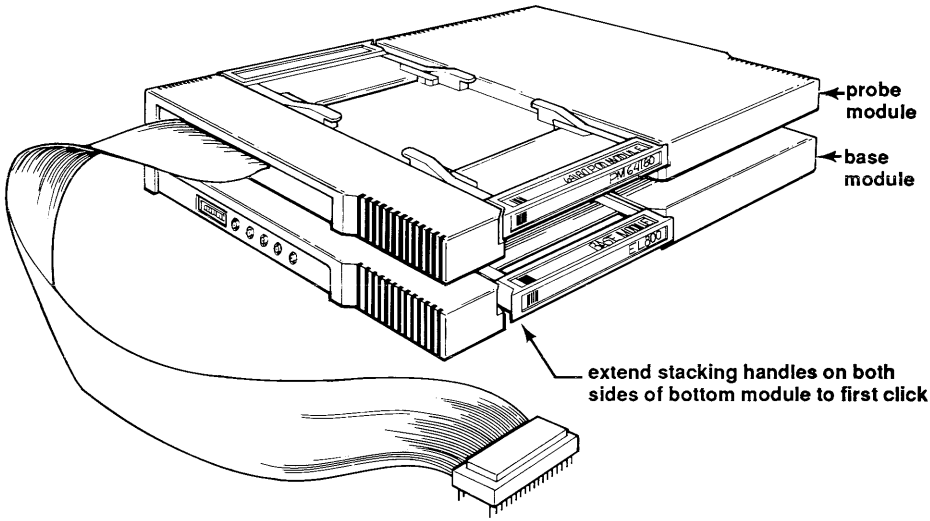
1. Flip both connector covers on the base unit to the open position (see Figure 4-8).

   Figure 4-8. Opening Connector Covers



2. Pull out both dark grey handles on the top of the bottom module to the first click. In this position, the dark grey handles extend 1/3" inch from the module.
3. Place the top module on top of the bottom module.
4. Push down on the top module; you'll hear a click when the module is firmly seated.
5. Push in the handles in the bottom module to lock the two modules together.

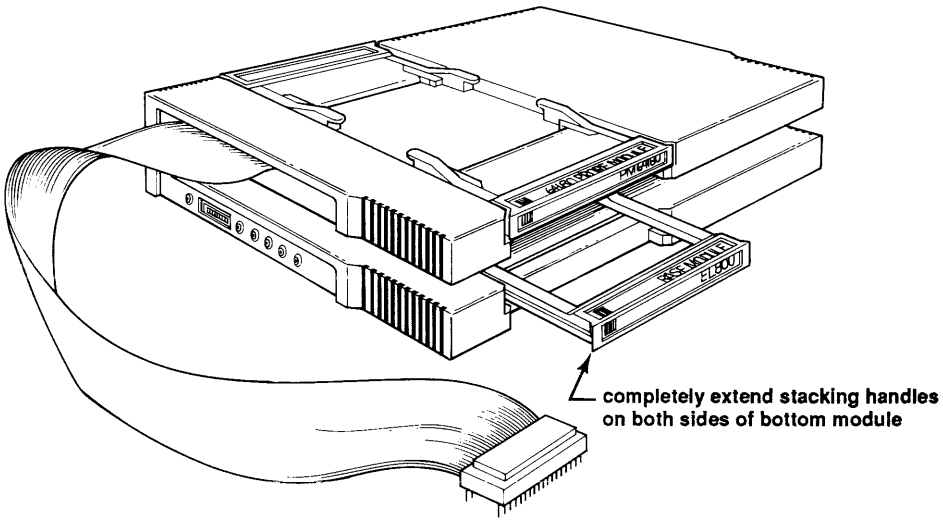
Figure 4-9. Stacking the Modules



To unstack the modules:

1. Pull the dark gray handles on the module just below the separation point out to the second click (extending approximately 2" from the module). (See Figure 4-10.) This releases the connection between the modules. DO NOT pry apart the units as this will damage the module connectors.
2. Lift the top module(s) off.

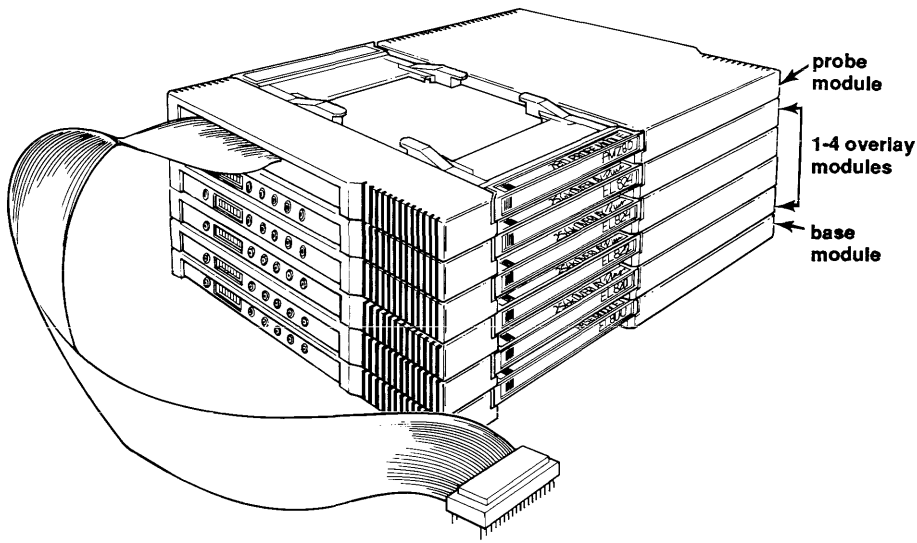
Figure 4-10. Unstacking the Modules



Stacking Order

No matter what combination of optional modules you purchase, the base module always goes on the bottom. The following diagram indicates the order for additional optional modules such as overlay memory.

Figure 4-11. Stacking Order



Overlay Memory Modules

There are 6 overlay memory modules available:

64KB overlay memory with no battery backup (EL-810)
64KB overlay memory with battery backup (EL-820)

128KB overlay memory with no battery backup (EL-812)
128KB overlay memory with battery backup (EL-822)

256KB overlay memory with no battery backup (EL-814)
256KB overlay memory with battery backup (EL-824)

Any four of these can be stacked together, so you can get exactly the amount of overlay memory you need, from 64K to 1 MB. The overlay modules should be stacked in between the base module and the probe module.

There are 8 dip switches on the left side panel. Switches 1 and 2 are used to show the configuration of overlay modules you are using. Switches 3-8 are not used, and should be left at 1. No two stacked modules should have the same switch 1 and 2 setting. (See Figure 4-13.)

SW1	*SW2*	*Description*
0	0	First overlay module
0	1	Second overlay module
1	0	Third overlay module
1	1	Fourth overlay module

A yellow LED on the right side panel of each overlay module indicates when overlay memory is accessed.

Figure 4-12. Overlay Memory Module

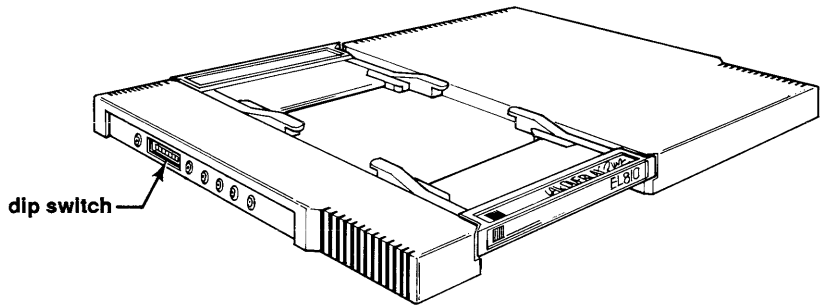
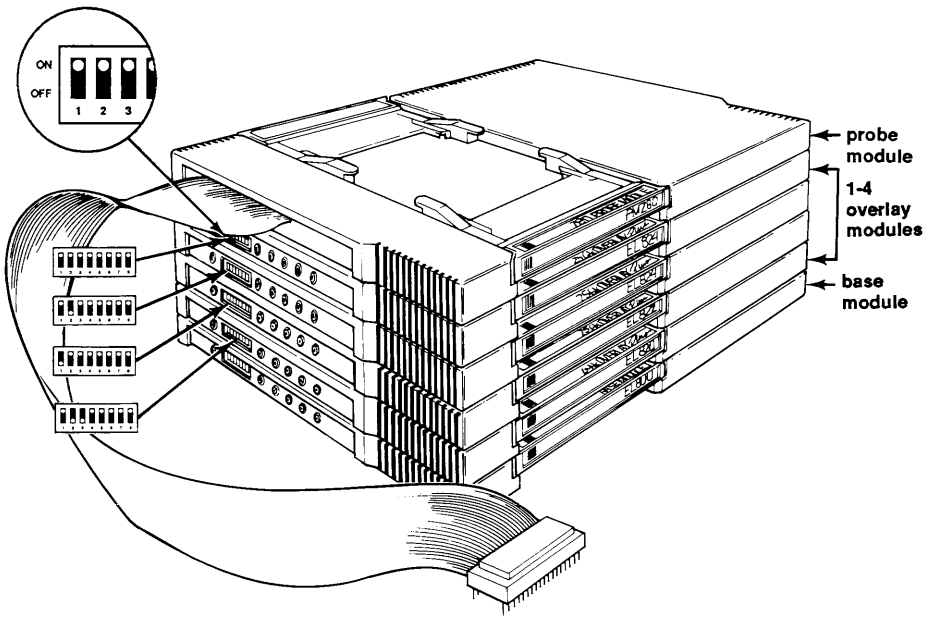


Figure 4-13. Overlay Memory Showing Dip Switch Settings



Maintenance

A minimum of maintenance is required for your EL 800 emulator. The three areas which require attention are the cables, probe-tip and cooling vents.

Cables

The cables are the most vulnerable part of the EL 800 due to flexing during use. Inspect the cables regularly for obvious damage such as cuts, breaks or tears. Even if there is no visible damage, wires within a cable may break, causing erratic problems if the cable is flexed while the emulator is running. To isolate a faulty cable, swap the cable with a known good cable.

Probe Tip

The probe tip is the small assembly that plugs into the target system microprocessor socket. With the 64180 EL 800, you may use either the DIP probe tip or the PLCC probe tip (see Section 5 for specifics). The Z80 probe tip is available only in the DIP configuration.

Inspect the adapter each time you use the probe tip, as the pins can be easily bent or broken during insertion or extraction. When you are not using the probe tip, store it in the protective cardboard box it was shipped in.

You can protect the adapter by installing a microprocessor socket onto the adapter. If a pin on the socket is broken, it is easier to replace than the probe tip.

Cooling Vents

Do not block the cooling vents on the back of each module. The EL 800 can be used in either the vertical or horizontal position. We encourage use in the vertical position, as it increases the efficiency of the free convection cooling.

Troubleshooting

If your emulator is not working, please do the following:

1. Check that the stacked modules are seated properly, with all the handles pushed completely in.
2. Make sure the the emulator is plugged into a target system or to the built-in test target. If using the test target, make sure it is firmly seated in the bottom slot in the probe module.
3. Make sure power is on to both the emulator and the target.
4. Push the emulator reset button and restart your EL 800 control software.

Common start-up problems are covered in Section 2.

If you experience any problems not listed, please contact Applied Microsystems Corporation at (800)426-3925 (US), (206)882-2000 (in WA. state), or call your local sales office (International).

We do not recommend a component-level repair in the field, unless performed by a qualified service engineer.

Specifications

ELECTRICAL	
Input power:	110 vac 60 hz or 220 vac 50 hz Table top power supply
PHYSICAL	
Dimensions (per module):	8.5" x 11.0" x 0.85" 21.6 cm x 27.9 cm x 2.2 cm
Probe tip length:	16", 40.6 cm
ENVIRONMENTAL	
Operating temperature:	0° C to 40° C
Humidity:	0-90% noncondensing

Exit the System

You can exit the EL 800 control software from any of the main windows or from the main menu by typing **:X**. You will see a prompt to confirm your exit. Any configuration changes you make will be saved in the configuration database when you leave the EL 800 control software.

You should always exit the software before turning the emulator power off.

SECTION 5

Table of Contents

Chip-Emulator Characteristics

CHIP-EMULATOR CHARACTERISTICS

This section deals with the Z80 and 64180 microprocessors in two contexts:

1. a functional overview, and
2. sensitivities in the design and use of the microprocessors which may affect emulation with the EL 800 emulator.

Information on the Z80 begins on page 5-1. Information on the 64180 begins on page 5-9.

Z80

This section contains a functional overview of the Z80 microprocessor, information on additional Z80 address comparators, Z80 emulator equivilent circuit information, and Return from Interrupt (RETI) considerations.

Z80 Functional Overview

The Z80 is an 8-bit microprocessor in the 8080 architectural class. The Z80 has a 16-bit address bus, unmultiplexed, and an 8-bit data bus.

The processor has six general-purpose 8-bit registers that are also accessible as three 16-bit register pairs, one 8-bit accumulator, one 8-bit flag register, a 16-bit program counter, a 16-bit stack pointer, and two 16-bit index registers.

In addition to the above, the accumulator, flag, and general-purpose registers are duplicated in the "prime" registers. The Z80 instruction set contains commands that toggle accessibility between the main register set an the primes.

Z80 interrupts include a non-maskable interrupt, and a maskable interrupt usable in three modes.

Of additional interest are the block-transfer and block-search instructions.

The Z80 uses separate I/O and memory addresses.

For more information, consult the *Zilog Components Data Book.*

Address Comparators

The Z80 uses only 16 address lines (A:0-15), and the EL 800 has 20 available (A:0-19). The extra four address lines can be used as four additional inputs from the target system. To use these, set the EXTADDR switch (Configuration: Emulator window :CE) to 1, and connect the gripper clips from the Z80 probe module to the device you want input from. Use 20 bit values for the addrA and addrB comparators.

There are two rules when using EXTADDR set to 1:

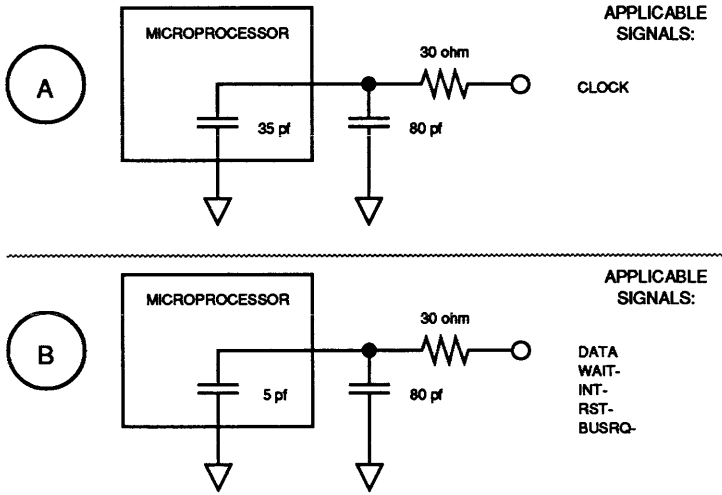
1. If you want to use two addresses which differ only in the upper nibble, you must use two separate comparators. For example, you can't set two items in comparator A to AFFFF and BFFFF. Put one item in comparator A and one in comparator B.

2. If you enter an address range that spans the 64K boundary of A0-A15, it uses up both address comparators for that event state. For example, if you specify the range AFFFF to B1000, you cannot specify any other addresses or ranges for comparators A and B in that state.

Equivilent Circuits/Input Loading

This section contains information regarding emulator/processor input loading and equivilent circuits. Equivalent circuits for the various probe tip microprocessor inputs are shown in Figures 5-1A and 5-1B. These models include the input capacitance of the processor itself and the additional emulator loading.

Figure 5-1. Emulator Input Equivalent Circuits (Z80)



While the emulator input capacitance is small, a marginal input signal, particularly the clock, may be degraded to an inoperable level by the introduction of the emulator probe tip. You should observe a strong, clean clock signal at the processor's clock input.

It should be noted here that it is possible to drive the Z80 with a weak clock in such a fashion that the processor will appear to function correctly but will generate hard-to-diagnose errors.

Return-From-Interrupt (RETI) Considerations

If your target circuit uses any of the Z80 peripheral devices in interrupt mode 2, and you wish to run your interrupt service routines out of overlay, you must take certain steps to ensure proper emulation with your EL 800 emulator. This section explains the potential problems, and the necessary steps to avoid them.

In interrupt mode 2, the interrupting peripheral device must see the return-from-

interrupt (RETI) instruction at the end of the interrupt service routine to complete its interrupt cycle. To see and act upon this instruction, the interrupting device must see the RETI bytes on the data bus, an active RD- indicating a read, and an active M1- indicating an opcode fetch cycle. This set of conditions can, under certain circumstances, make emulation more difficult.

The first problem is bus contention. If the interrupting peripheral is to complete its interrupt cycle, the target's RD- line must be asserted, and the local target data bus may be driven by the target memory, either directly as in Figure 5-2A, or through the bidirectional buffer, as in Figure 5-2B. Note the use of RD- as a buffer direction control signal. The local target data bus may also be driven by the overlay RAM through the control FET.

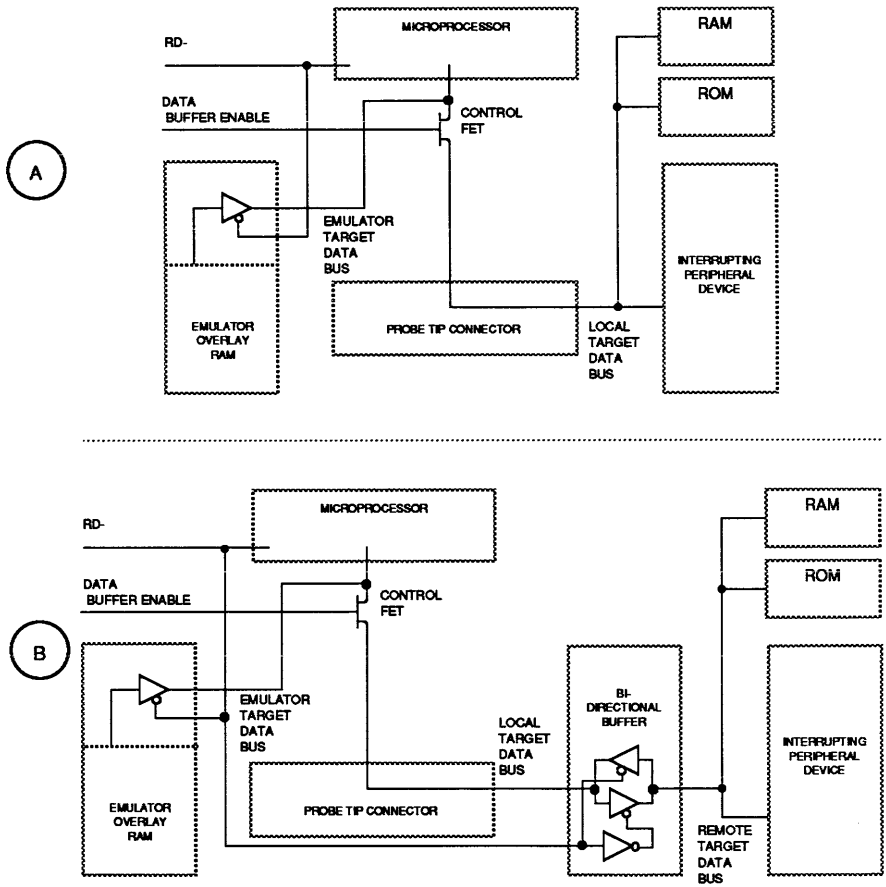
The bus contention arises from the bus being driven simultaneously by both the target and overlay. Not only is it generally bad practice to allow bus contention, the emulator's target data bus may be corrupted in the process, possibly causing erroneous execution.

The second problem is the inability of the interrupting peripheral to see an RETI fetch and complete its interrupt cycle. In the circuit of Figure 5-2A, when an RETI is fetched from overlay, the interrupting peripheral sees a corrupted data bus, and in the circuit of Figure 5-2B, the peripheral sees valid data, but not an RETI. Essentially then, in either circuit configuration, the emulator's target data bus integrity is threatened, and the interrupt cycle has not been completed.

The block diagrams of figures 5-2A and 5-2B represent the two major design configurations for the purpose of this discussion. These figures show the EL 800 emulator already installed. In Figure 5-2A, the probe-tip connector, where the processor will be in the finished circuit, is placed directly on the same data bus as the system memory and the interrupting peripheral. This is the local target data bus.

In Figure 5-2B, the probe-tip connector is buffered from the memory and peripheral bus, now the remote target data bus, by a bidirectional buffer, such as a 74LS245.

Figure 5-2. Target Circuit Configurations (Z80)



DATA Switch Function

To address the bus contention problem, the EL 800 has logical control over the data-buffer-enable line, and can selectively disconnect the emulator target data bus from the local target data bus. User control of the data-buffer-enable line is asserted by setting the DATA switch in the Configuration/Emulator window. The DATA switch settings have the following effects:

DATA = 0 This setting allows data to pass between the emulator target data bus and the local target data bus on all target and overlay accesses except reads from overlay.

DATA = 1 This setting allows data to pass between the emulator target data bus and the local target data bus on target accesses only. Data is not passed during overlay accesses.

DATA = 2 This setting allows data to pass between the emulator target data bus and the local target data bus during all executions cycles, and also during emulator peeks and pokes to target RAM.

RETI In Overlay Solutions

Three solutions to the RETI-in-overlay situation are shown below, two for each setting of the DATA switch, with buffered and unbuffered data busses.

DATA = 0, Unbuffered

A solution for the RETI problem in the circuit of Figure 5-2A, with the data switch set to 0, is as follows:

If your target is in the form of Figure 5-2A, setting the emulator configuration data switch to 0 will ensure normal target circuit operation in all cases except that of read from overlay memory. In the case of an overlay memory read, the local target data bus is driven by the target memory, and the emulator target data bus is driven by overlay RAM. The control FET is not conducting, and no bus contention exists.

If the address read is a valid target memory address, the valid and uncorrupted contents of that location appear on the local data bus. With the interrupt service routine's RETI mapped to overlay, the interrupt service routine in overlay can be

terminated with a jump to a previously unused location in target RAM. If an RETI is placed in this location, it will be seen by both the interrupting peripheral, which will terminate its interrupt cycle, and by the emulator, which will have de-selected overlay RAM, avoiding contention, according to our mapping scheme.

This solution changes timing slightly, requires an unused target RAM location, and the RETI-jump substitution requires a change in code and code size. If these changes are unacceptable, the target circuit can be designed with socketed ROM, pin-compatible with some RAM device. If the target ROM is replaced with RAM, and the RETI is loaded into that RAM at the same addresses occupied by RETIs in overlay, exact timing is maintained, and interrupt cycles are completed.

DATA = 1, Unbuffered

A solution for the RETI problem in the circuit of Figure 5-2A, with the data switch set to 1, is implemented in the same way as the previous example, with the exception of the DATA switch setting =1, and has essentially the same results. This solution is intended for the situation where the ROM on your target circuit is not read-write decoded.

DATA = 2, Unbuffered

A solution for the RETI problem in a circuit like Figure 5-2A, with the DATA switch set to 2, is implemented with the memory space containing the RETI mapped to overlay, the target memory device(s) must be removed or hard-wired de-selected to avoid bus contention. Care should be taken to ascertain that **ALL** target devices occupying addresses mapped to overlay are rendered incapable of driving the local data bus.

CAUTION

Bus contention can shorten the life of emulator and target components, and corrupt the emulation process.

DATA = 0, Buffered

This is the circuit configuration of Figure 5-2B. The RETI solutions are the same as in the DATA=0, unbuffered bus situation.

DATA = 1, Buffered

This is the circuit configuration of Figure 5-2B. The RETI solutions are the same as in the DATA=0, unbuffered bus situation.

DATA = 2, Buffered

This is the circuit configuration of Figure 5-2B. Unless you add direction-control circuitry to the target bus bidirectional buffer to resolve bus contention problems, this configuration may not be safely and effectively implemented.

64180

This section contains a functional overview of the 64180 microprocessor, information on equivilent circuits and input loading, Z-mask/probe tip considerations, and Return from Interrupt (RETI) considerations.

64180 Functional Overview

The 64180 is an 8-bit microprocessor in the 8080 architectural class, and is upward-compatible with the Zilog Z80. The 64180 has a 19 or 20-bit address bus, essentially unmultiplexed, and an 8-bit data bus.

The processor has six general-purpose 8-bit registers that are also accessible as three 16-bit register pairs, one 8-bit accumulator, one 8-bit flag register, a 16-bit program counter, a 16-bit stack pointer, and two 16-bit index registers.

In addition to the above, the accumulator, flag, and general-purpose registers are duplicated in the "prime" registers. The 64180 instruction set contains commands that toggle accessibility between the main register set an the primes.

The 64180 incorporates a memory management unit, a two-channel direct memory access controller, one synchronous and two asynchronous serial ports, two 16-bit programmable counter/timers, and a 12-source interrupt controller.

64180 interrupts include a non-maskable interrupt, three maskable interrupts, an undefined-opcode interrupt, two timer interrupts, two DMA interrupts, and three serial port interrupts.

Of additional interest are the block-transfer and block-search instructions.

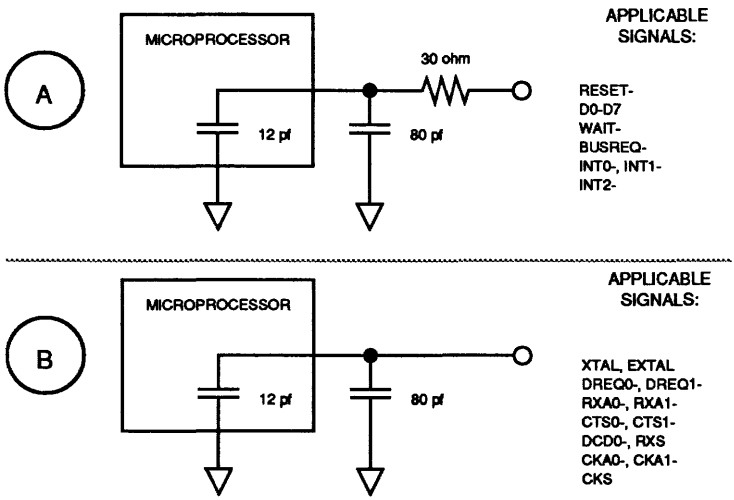
The 64180 uses separate I/O and memory addresses.

For more information, consult the *Hitachi 64180 8-Bit Microprocessor User's Manual.*

Equivilent Circuits/Input Loading

Equivalent circuits for the various probe tip microprocessor inputs are shown in Figure 5-3. These models include the input capacitance of the processor itself and the additional emulator loading.

Figure 5-3. Emulator Input Equivalent Circuits (64180)



While the emulator input capacitance is small, a marginal input signal may be degraded to an inoperable level by the introduction of the emulator probe tip. You should observe strong, clean input signals at the processor's inputs.

It should be noted here that it is possible to drive the 64180 with a weak input in such a fashion that the processor will appear to function correctly but will generate hard-to-diagnose errors.

Z-Mask/Probe Tip Type Considerations

The Hitachi HD64180 is manufactured in two common packages, the 64-pin DIP package, and the 68-pin PLCC package. The EL800 64180 emulator is manufactured either with the probe tip configured to be inserted in the DIP or PLCC processor socket. In addition, the probe tip that plugs into the DIP socket also accepts only a DIP-packaged processor, and the PLCC-socket-fitting probe tip accepts only a PLCC-packaged processor.

Under many conditions, and with the use of appropriate adapters (where available), the DIP version may be used with a PLCC target, and the PLCC version may be used with a DIP target. Note that:

- Such an adapter will increase susceptibility to noise, ground, and loading problems
- The emulator with adapter may not run in a marginal target
- There are some obvious restrictions due to a different number of signals on the two packages

This mixing of connector types may be done for three reasons. Two of the reasons are simple mechanical adaptation (although some logical and electronic considerations pertain), but the third reason is to allow complete emulation of a DIP-socketed Z-mask 64180.

PLCC Target/DIP Emulator

The HD64180 silicon itself is capable of driving 20 address lines, to address 1 Mbyte of memory. In the PLCC version, all 20 address lines are brought out, and the PLCC 64180 can actually address a 1 Mbyte memory.

The DIP version, with 4 pins fewer than the PLCC device, can not supply pins for all address lines, and only 19 lines are brought out. The DIP 64180 can therefore address a maximum of 512 Kbytes of memory.

If an adapter is used to fit a DIP-configured emulator to PLCC-configured target, the user must take two steps to assure proper emulation.

1. Address line 19 on the target must be pulled up or down, according to the user's needs, with an appropriate resistor, limiting the physical address space to 512 Kbytes.
2. The user must verify that his code can run in the hardwired-selected 512 Kbyte memory space. If necessary, code must be modified to meet this criterion.

DIP Target/PLCC Emulator

This is essentially the reverse of the previous condition. In this case, the processor/emulator is capable of driving one more address line (A19) than the target circuit. The processor/emulator is capable of addressing 1 Mbyte of memory, and the target's address bus can only address 512 Kbytes of memory.

To emulate in this configuration, it should be noted that, although the target does not see A19, it is still driven by the processor according to the MMU and your code. For meaningful trace and break to be effected, care should be taken (with your code) to assure a known state of A19.

DIP Target/PLCC Emulator, Z-Mask Processor

This condition is a variant of the previous condition, and is significant for two reasons:

1. If the Hitachi or Zilog Z-mask version of the 64180 is used, the processor can be set to suppress selectively the processor output signal LIR-, a signal without which the DIP version of the emulator cannot emulate completely.
2. The PLCC version of the EL800 64180 emulator can completely reconstruct code execution without the LIR- signal, allowing complete emulation of the Z-mask processor.

This condition is therefore important when emulating a DIP target using a Z-mask processor. The PLCC EL800 with a PLCC-DIP adapter is the solution.

The code restrictions of the previous example are the only restrictions in this condition.

Z-Mask Emulation With DIP Emulator

Reference has been previously made to "incomplete" emulation when emulating the Z-mask 64180 processor with the DIP-processor-configured EL800 emulator. This section addresses Z-mask emulation and its limitations with the DIP emulator.

If you use the Z-mask processor and set bit D7 of internal I/O register OMCR to 0, you will suppress normal assertion of the processor's LIR- output signal. In normal operation of the DIP-configured EL800 64180, partial loss of LIR- will degrade emulator operation to the point of inoperability, and the following steps should be taken.

Replace the H64.LCA file in your working directory with the file presently named H64Z.LCA, also in your working directory, as the file read by the emulator software at bootup. This can be accomplished by renaming H64.LCA to H64.TMP and THEN renaming H64Z.LCA to H64.LCA.

If the emulator is booted up with the new H64.LCA file, emulator operation will be correct with the suppressed LIR- signal, with one exception: internal DMA read cycles will appear to the emulator as opcode fetches, and will be traced as such.

Return-From-Interrupt (RETI) Considerations

If your target circuit uses any of the 64180 peripheral devices in interrupt mode 2, and you wish to run your interrupt service routines out of overlay, you must take certain steps to ensure proper emulation with your EL 800 emulator. This section explains the potential problems, and the necessary steps to avoid them.

In interrupt mode 2, the interrupting peripheral device must see the return-from-interrupt (RETI) instruction at the end of the interrupt service routine to complete its interrupt cycle. To see and act upon this instruction, the interrupting device must see the RETI bytes on the data bus, an active RD- indicating a read, and an active LIR- indicating an opcode fetch cycle. This set of conditions can, under certain circumstances, make emulation more difficult.

The first problem is bus contention. If the interrupting peripheral is to complete its interrupt cycle, the target's RD- line must be asserted, and the local target data bus may be driven by the target ROM, either directly as in Figure 5-4A, or through the bidirectional buffer, as in Figure 5-4B. Note the use of RD- as a buffer direction

control signal. The local target data bus may also be driven by the overlay RAM through the control FET.

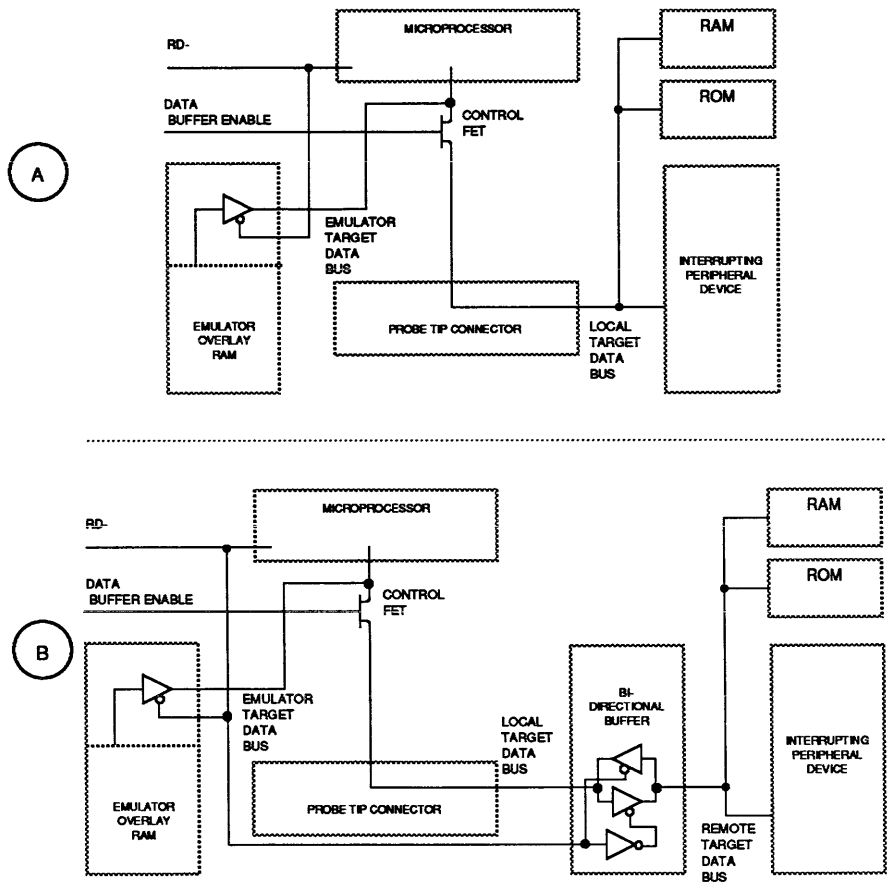
The bus contention arises from the bus being driven simultaneously by both the target and overlay. Not only is it generally bad practice to allow bus contention, the emulator's target data bus is corrupted in the process, possibly causing erroneous execution.

The second problem is the inability of the interrupting peripheral to see an RETI fetch and complete its interrupt cycle. In the circuit of Figure 5-4A, when an RETI is fetched from overlay, the interrupting peripheral sees a corrupted data bus, and in the circuit of Figure 5-4B, the peripheral sees valid data, but not an RETI. Essentially then, in either circuit configuration, the emulator's target data bus integrity is threatened, and the interrupt cycle has not been completed.

The block diagrams of figures 5-4A and 5-4B represent the two major design configurations for the purpose of this discussion. These figures show the EL 800 emulator already installed. In Figure 5-4A, the probe-tip connector, where the processor will be in the finished circuit, is placed directly on the same data bus as the system memory and the interrupting peripheral. This is the local target data bus.

In Figure 5-4B, the probe-tip connector is buffered from the memory and peripheral bus, now the remote target data bus, by a bidirectional buffer, such as a 74LS245.

Figure 5-4. Target Circuit Configurations (64180)



DATA Switch Function

To address the bus contention problem, the EL 800 has logical control over the data-buffer-enable line, and can selectively disconnect the emulator target data bus from the local target data bus. User control of the data-buffer-enable line is asserted by setting the DATA switch in the Configuration/Emulator window. The DATA switch settings have the following effects:

DATA = 0 This setting allows data to pass between the emulator target data bus and the local target data bus on all target and overlay accesses except reads from memory.

DATA = 1 This setting allows data to pass between the emulator target data bus and the local target data bus on target accesses only. Data is not passed during overlay accesses.

DATA = 2 This setting allows data to pass between the emulator target data bus and the local target data bus during all executions cycles, and also during emulator peeks and pokes to target RAM.

RETI In Overlay Solutions

Three solutions to the RETI-in-overlay situation are shown below, two for each setting of the DATA switch, with buffered and unbuffered data busses.

DATA = 0, Unbuffered

A solution for the RETI problem in the circuit of Figure 5-4A, with the data switch set to 0, is as follows:

If your target is in the form of Figure 5-4A, setting the emulator configuration data switch to 0 will ensure normal target circuit operation in all cases except that of read from target memory. In the case of target memory read, the local target data bus is driven by the target memory, and the emulator target data bus is driven by overlay RAM. The control FET is not conducting, and no bus contention exists.

If the address read is a valid target memory address, the valid and uncorrupted contents of that location appear on the local data bus. With the target ROM space containing the interrupt service routine's RETI mapped to overlay, and the remaining

target RAM mapped to target, the interrupt service routine in overlay can be terminated with a jump to a previously unused location in target RAM. If an RETI is placed in this location, it will be seen by both the interrupting peripheral, which will terminate its interrupt cycle, and by the emulator, which will have de-selected overlay RAM, avoiding contention, according to our mapping scheme.

This solution changes timing slightly, requires an unused target RAM location, and the RETI-jump substitution requires a change in code and code size. If these changes are unacceptable, the target circuit can be designed with socketed ROM, pin-compatible with some RAM device. If the target ROM is replaced with RAM, and the RETI is loaded into that RAM at the same addresses occupied by RETIs in overlay, exact timing is maintained, and interrupt cycles are completed.

DATA = 1, Unbuffered

A solution for the RETI problem in the circuit of Figure 5-4A, with the data switch set to 1, is implemented in the same way as the previous example, with the exception of the DATA switch setting =1, and has essentially the same results. This solution is intended for the situation where the ROM on your target circuit is not read-write decoded.

DATA = 2, Unbuffered

A solution for the RETI problem in a circuit like Figure 5-4A, with the DATA switch set to 2 is implemented with the target ROM space, containing the RETI, mapped to overlay, and the remaining target RAM mapped to target, the target ROM must be removed or hard-wired de-selected to avoid bus contention. Care should be taken to ascertain that **ALL** addresses mapped to overlay are rendered incapable of driving the local data bus.

CAUTION

Bus contention can shorten the life of emulator and target components, and corrupt the emulation process.

DATA = 0, Buffered

This is the circuit configuration of Figure 5-4B. The RETI solutions are the same as in the DATA=0, unbuffered bus situation.

DATA = 1, Buffered

This is the circuit configuration of Figure 5-4B. The RETI solutions are the same as in the DATA=0, unbuffered bus situation.

DATA = 2, Buffered

This is the circuit configuration of Figure 5-4B. Unless you add direction-control circuitry to the target bus bi-directional buffer to resolve bus contention problems, this configuration may not be safely and effectively implemented.

SECTION 6

Table of Contents

Operation

OPERATION

WINDOW BASICS

The user interface for the EL 800 microprocessor development system groups related emulation functions in windows. You can use multiple windows simultaneously, and position and size the windows as you like.

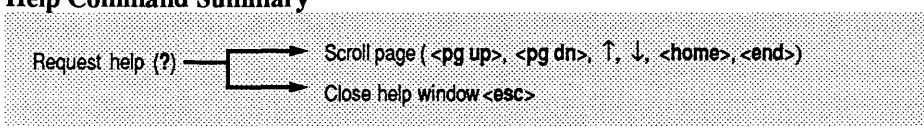
This section first covers the basics of using the windows, and then provides an alphabetic reference for each of the main windows, including all commands associated with the window. Sub-windows and sub-window commands are described after the main window they are associated with. Within each window section, under the heading "Command Summary", there is an illustration showing all commands relevant to that window. A quick-reference list of all commands is also provided in Appendix A.

Information in windows can be scrolled with the cursor control and paging keys if it spans more than one screen. Data in visible windows is updated as it changes.

Getting Help

If you need additional information on any command, window, or entry, you can type a question mark (?) to display a help message. The help message is displayed in a separate window on the screen. The information shown in the help window is relevant to your current activity. When you are finished reading the help information, press <esc> to return to the active window.

Help Command Summary



Prompt Basics

The EL 800 lists all possible options for your current activity on a command line at the bottom of the screen. This line is called the *prompt*, since it prompts you for the next function. The following rules are used to select commands from a prompt line:

1. Upper case letters (highlighted) in an item are used to select that item.
2. If there are no upper case letters, use the highlighted numbers to make a selection.
3. Words enclosed in parentheses ( ), or all lower case words, indicate that you should type in a value. Don't type the parentheses themselves.
4. Square brackets [ ] indicate optional items. Do not type the brackets.
5. Punctuation shown on the command line is important! Enter any commas (,) and semicolons (;) as you see them on the prompt line.

The prompt line essentially anticipates your next request. Try typing a few commands, and watch the command line change as you enter additional information. The interface is responding to the incoming information and actually guides you through command entry.

When a window is activated, the prompt shows the choices available for that window. Note that choices are separated by a slash (/).

When prompted for a value, you can either type the hexadecimal value or use the *expression analyzer* to convert symbols, math operators, C language operators, or numbers in varying radixes to a hexadecimal value (see the *Expression Analyzer* section).

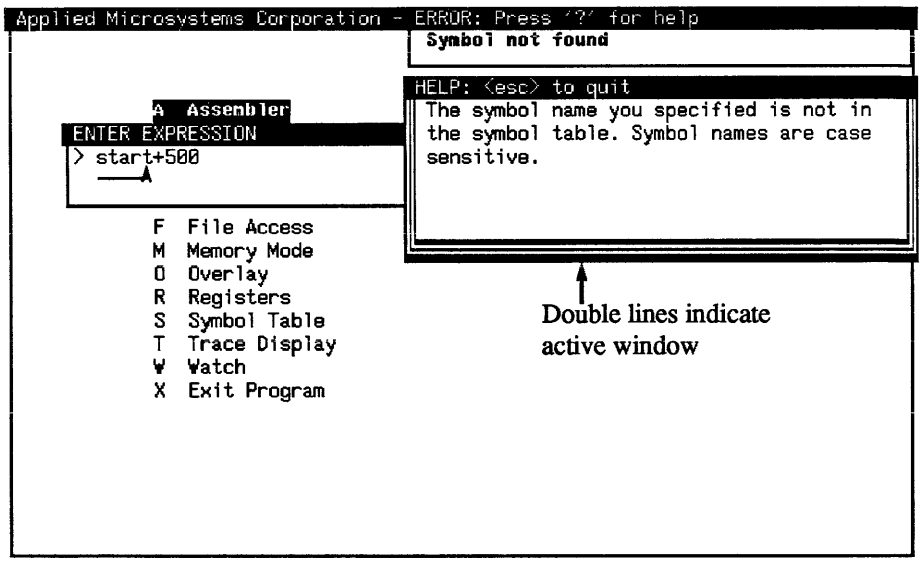
Error Messages

If you type an incorrect value, or if an error occurs, an error message appears in the upper right corner of the screen. You can get additional information on each error message by typing a question mark (?) while the error is displayed.

Note that you must press <esc> to remove the help message from the screen.

You can remove the error message from the screen by pressing any key. The following example shows how an error message and related help message are displayed. The error occurred because a non-existent symbol name was specified in the expression analyzer as the assembler starting address.

Error Message and Help Windows



Switching Windows

There are three ways to activate any window:

1. If you are at the Main menu, type the first letter of the window name.
2. If you are at the Main menu, use ↑ and ↓ to highlight the window name and press **<return>**.
3. From anywhere in the control software, type a colon (:) followed by the first letter of the window name.

After selecting the window, you will either see a prompt on the bottom line of the screen (A, C, and M windows), or the new window will appear (B, D, E, F, O, R, S, T, and W windows). All windows listed on the Main menu are called *top level* windows.

In the Assembler, Configuration, and Memory Mode windows, the prompt indicates that you must make a choice or enter data before activating the window.

To close a window, activate it and then press **<esc>**.

Displaying Multiple Windows on the Screen

You can have multiple windows on the screen at one time. From the first prompt in any window, type a colon (:) and you'll see the following prompt:

_ << A/B/C/D/E/F/M/O/R/S/T/W/X

Type the letter of the window you want to activate. The new window overlaps the previous window, unless you had previously moved or resized the window.

If two or more windows are on the screen simultaneously, the *active window* is shown surrounded by double lines. The active window, always the most recently invoked, determines which commands are listed on the command line at the bottom of your screen and your current function. A window must be active before you can use it. Typing <esc> from a window returns you to the previously active window. Typing <esc> from a top level window closes the window.

Any visible window's data is constantly updated as emulation progresses so that you can change one aspect of emulation and view the resultant changes in other areas. For example, while stepping through code in the Emulate window, you can watch the Memory, Register, Trace, and Watch windows simultaneously for changes.

Moving and Sizing Windows

With one or more windows open on your screen, you can position and size each window as you see fit. Press <F1> to move the active window. (If the window you want to move is not currently active, type a colon (:) followed by one of the letters [ABCDEFMORSTWX] to activate the proper window.) Use the left, right, up, and down cursor control keys to move the window to the position you desire. When finished, press <esc> or <return> to return to the active window.

Sizing windows works similarly. Press <F2> to resize the active window. Use the cursor control keys to size the window as you like. Be sure to press <esc> or <return> when you finish.

Using the <F1> and <F2> keys on a particular window allows you to completely configure your screen the way you like it. Once set, the position and size of a window will be maintained until you exit the control software, even if you close and then reopen the window. You can also save the window configuration permanently by going to the File Access (:F) window, selecting Save Windows, and entering the filename for configuration storage. This window placement and size will be used when you next open the window or control software.

If you want to save several different window configurations, that is also possible. You can then exit from the control software and return, restoring the window configuration of your choice.

Shell Escape

The *shell escape* feature allows you to temporarily leave the EL 800 control software and enter operating system commands. To get to the shell escape process (COMMAND.COM is the default), type an exclamation point (!). The shell escape only works from the Main Menu and the top level windows. Once you have chosen a command or sub-window, you must **<backspace>** or **<esc>** out to the top level of the window before using the shell escape. To return to the EL 800 software, use the DOS "EXIT" command.

To change the default shell escape to a program other than COMMAND.COM, use the Configuration/System window (see page 6-56).

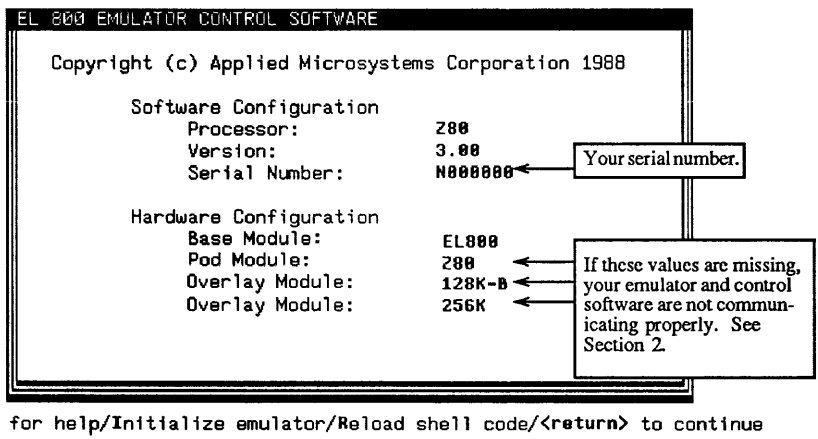
If the shell escape feature seems to be working erratically, you may have overextended your host computer's memory. Try clearing all unnecessary resident programs from memory and restarting the EL 800 control software.

COVER WINDOW <ctrl-c>

Description

The Cover window is the first screen displayed after you start the EL 800 software. It shows the software serial number and version, the processor type, and the names of the hardware modules installed. You cannot change the configuration information from this window. (It can only be changed by switching pods, overlay, or base modules.)

Cover Window



You can take four actions from the Cover window:

Go to Main Menu To access all emulator functions via the main menu, press **<return>**.

Initialize emulator The EL 800 emulator automatically saves the current status of many windows and internal states when you exit the software, so that you can resume debugging where you

left off. If you want to reset all the windows and internal states to their default configuration, press **I** to initialize the emulator. The chart in the *Cover Window: Initialize* section shows the differences between the hardware emulator reset button and the Initialize choice. See Appendix D for details on what is saved at power-on, reset, power off, software startup and exit.

Reload shell code The emulator shell code is used to configure the emulator logic, and should only be reloaded (**R**) if you change to a new hardware processor module or if you receive a software update. The shell code is saved in the battery backed up RAM in the emulator, so does not need to be reloaded even if the emulator's power is turned off.

Get help Type **?** to view help information which explains the basics of using the EL 800 control software and how to use the other three choices in the Cover window.

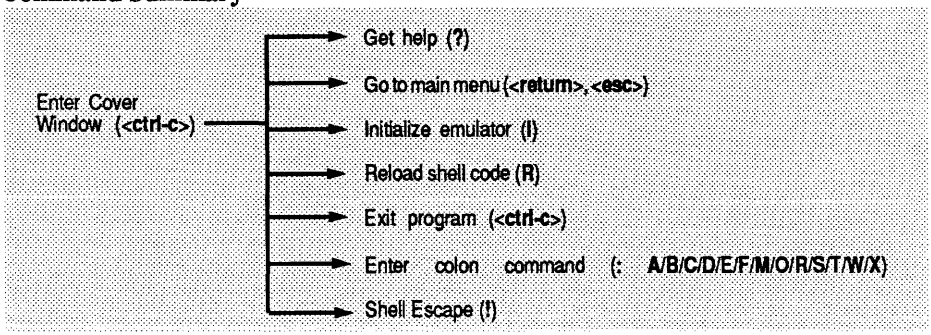
If you want to view the Cover window after using other windows, type **<ctrl-c>**. (Typing **<ctrl-c>** *from* the Cover window exits the control software completely.)

Troubleshooting Initial Startup

If the hardware and software are not communicating when you start the EL 800 software, you will not see the module types and sizes listed in the Cover window. In this case, and for more information, see the troubleshooting chart in Section 2, *Getting Started* (page 2-20).

If the control software cannot make contact with the emulator, you will see the error message "Emulator not opened: Ignore/Retry." At this point you can type **<ctrl-c>** to quit the control software, **I** to ignore the lack of contact with the emulator and start the control software, or **R** to try making contact again.

Command Summary



COVER WINDOW: INITIALIZE <ctrl-c>I

This window is used to reset the emulator configuration to its default state. The following table shows exactly what is affected during this emulator initialization, and the differences between using this command and a hardware reset (performed with the reset button on the emulator).

Differences between the Initialize Emulator command and a Hardware Reset

Initialize Emulator

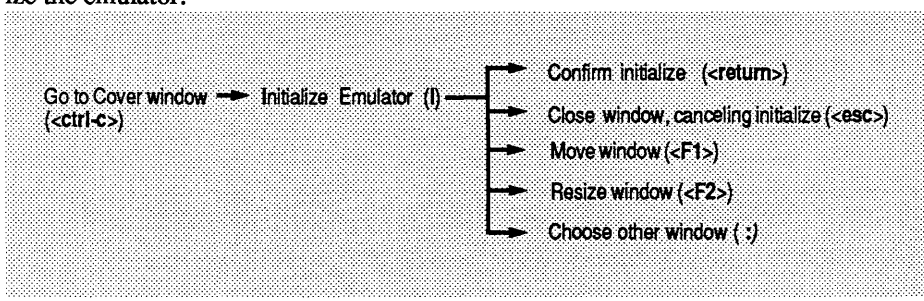
- Clears overlay map
- Sets counters X and Y to "stop counting"
- Sets trace control to "begin tracing" (position in trace buffer is not reset)
- Switches to Advanced Event System state 1 and initilizes AES
- Clears soft-switches to default state
- Loads new target PC vector
- Sets counters to zero

Hardware Reset

- Loads new target PC vector
- Clears position in trace buffer

Initializing and Resetting the Emulator

See Appendix D for complete information on what happens when you start and exit the software, turn power on to the hardware, turn power off, reset, and initialize the emulator.



Initialize Emulator Window

<< press **<return>** to confirm/**<esc>** to cancel

COVER WINDOW: RELOAD <ctrl-c>R

Description

Emulator shell code is retained in the battery backed up RAM in the emulator, and so does not need to be reloaded if emulator power is turned off.

You only need to reload the emulator shell code if:

1. you change to a new hardware processor module
2. you receive a software update

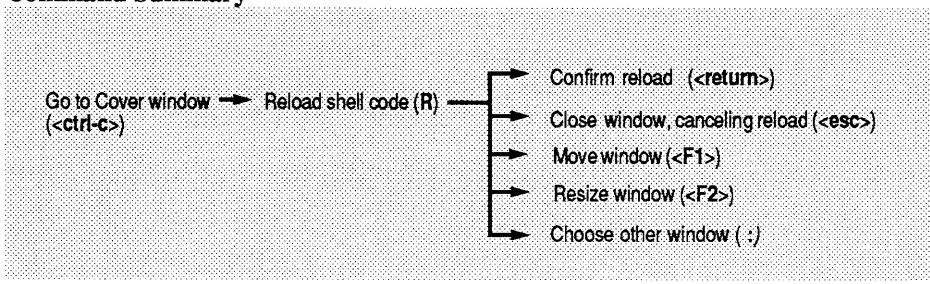
There are four files that are loaded each time you reload the shell code (plus the h64z.lca file with Z mask 64180 processors). *???* is either Z80 or 64180, depending on which processor you are using:

1. *???*1.shl Operating kernel
2. *???*2.shl Processor specific applications software
3. *???*.pod Pod processor software
4. *???*.lca Base unit software

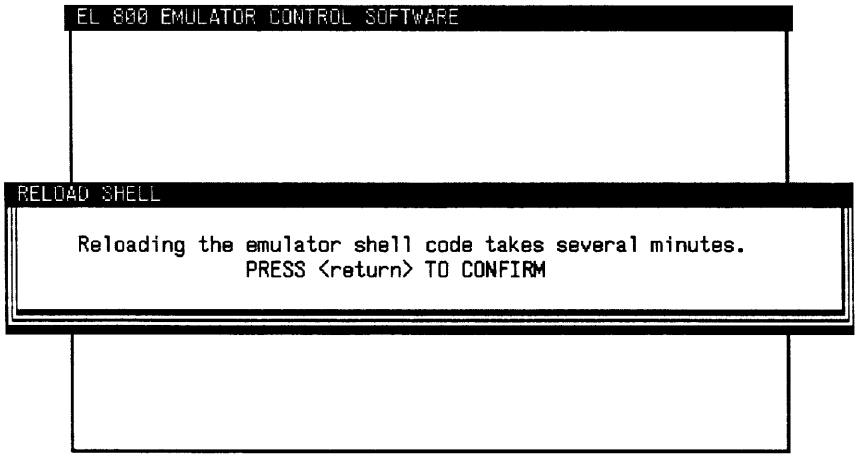
These files must be in the same directory as the executable file EL*???*.exe (see Section 2)

Reloading the emulator shell code takes approximately 5 minutes.

Command Summary



Reload Shell Code Window



_ << press **<return>** to confirm/**<esc>** to cancel

MAIN MENU

Description

The Main Menu provides a list of the available windows:

:A	Assembler	Line assembler, memory disassembler
:B	Break/Event	Advanced event system and basic breakpoints
:C	Configuration	Communications, emulator soft switches, shell escape, system processes
:D	Diagnostics	Target system diagnostics: RAM tests, scope loops
:E	Emulate	Go, step, reset emulator, set and clear event state variables
:F	File Access	Change directory; upload/download files; save symbol table, breakevent system, emulator switches, overlay mapping, trace and/or windows; change default file format; edit or view file; run make utility
:M	Memory Mode	Display and modify memory
:O	Overlay	Set up, map, and display overlay memory; copy data blocks between target and overlay
:R	Registers	Set, clear, and display CPU registers
:S	Symbol Table	Add, delete, find and display symbols
:T	Trace Display	Display raw or disassembled trace data
:W	Watch	Dynamic display of memory locations, symbols, and registers
:X	Exit Program	Return to DOS

There are three ways to go to any window from the Main Menu:

1. Type the first letter of the window name.
2. Use ↑ and ↓ to highlight the window name and press **<return>**.
3. Type a colon (:) followed by the first letter of the window name.

Main Menu


Applied Microsystems Corporation - Z80 - Main Menu Type ? for Help
A Assembler
B Break/Event
C Configuration
D Diagnostics
E Emulate
F File Access
M Memory Mode
O Overlay
R Registers
S Symbol Table
T Trace Display
W Watch
X Exit Program
<< enter window selection

EXPRESSION ANALYZER <space>

Description

The expression analyzer accepts hexadecimal, decimal, octal, and binary numbers, symbols, and math operators. It also understands most of the arithmetic operators of the C language. The expression analyzer can be called from any prompt that expects a numeric value by pressing the space bar.

Hex numbers (default) must start with a numeric digit, and *should not* have a following H. To specify numbers in other radixes, prefix the value with a zero (0) and one of [**X, L,O, N**], i.e.:

0X*nnnn*	hexadecimal (0*nnnn* also works)
0L*nnnn*	decimal (*.nnnn* also works)
0O*nnnn*	octal
0N*nnnn*	binary

The expression analyzer recognizes the standard C language operators. Appendix F, *Using Expressions*, contains detailed information about precedence and use in the expression analyzer.

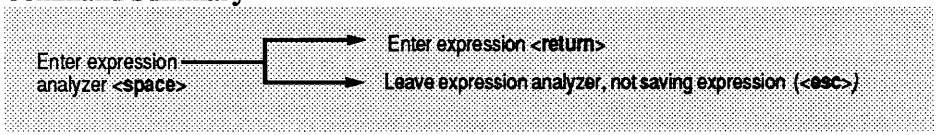
After an expression has been entered, press **<return>**. The expression will be evaluated, and the result will be inserted on the prompt line. Note that the result is shown as the least significant 32 bits of the evaluation, with leading zeros suppressed.

You can also see the value of a register by using the expression analyzer, or use the value of a register in an expression. For instance, entering

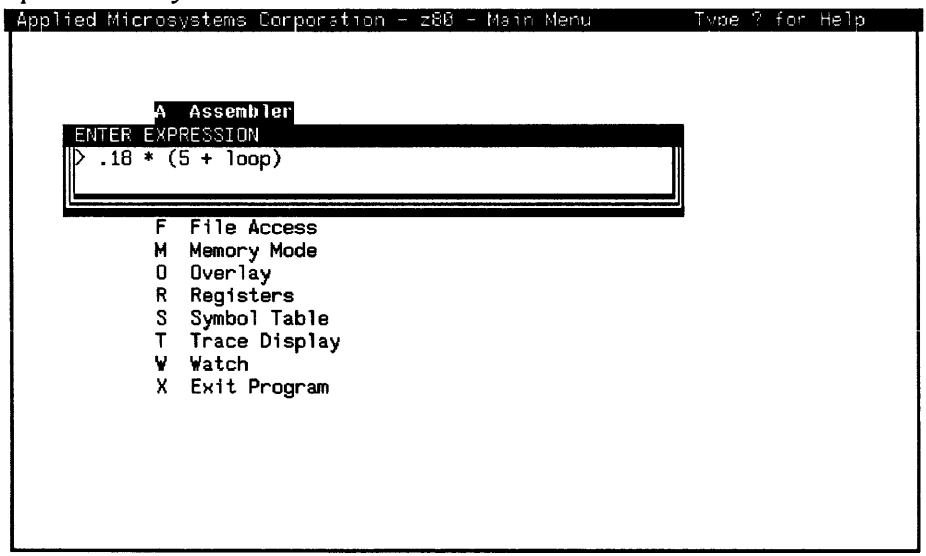
```
*((WORD *) SP +4)
```

yields the word residing 4 bytes down in the CPU's stack. See the *Registers Window* section for more information on displaying register values. For more information on the expression analyzer and expressions it recognizes, see Appendix F.

Command Summary



Expression Analyzer Window



Examples

.18 * (5 + loop)	The value is the sum of 5 (hex) plus the value of the symbol "loop" multiplied by 18 (decimal).
(WORD)123 + 1	Add one to the two byte value stored at address 123. If the value at 123 is F0FF, this expression would be equal to F100.

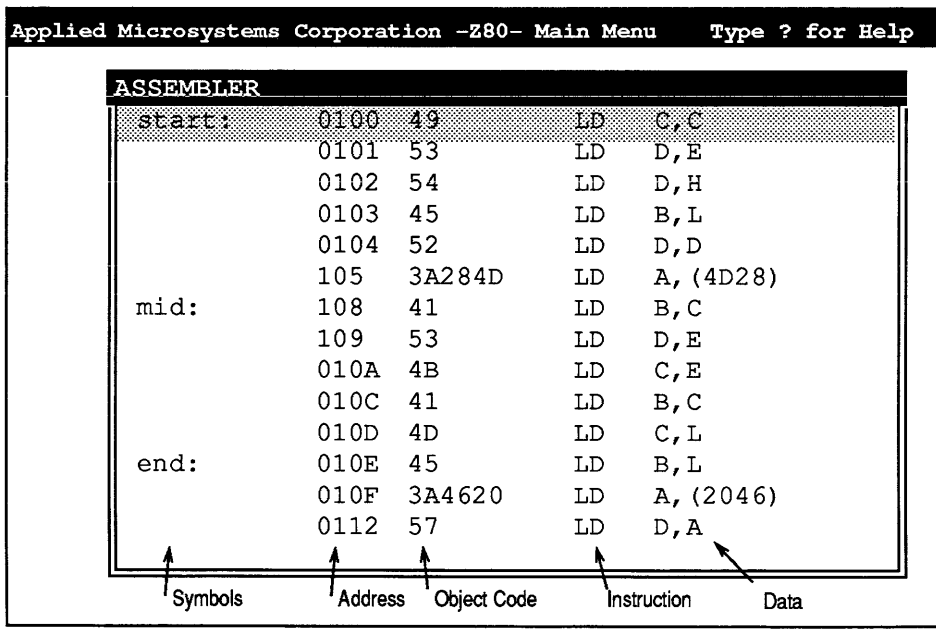
ASSEMBLER WINDOW : A

Description

This window lets you view disassembled memory and assemble lines of code into target or overlay memory space, depending on overlay mapping. Enter the address where you'd like to start viewing your assembly code and press **<return>**. If no address is given, the disassembly returns to your last position in disassembled memory.

The initial prompt "enter beginning address" expects a hexadecimal address. You can use the expression analyzer to enter an expression for this address using symbols, numeric data, and C language and math operators by pressing the space bar. (See the *Expression Analyzer* section for more information.)

Assembler Window



_ << enter line to be assembled

When the disassembled memory is on the screen, you can use the line assembler to change any line. The syntax for entering the line to be assembled is:

[*label:*]*instruction data*

The instruction you enter will write over the highlighted instruction. You do not have to type in the label.

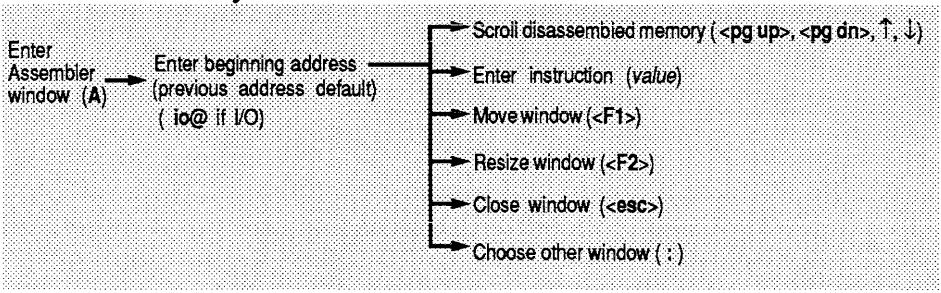
label	Case sensitive. Labels do not need to be in the symbol table; using a label here automatically adds it to the symbol table, overwriting it if it already exists. To remove a label, you must go to the Symbol Table window and type **D** for the delete symbols prompt (**:SD**).
instruction	Must be valid syntax for your microprocessor.
data	Can be any value acceptable by the expression analyzer (see that section). This includes hexadecimal addresses, decimal addresses, symbols, or expressions. Hexadecimal numbers must start with a digit, and must not have a following H.

When you press **<return>**, the new line is assembled and the emulator tries to write it to the highlighted address. This address must be RAM or overlay with write permission set (*not* ROM) or the new line cannot be written to memory.

If your program resides in overlay memory, the new line will be written to overlay memory. If you do not turn off the emulator power, your program remains in memory even if you exit the EL 800 control software and so remains available for your next emulation session. If you do exit the control software and subsequently restart it, do not initialize the emulator unless you want to clear the overlay memory map.

You can display any type of memory space supported by your processor. For the Z80 and 64180, the choices are I/O or memory. To display I/O space, type io@ immediately before the starting address. To switch back to memory space, type mem@.

Command Summary



Examples

start: ld a,b	Enter instruction "ld a,b" and label this line with the symbol "start."
ld A,(main)	Load accumulator A with the contents of the memory location indicated by the value of "main."

See Also

Expression Analyzer, page 6-15

Appendix F

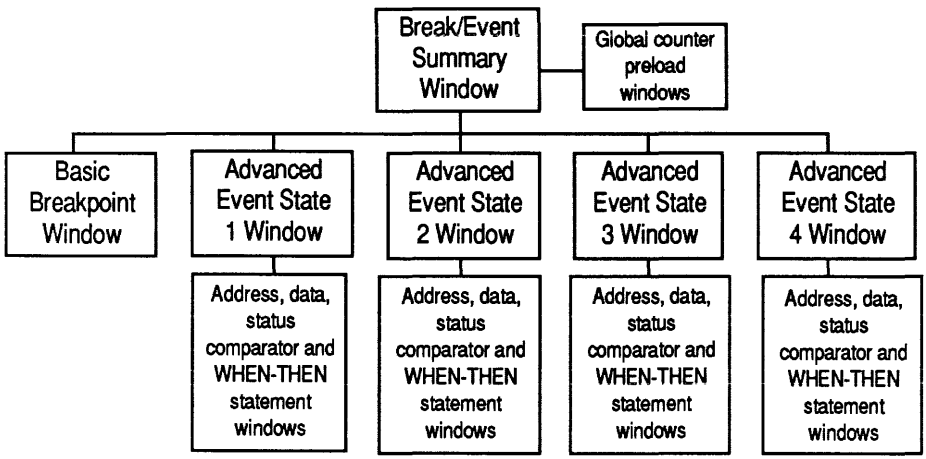
BREAK/EVENT SUMMARY WINDOW :B

Description

The Break/Event window provides access to two separate breakpoint systems:

- Advanced Event System (AES)
- Basic Breakpoint System (only available if you have an Overlay Module)

When you select the Break/Event window, a summary screen composed of several windows is displayed, showing basic breakpoints, an Advanced Event system summary, and counter values, if any. Each window can be separately configured, scrolled, positioned, or removed from the screen entirely. This flexibility allows you to enlarge interesting windows, reduce less interesting windows, and delete unused windows to meet your individual needs. This diagram shows the structure of the program windows:



These windows are described separately following this section.

The Break/Event window has five major functions:

Enter Basic Breakpoint window:

The Basic Breakpoint window is used to set up the Basic Breakpoint system. You must have at least one overlay module to use the Basic Breakpoint system.

Display WHEN-THEN summary:

Shows all the WHEN-THEN statements in the four state windows, if any. Each WHEN-THEN statement is numbered and shown in the group (one group per state) for which it is active.

Preload counters X and Y:

The global counter comparators are preloaded from the summary window. All the other comparators are set up within the four state windows, since there are unique sets of address, data and status comparators for each state.

Enter state windows:

Enter one of the four advanced event State windows. (See Section 3, the *State Window*, and the *WHEN-THEN Window* sections for information on setting up the WHEN-THEN statements used to define a state.)

Clear ALL events and breakpoints:

The Clear option clears the Basic Breakpoint System, the Advanced Event System comparators, and all WHEN-THEN expressions from all four state windows. It also resets counter X and Y to 0, sets the event state to 1 and turns trace on. *There is no undo function to restore a cleared setup,* but you can save the setup before clearing it with the **:FSB** command.

Basic Breakpoint System

The Basic Breakpoint system is used for breaking on addresses and ranges. You must have at least one Overlay Module to use this system. When you set a basic breakpoint, it is visible from the main Break/Event window. The limit to the number of basic breakpoints you can set depends on the available memory in your host computer. You can set ranges of breakpoints to eliminate the need for multiple breakpoint entries.

Advanced Event System

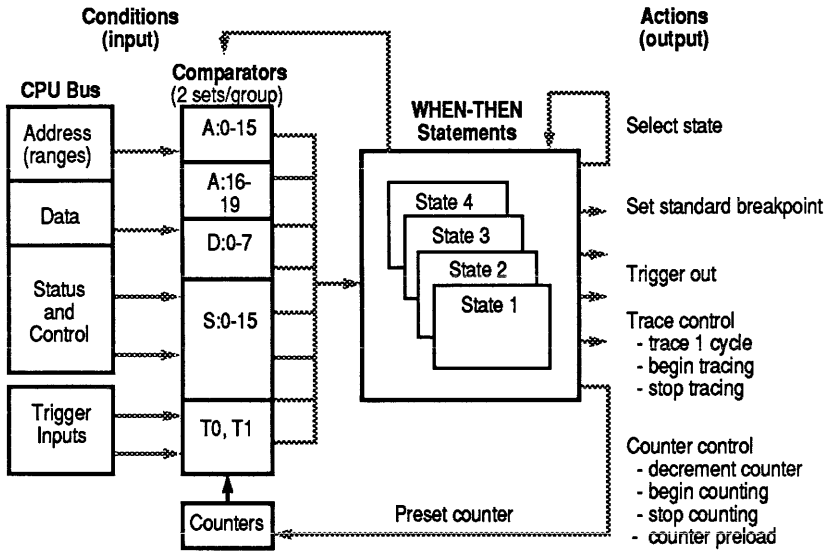
The Advanced Event System provides flexible system and breakpoint control, allowing you to isolate or break on a predefined series of events, and then perform various actions. You use the Advanced Event System from the State windows. Each event is specified in the form:

WHEN *conditions* **THEN** *actions*

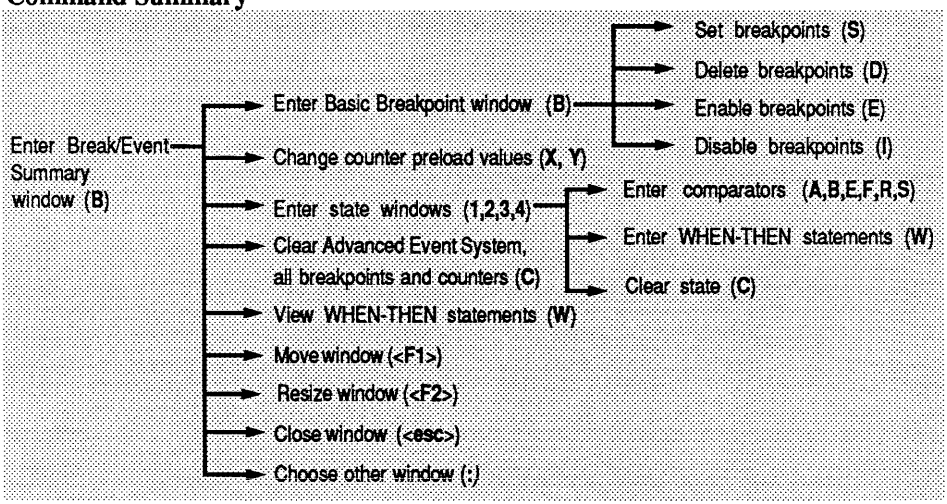
Conditions are logical combinations of address, data, status, count limits, and trigger inputs. *Actions* are combinations of breaking emulation, trigger outputs, trace control, counter control, and state switching. There are four independent sets of comparators: one set for each state window.

You can see a summary of the events set up in each State window on the Break/Event Summary window.

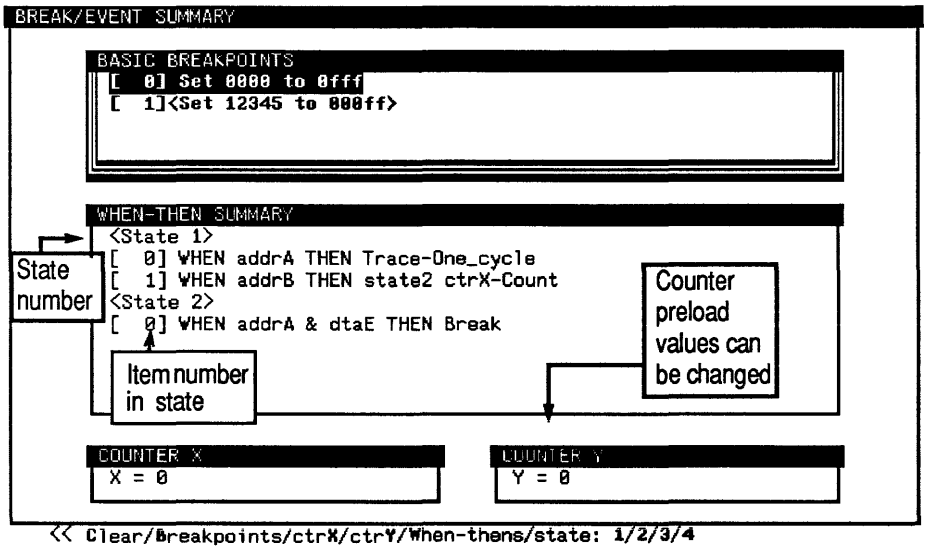
The diagram below shows the structure of the Advanced Event System.



Command Summary



Break/Event Summary Window



Deleting Items

In general, **Clear** is used to clear several related configurations at once, such as the whole breakevent system or an entire state. **Delete** is used to clear individual elements in a given window. To delete all the items from the current state window:

1. Type **C** for clear.

To delete items from comparator or WHEN-THEN windows:

1. Activate the window.
2. Type **D** for delete.
3. Specify the item number in the window. You can also simply highlight the item to be deleted using the cursor control keys. For address comparators, data comparators and WHEN-THEN expressions, you can type an asterisk (*) to indicate all items in the window.
4. Press **<return>**.

For example to delete address comparator 3 from the address comparator A subwindow, type **D3<return>**.

To escape from the Delete prompt, type **<backspace>** or **<esc>** to return to the previous prompt.

See Also

Appendix G, *Debugging Multiprocessor Systems*, for uses and examples of the Advanced Event system with other instruments and multiprocessor debugging.

BASIC BREAKPOINT WINDOW :BB

Description

The Basic Breakpoint window is used to set up the Basic Breakpoint system. You must have at least one Overlay Module to use this window. Once your breakpoints are set up, use the Emulate window to start running the emulator until you reach a breakpoint (**:EG**).

Use the Set command to set breakpoints, either single address or range breakpoints. Both types will break when either a read or write occurs to the address or range.

Each time the system prompts you for an address, it expects a hexadecimal value. To enter an address via an expression using symbols, numeric data, or math operators, press **<space>** to use the expression analyzer.

When you set a breakpoint, it is automatically enabled. To disable one or all breakpoints, use the **I** command. When a breakpoint is disabled, it will be ignored when you execute your code. The Go command in the Emulate window (**:EG**) stops only at enabled breakpoints.

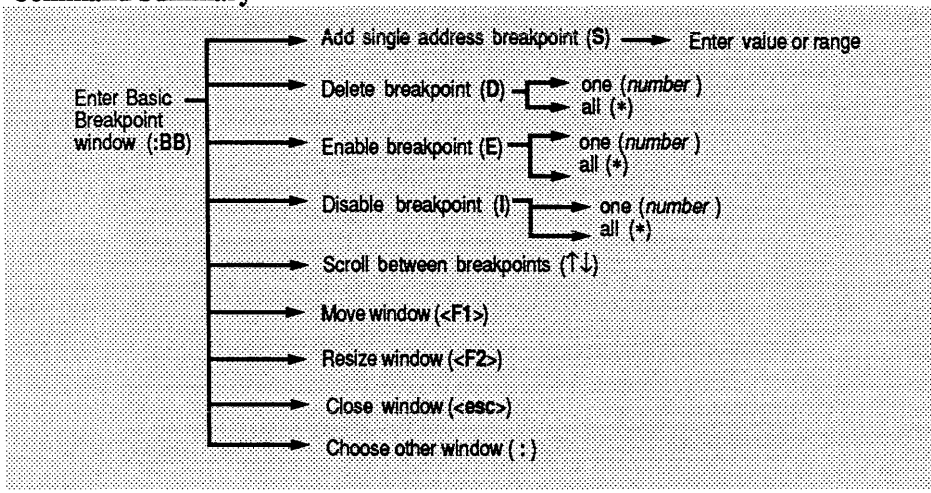
Enabled breakpoints are shown in bright type on the screen. Disabled breakpoints are displayed in half intensity, and are surrounded by angle brackets. For example, breakpoint 1 is enabled, and 2 is disabled:

```
[1] Set 0000 to 0fff
[2]<Set 12345 to 12400>
```

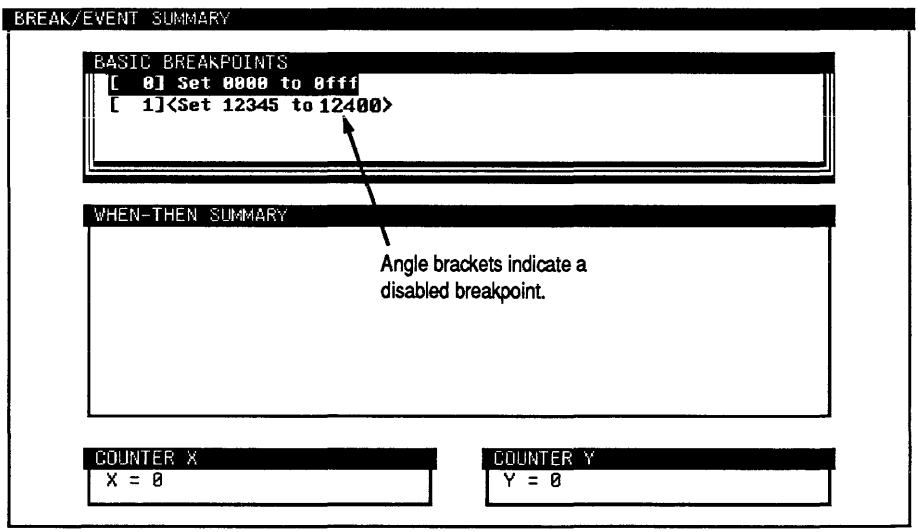
You can delete one or all breakpoints by typing **D** and either the index number of one breakpoint or an asterisk (*****) to specify all breakpoints. (You can also delete individual breakpoints by using the cursor keys to highlight the breakpoint you want to delete and typing **D <enter>**.) *There is no "undo" function to restore cleared breakpoints.*

When you leave this window, your breakpoints remain set until you exit the emulation session with **<ctrl-c><ctrl-c>** or **:X**. To save your breakpoint setting between sessions, or to save multiple breakpoint settings, use the File Access Save Breakevents command (**:FSB**). To restore a saved Basic Breakpoint system setup, use the File Access Restore Breakevents command (**:FRB**). See the *File Access* section for more information.

Command Summary



Basic Breakpoint Window



See Also

Expression Analyzer, page 6-15
File Access window, page 6-69
Section 3

X AND Y COUNTER WINDOWS :BX, :BY

Description

There are two global counters available for use in each of the four advanced event states. The ctrX and ctrY preload windows are used to load a value into the counters. Preloading a counter value is useful when you want to break or perform some other action when the counter reaches zero. The counters will decrement from your loaded value to zero per instructions you set up in the THEN portion of the WHEN-THEN statement (see page 6-43).

For example: **WHEN addrA THEN ctrY-Count**

WHEN ctrY THEN Break.

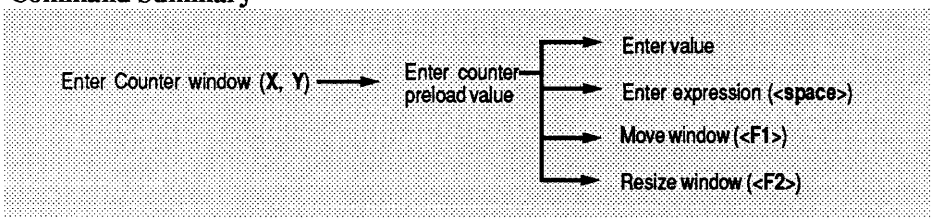
These statements cause counter Y to decrement on each access to the address in comparator A. When counter Y reaches zero, emulation breaks. In this way you can break emulation on the *n*th access to a specific address, where *n* is the preloaded counter Y value.

To view the actual value in the hardware register, use the event-state selection in the Emulate window (**:EE**).

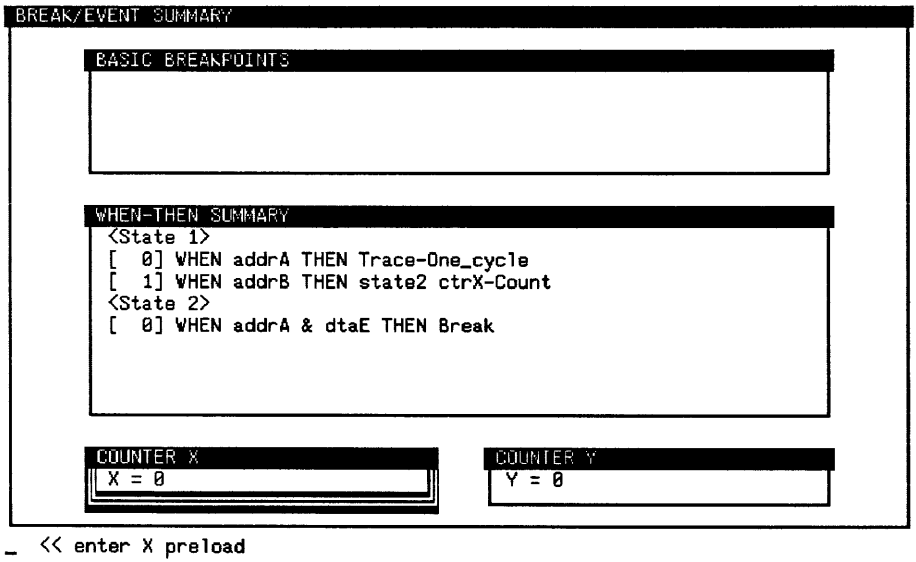
The global counter preset values can be viewed from any of the state windows (unless you have resized the windows and covered up the counter value display), but can only be changed through the Summary window. All the other comparators are set in the state windows, since there is a separate set of comparators for each state.

Each time the system prompts you for a counter preload value, it expects a hexadecimal value. To enter a value via an expression using symbols, numeric data, decimal numbers, or math operators, press <space> to use the expression analyzer.

Command Summary



X and Y Counter Windows



See Also

WHEN-THEN window, page 6-43

Expression Analyzer, page 6-15

Emulate window: Event-state, page 6-67

Section 3

STATE WINDOWS :B1, :B2, :B3, :B4

Description

There are four state windows, one for each *state*. A state is a combination of conditions and resultant actions defined by a series of WHEN-THEN statements. The four state windows are used to set up the powerful Advanced Event System. The existence of four separate states means that you can nest combinations of conditions and increases the effectivity of the comparators by four. It also lets you take several different actions, based on different criteria, during one emulating session. For instance, study this example:

STATE 1:

WHEN addrA & dtaE THEN Trace-Begin state 2

STATE 2:

WHEN stR & dtaE THEN Break

The state 1 WHEN-THEN statement specify that when the conditions set in address comparator A and data comparator E are met, begin tracing data and switch to state 2. The state 2 WHEN-THEN statement specify that when the conditions set in status line R and data comparator E are met (in state 2), break emulation.

In this example, emulation will only be broken if state 1 conditions are initially satisfied with subsequent satisfaction of state 2 conditions.

There are several steps to using the full capabilities of the Advanced Event System:

Step 1. Set up the address, data and status comparators for state 1 by typing the letter from the prompt line: e.g. A for addrA, R for statusR to activate the appropriate comparator window. The prompt line will guide you to type in the information required to set up each comparator. To set up global counter comparators ctrX and ctrY, you must first activate their windows from the Break/Event Summary window.

NOTE: When setting comparators, the system does not stay in set mode for multiple entries. Therefore, when setting more than one address, data, or status condition, you must type S to enter set mode once for each condition entered.

Step 2. Enter your WHEN-THEN expressions for state 1 (page 6-43). WHEN-THEN statements can take advantage of the comparator conditions you entered in Step 1. For instance, your WHEN-THEN statement could read:

WHEN addrA & stR THEN Trace-Begin

which signifies that when the condition you entered in address comparator A and the condition you entered in status comparator R are both met, begin trace. If you are debugging a condition requiring nested statements, you will reference another state as one of the WHEN-THEN statement actions.

Step 3. Set up the comparators and expressions in any other referenced states.

Step 4. Run your program using the Emulate window go command (:EG) (page 6-63).

To delete comparators or WHEN-THEN statements, activate the appropriate window and follow the prompt line. Note that when you delete address comparators, data comparators, or WHEN-THEN statements, you have a choice between deleting one item, or all items. To delete one item, highlight the item using the cursor control keys or enter the item's number. To delete all items in a window, type an asterisk (*) and press **<return>**.

You can also clear an entire state with the Clear command. This command clears all comparators (address, data, and status) and all WHEN-THEN statements in the current state. *There is no undo function for a cleared state.*

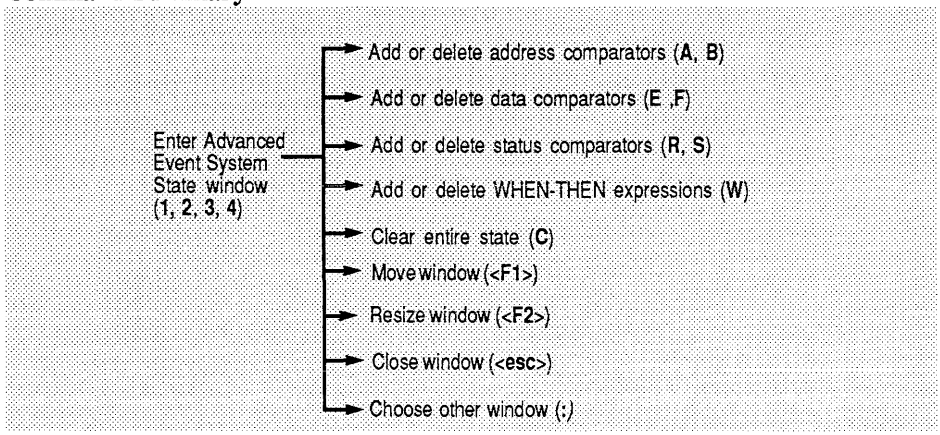
You can find further instructions for entering Advanced Event System commands in the following pages on comparators (address, data, and status) and WHEN-THEN statements. Practical examples of Advanced Event System use are contained in Section 3. Appendix G contains examples of using the Advanced Event System to debug multiprocessor targets..

Real-time modification of established events and breakpoints can only occur with the emulator stopped. By setting the REALTIME switch to OFF in the Configuration Emulator window (**:CE**), you can modify events and breakpoints while the emulator is running. However, emulation may occasionally halt to allow

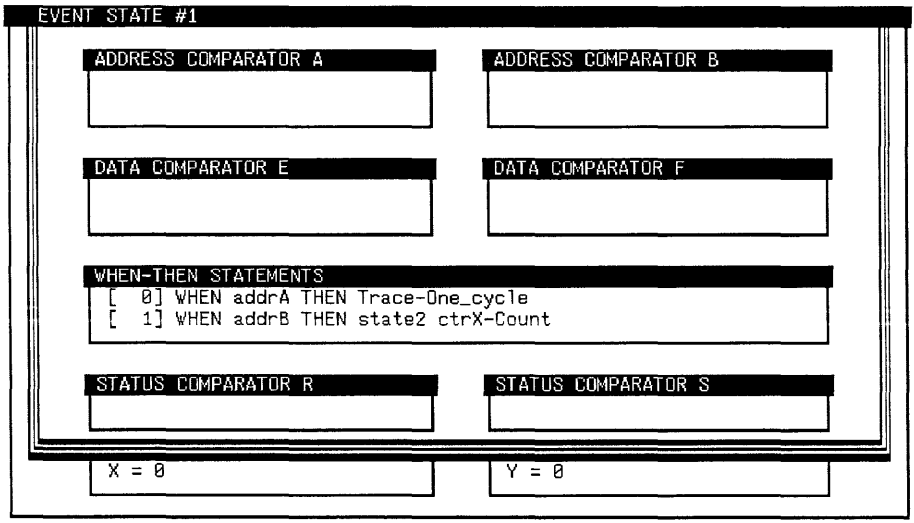
this capability, so real-time emulation is compromised.

To save your Advanced Event System setup in a file, use the File Access Save Breakevents command (**:FSB**). This command allows you to save different setups in separate files. To restore a particular setup, use the File Access Restore Breakevents command (**:FRB**).

Command Summary



State Window



See Also

Break/Event Summary window, page 6-20

File Access window, page 6-69

Section 3

Appendix G

STATE WINDOWS: Address Comparators :B*n*A, :B*n*B

Description

Address comparators are used to match specific address accesses. There are two address comparators in each state. The conditions in the address comparators are met if the address you enter in the comparator is encountered on the address bus during emulation. Used with a WHEN-THEN statement, you can perform any valid action (break, trace on/off, trigger, etc.) when the address comparator condition is met.

The address comparators in each state can be assigned specific addresses, a range of addresses, or values with *don't care* masks that function like wildcards, allowing you to match certain bit positions and ignore others. Typing **A** or **B** from the State window activates the appropriate address comparator window. Each comparator is a logical OR of the addresses, ranges, or don't care values specified in the window.

Each time the system prompts you for a comparator value, it expects a hexadecimal value. To enter a value via an expression using symbols, numeric data, decimal numbers, or math operators, press **<space>** and use the expression analyzer. Expressions using symbols are translated, and the hex address is displayed.

To enter a range, type a comma (,) after the starting address, and then enter the ending address. The EL 800 has 20 available address lines (A:0-19) for you to use (provided the processor supports that number).

To enter a don't care value for an address, type a semicolon (;) after the address (don't care values cannot be used with ranges). In the don't care mask, 1 signifies that the bit position is significant, 0 indicates that the bit position is a don't care. Therefore (assuming a 16-bit address range), to show that you care only about the values in bit positions 4-15 but are not concerned with the values

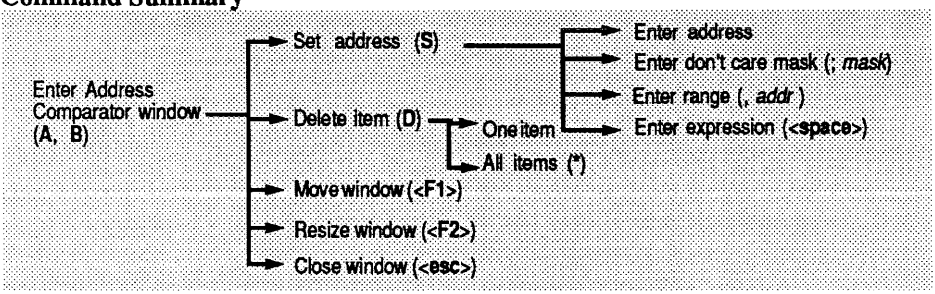
of bit positions 0-3, enter the address as: **100 ; ff0.** Such an entry would cause the comparator condition to be met on the following address accesses:

```
 100 to  10f
1100 to 110f
2100 to 210f
3100 to 310f
     .
     .
     .
f100 to f10f
```

To delete an address comparator, type **D** and specify a specific entry number or type * for all entries.

NOTE: The Z80 can only address 256 bytes of I/O space, so it doesn't make sense to set don't care masks for anything other than the lower 8 bits.

Command Summary



Examples

Your keystrokes	*What appears on the screen*
123,1F4	[1] Set 123 TO 1F4
1FFF;FFFE	[2] Set 1FFF DC FFFE

See Also

Section 5

STATE WINDOWS: Data Comparators :B*n*E, :B*n*F

Description

Data comparators are used to match specific patterns on the data bus. There are two 8-bit data comparators in each state. The conditions in the data comparators are met if the data you enter in the comparator is encountered on the data bus during emulation. Used with a WHEN-THEN statement, you can perform any valid action (break, trace on/off, trigger, etc.) when the data comparator condition is met.

In a data comparator, you can enter specific data, a range of data, or specify *don't care* masks that function like wildcards, allowing you to match certain bit positions and ignore others. Typing **E** or **F** from the State window activates the appropriate data comparator window. Each comparator is a logical OR of the data, ranges, or don't care values specified in the window.

Each time the system prompts you for a comparator value, it expects a hexadecimal value. To enter a value via an expression using symbols, numeric data, decimal numbers, or math operators, press **<space>** and use the expression analyzer. Expressions using symbols are translated, and the hex address is displayed.

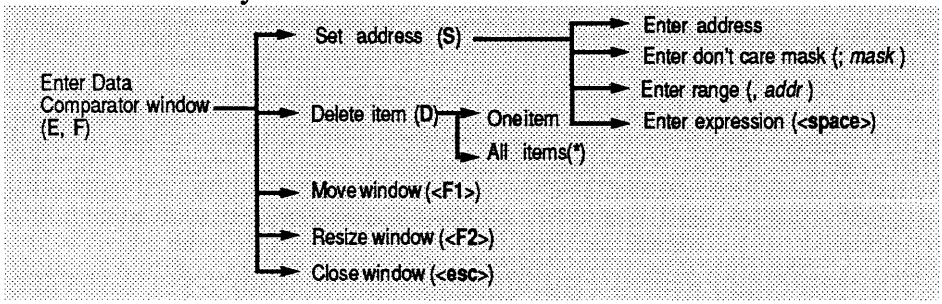
To enter a range, type a comma (,) after the starting address, and then enter the ending address.

To enter a don't care value for data, type a semicolon (;) after the data (don't care values cannot be used with ranges). In the don't care mask, 1 signifies that the bit position is significant, 0 indicates that the bit position is a don't care. The default don't care mask is **FF**. Therefore, if you are interested in all data patterns that have bits 0-5 set high, for example, you would enter: **3f ; 3f**. The first entry (3f - 0011 1111 in binary) determines whether the bits you are interested in will be high or low. The semicolon indicates that a don't care mask will follow, and the mask itself (3f) means that you only care about bits 0-5, and you don't care whether bits 6 and 7 are high or low.

If you wanted bits 0, 3, 4, and 5 cleared (low)and bits 1, 2, and 6 set (high) enter: **46 ; 7f**.

To delete a data comparator, type **D** and specify a specific entry number or type ***** for all entries.

Command Summary



Examples

Your keystrokes	*What appears on the screen*
1F;1F	[1] 1F DC IF
33	[2] 33

STATE WINDOWS: Status Comparators :B*n*R, :B*n*S

Description

Status comparators are used to match specific status signals entering the CPU. There are two status comparators in each state. The condition in the status comparator is met if the status signal you enter in the comparator is encountered during emulation. Used with a WHEN-THEN statement, you can perform any valid action (break, trace on/off, trigger, etc.) when the status comparator condition is met.

Each comparator is a logical AND of the status signals displayed in this window.

After typing an **S** to set a status comparator, there are two ways to select a status signal:

1. Highlight a status signal with the arrow keys, then press **<space>**.
2. Type the upper case letter in the status signal, or the leftmost number in the status signal: e.g., **Y** for an**Y**, **2** for fetch**2**_3.

To change a signal selection, you must first delete the current selection and then enter a new signal name (only one selection per window is possible).

Status comparators cannot be inverted within the comparator itself. To invert the polarity of a status signal, enter an exclamation point (!) before the status signal name in your WHEN-THEN statement. You can also OR status comparators together using the WHEN-THEN statement. For instance:

WHEN !stR | stS THEN Trace-Stop

This statement stops trace when the inverted signal defined in status comparator R OR the signal defined in status comparator S is encountered on the status bus.

To delete a status comparator, type **D** and press **<return>**.

The following status signals are available for each processor. Some of them combine multiple CPU status lines.

Z80 Status Signals

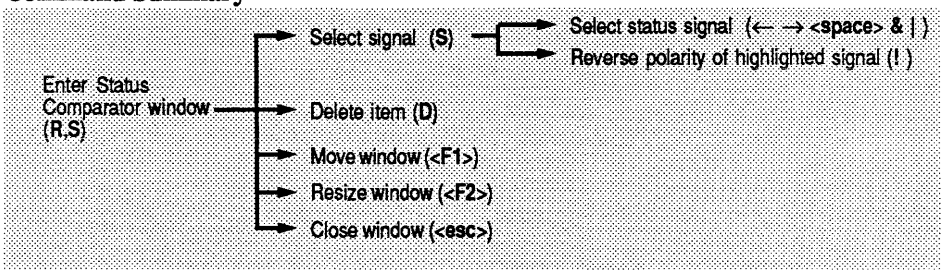
iOrq	Valid during I/O read or write operation, excluding interrupt acknowledge.
Data	Valid during any non-fetch data read or write operation.
fetch1	Valid during the first opcode fetch (first M1).
fetch2_3	Valid during the second or third operand fetch of multiple M1 instructions.
anY	Valid during any fetch: fetch1 or fetch2_3.
Memrq	Valid during a memory read or write operation.
inTack	Valid during an interrupt acknowledge cycle.
Wr	Valid during DMA, memory or I/O write operation.
Rd	Valid during DMA, memory or I/O read operation.
brQ	Valid when an external device requests the bus.
busacK	Valid when bus control is granted to an external device.
Int	Valid while an interrupt request is pending.
Nmi	Valid when a non-maskable interrupt is asserted low.
wAit	Valid when one or more wait states have occurred in current bus cycle.

64180 Status Signals

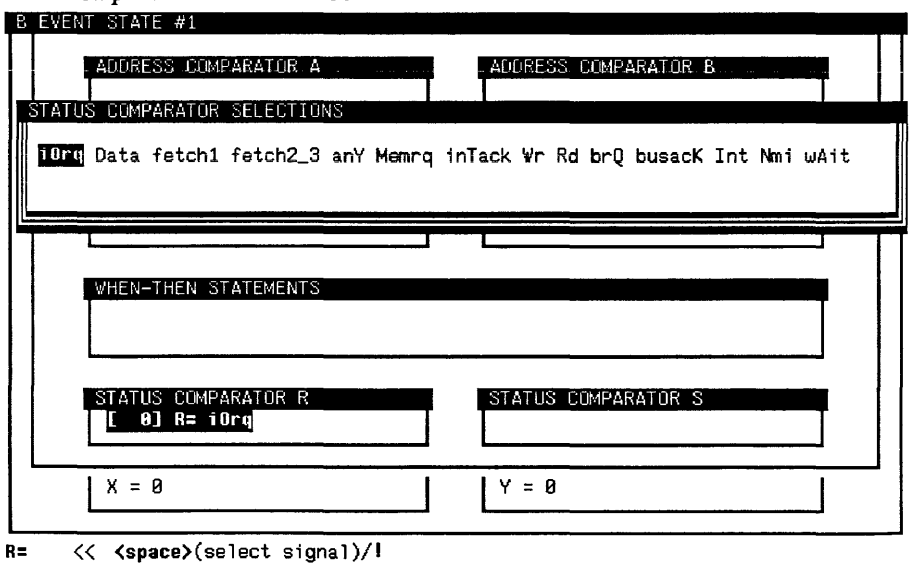
iOrq	Valid during I/O read or write operation, excluding interrupt acknowledge.
Data	Valid during any non-fetch data read or write operation.
fetch1	Valid during the first opcode fetch (first M1).
fetch2_3	Valid during the second or third operand fetch of multiple M1 instructions.
anY	Valid during any fetch: fetch1 or fetch2_3.
Memrq	Valid during a memory read or write operation.
inTack	Valid during an interrupt acknowledge cycle.
Wr	Valid during DMA, memory or I/O write operation.
Rd	Valid during DMA, memory or I/O read operation.
busacK	Valid when bus control is granted to an external device.

Int1	Valid while an interrupt 1 request is pending.
Int2	Valid while an interrupt 2 request is pending.
Int3	Valid while an interrupt 3 request is pending.
Nmi	Valid when a non-maskable interrupt is asserted low.
wAit	Valid when one or more wait states have occurred in current bus cycle.

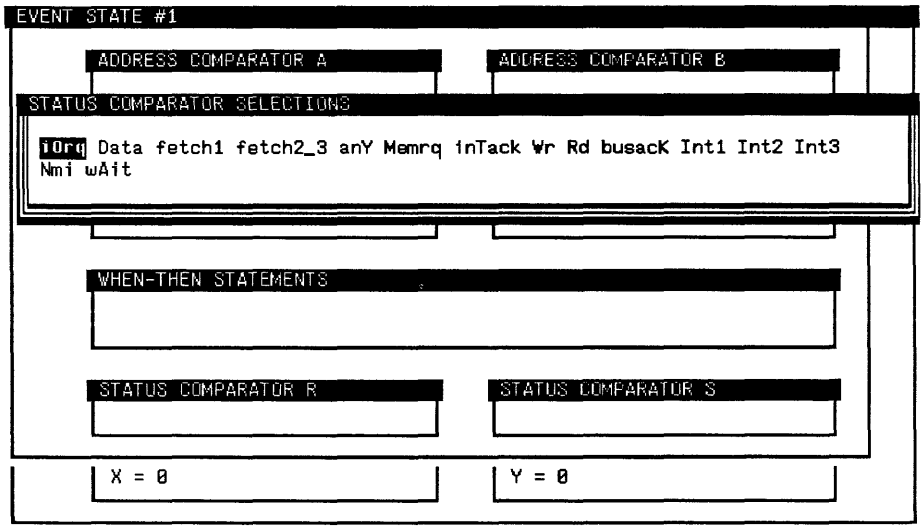
Command Summary



Status Comparator Window - Z80



Status Comparator Window - 64180



See Also

Section 3

Section 5

Appendix G

STATE WINDOWS: When-Then Statements :B*n*W

Description

The Advanced Event System uses WHEN-THEN statements to specify program states and desired actions. You may use an unlimited number of WHEN-THEN statements to meet your desired objectives. WHEN-THEN statements are entered in the following form:

WHEN *conditions* THEN *actions*

Conditions can be one or more of the following:

Address comparators	**addrA**, **addrB**
Data comparators	**dataE**, **dataF**
Status comparators	**statusR**, **statusS**
Global counter comparators	**ctrX**, **ctrY**
Trigger inputs	**trigIn** (trigger 1 or 2)

Conditions can be combined with the following operators: AND (&) and OR(|). The AND symbol (&) has higher precedence than the OR symbol (|). Negation can be specified using an exclamation point (!). For example: **addrA | !addrB** means when address comparator A **OR NOT** address comparator B.

You can select from many actions to be triggered by specified conditions. Once you have specified the conditions you are interested in, type T (for Then) and the prompt will display a list of actions. The actions can be one or more of the following:

Break emulation	**Brk**
Trace control	**Trace** (**O**ne cycle, **B**egin or **S**top)
Counter control	**X,Y** (**B**egin, **S**top, **P**reload, or **C**ount)
Trigger outputs	**trigOut** (**trigger1** or **trigger2**)
Switch states	**st1**, **st2**, **st3**, **st4**

No punctuation or operators are necessary to combine actions, as they are automatically ANDed together. Type the letter or number of each action and the prompt line will automatically separate them with a space. The length of a WHEN-THEN statement is limited to the length of the prompt line, about 77 characters. If you have a set of conditions or actions that exceeds this length, you can enter as many as possible on one line, then form a new WHEN-THEN statement on the next line that describes the remaining conditions and actions.

Entering Conditions

For complete descriptions of how to specify data, status, address and counter comparator conditions, see the comparator descriptions on pages 6-35 to 6-42.

To use the two trigger inputs, connect a wire from the trigger inputs on the base module of your EL 800 (labeled IN1 and IN2) to a signal on your target board or to another instrument, such as a logic analyzer or emulator. For details, see Appendix G, *Debugging Multiprocessor Systems*.

Entering Actions

The actions are described in detail below.

Break: Break emulation.

Trace: There are three trace actions possible:

One-cycle	Trace only the specified cycle.
Begin	Turn on trace.
Stop	Turn off trace.

If you specify two trace actions on the same bus cycle, the priorities are as follows:

Begin, trace One-cycle, Stop

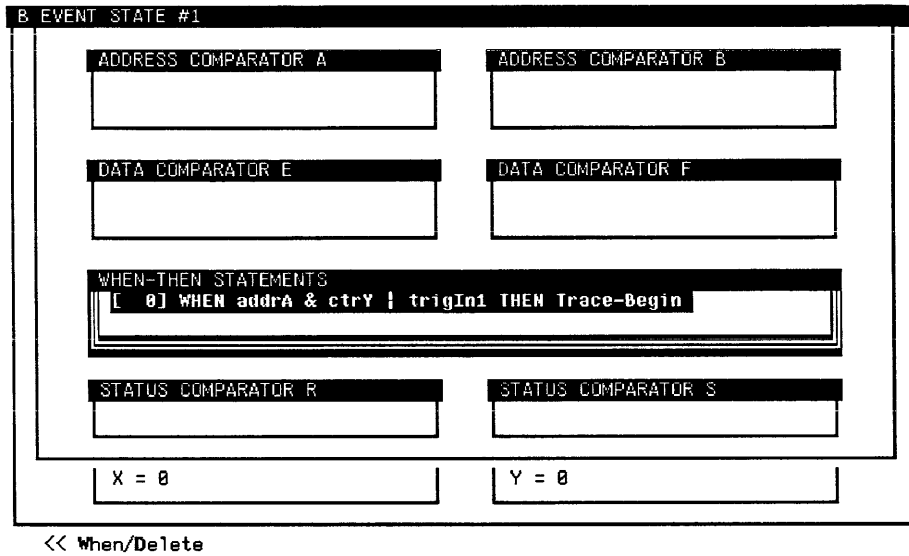
Trigger Outputs: There are two triggers outputs which can be used to trigger a logic analyzer, oscilloscope or another Applied Microsystems emulator. The connectors for the two triggers are on the left side of the base module, and are labeled OUT1 and OUT2.

Counters: There are four counter actions possible for each counter:

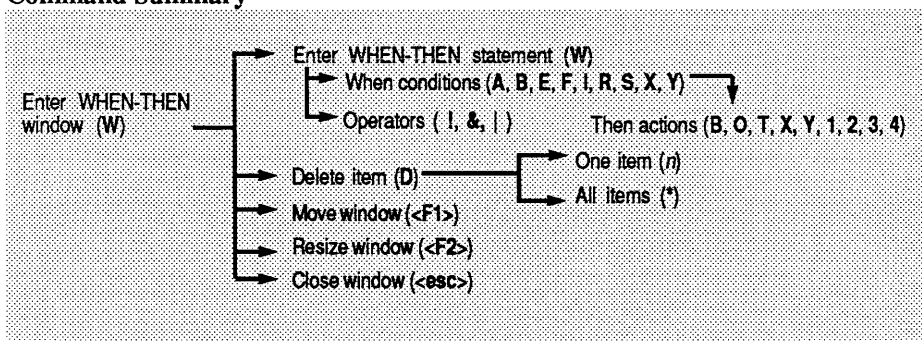
Begin	Begin counting
Count	Decrement counter
Stop	Stop counting
Preload	Set counter to preload value

Note that the counters are global, so changes to the counter in one state will affect the counters in other states. To view the current counter values, use the Event-state selection in the Emulate window (**:EE**).

WHEN-THEN Window



Command Summary



CONFIGURATION WINDOW :C

Description

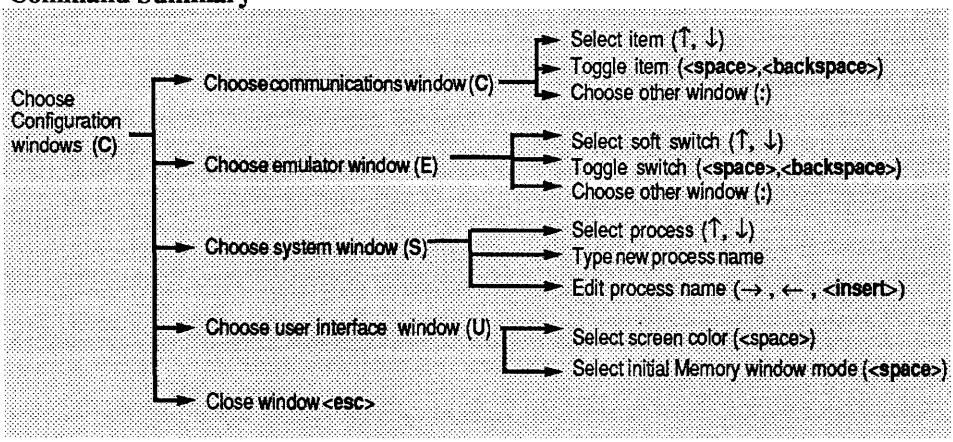
The Configuration window is used to set four separate types of configuration:

Communications:	Device, port, baud rate, and IRQ setting
Emulator:	Emulator soft switches
System:	Shell escape, edit, view and make utility names
User Interface:	Screen color(s), initial Memory window mode selection

To choose one, press **C**, **E**, **S** or **U**. Each of these selections brings up a new window, and each of these windows is separately explained on the following pages.

Changes made in these windows are saved in the configuration file *???*.CFG (*???* = Z80 or 64180) when you exit the window, so are preserved between sessions.

Command Summary



CONFIGURATION WINDOW: Communications :CC

Description

The Communications window is used to set the device type, port name, baud rate and IRQ value for the host. The IRQ value refers to the priority level of interrupts to the host processor.

To select a new value for an item, use the ↑ and ↓ to highlight the item, and then press <space> or <backspace> to toggle through the choices.

Your device type should be set to RS-232. The port name depends on which PC port you are using. We recommend that you set the baud rate to 19200.

Normal IRQ settings are:

com1	IRQ4
com2	IRQ3
com3	IRQ5
com4	IRQ7

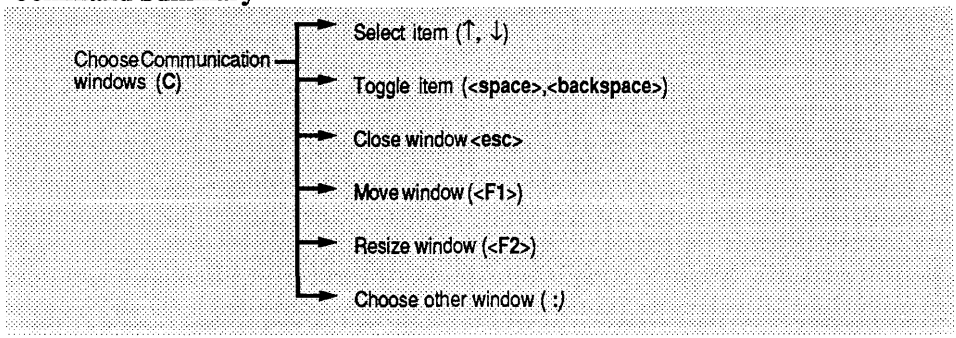
For some compatibles, IRQ settings may vary, so check your PC interface card manual for information on how to set the IRQ jumper. Parity is set to "no parity", and cannot be changed.

Never set identical IRQ values for any two of your serial ports.

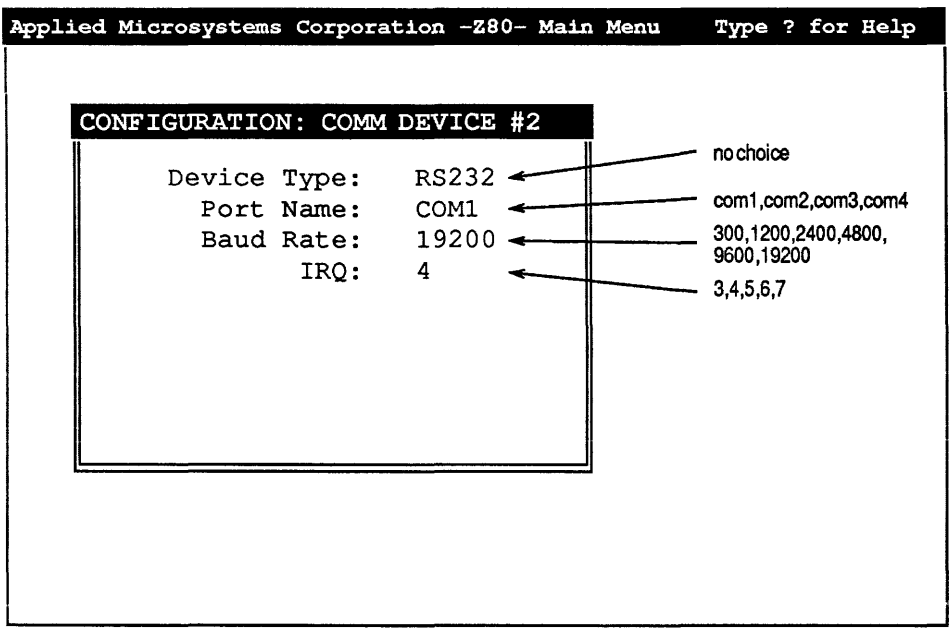
If you make any changes in the parameters in this window, the emulator will attempt to establish or re-establish communication with the emulator when you deactivate the window.

Your changes to the communications information are saved in the *???*.CFG file (*???* is either Z80 or 64180) when you exit the window. These settings will be used the next time you start the EL 800 control software.

Command Summary



Communications Window



See Also

Section 2

Appendix B: Cable Information

Appendix D: What Happens When

CONFIGURATION WINDOW: Emulator :CE

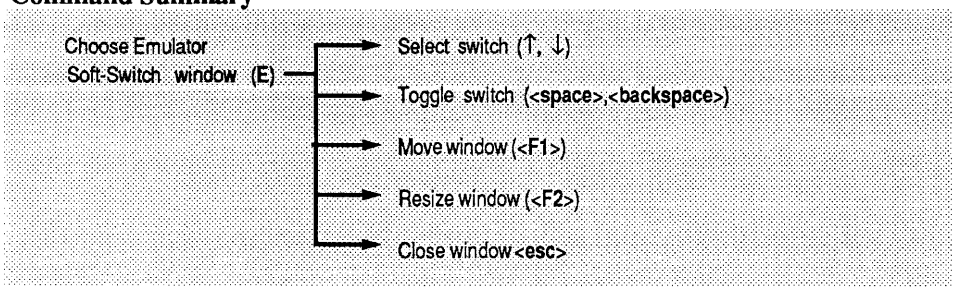
Description

This window lets you set several soft-switches used to configure your emulator. There are two types of switches: system control and processor control. System control switches are the REALTIME and RESET switches; all others are processor control switches, since they affect the ability of certain signals to reach the pod's CPU. The window also displays the type of data and program space you are using for memory accesses. (For some processors, you have a choice of data and program space. With the Z80 and 64180, memory space is the only option, so you cannot change these fields.)

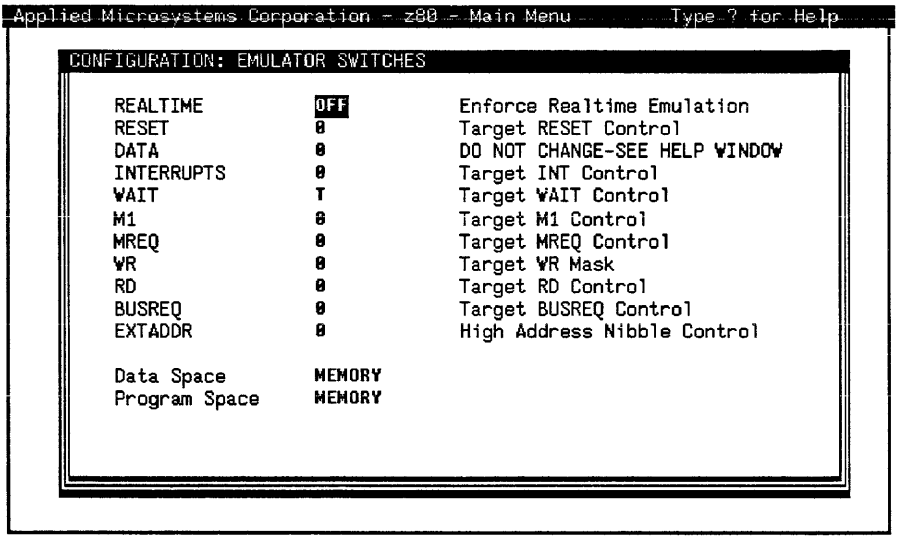
Note that there is some interaction between the switch settings. For example, it would not make sense to have DATA set to 1 and WR set to 0.

You can save different emulator switch configurations with the Save command available from the File Access window. These settings can also be easily restored. Please see the *File Access Window* portion of this section for details.

Command Summary



Emulator Window



System Control Soft-Switches

REALTIME This switch determines whether real-time execution is guaranteed by the emulator. To execute some commands in run mode (such as memory read and write commands), the pod CPU must be halted briefly. Using the REALTIME switch, you can prohibit asynchronous halting of the processor during run mode.

OFF OK to execute commands which impact real-time execution in run mode. In this position, your target may be briefly halted (with no indication to you) to execute commands. Use the ON setting if your target hardware or software is vulnerable to being asynchronously stopped.

ON With this setting, if you try to execute a command that impacts real-time execution, an error message is displayed and the command is canceled.

RESET This switch determines how target generated reset pulses affect the emulator CPU.

0 Setting the RESET soft-switch to "0" means that target generated resets will not be seen by the emulator CPU in pause mode. This is the default condition.

1 Setting the RESET soft-switch to "1" means that target generated resets *will* be seen by the emulator CPU during run and pause modes. Be aware, however, that allowing target resets to the pod CPU in pause mode can cause the pod to be unexpectedly reset. If this happens, the pod may be interrupted while responding to an emulator command. *If this occurs proper operation of the emulator cannot be guaranteed.*

DATA *CAUTION:* Setting this switch to 2 can cause bus contention problems and damage to target and emulator circuits under some conditions. See *RETI Considerations* in Section 5 for more information.

This switch controls data passage between the target, target data bus, the overlay memory, and the pod CPU. With the DATA switch set to 1, you probably want to also set the WR switch to 1. If you don't, the data is written to your target but may not necessarily match the data written to overlay.

0 Pass data to/from target and overlay in run mode except during reads from overlay. This is a safe setting to use while running code out of overlay, since it avoids bus contention.

1 Pass data to/from target and overlay in run mode except during overlay reads or writes. This is a safe setting to use while running code out of overlay, since it avoids bus contention, but writes to overlay will not predictably modify target memory. This setting is useful when a target device may drive the data bus during a write cycle.

2 Pass data to/from target and overlay at all times in run mode, and never in pause mode. This setting should not be used while running code out of overlay until you have read *RETI Considerations* in Section 5. Bus contention may occur.

INTERRUPTS Target interrupt control. This switch determines whether target interrupts are passed to the emulator CPU during run mode. During pause mode and single target bus cycles (peeks/pokes), interrupts from the target are not passed to the pod CPU.

0 Pass interrupts from target in run mode.

1 Do not pass interrupts from target in run mode.

The interrupts soft-switch does not apply to non-maskable interrupts (NMI).

WAIT Overlay wait state control. This switch is used to specify the number of wait states to add to overlay accesses. Wait states are inserted by the emulator hardware only if you enabled wait states when you mapped the overlay segment, and the value of the WAIT soft-switch is between 1 and 4. This switch allows you to simulate slow memory devices.

Enter the number of wait states to be added for segments mapped with wait states selected (P*n*). The range is from 1 to 4. The default is "T" wait states, which means that the number of wait states used by the target is automatically used for memory accesses. If you wish to specifically run *overlay* with zero wait states, do not enable wait states when you map the overlay segment, and select any value for this switch *except* "T."

The following table shows the interaction between the WAIT soft-switch value and the wait state enable/disable selection made during overlay mapping:

Number of Wait States During Overlay Memory Access		
Value of WAIT soft-switch	Wait State Selection during overlay mapping	
	Enabled	*Disabled*
1	1	0
2	2	0
3	3	0
4	4	0
T	**T**	**T**
T - use target wait states		

M1 - (Z80) Target M1 control. This switch determines whether the M1 signal is passed to the target during single target bus cycles and run mode.

0 Always pass M1 to target.

1 Pass M1 to target only on single target bus cycles or in run mode.

LIR - (64180) Target LIR control. This switch determines whether the LIR signal is passed to the target during single target bus cycles and run mode.

0 Always pass LIR to target.

1 Pass LIR to target only on single target bus cycles or in run mode.

MREQ Target MREQ control. This switch determines whether the MREQ signal is passed to the target during single target bus cycles and run mode.

0 Always pass MREQ to target.

1 Pass MREQ to target only on single target bus cycles or in run mode.

WR Target WR control. This switch determines when the WR signal is passed to the target system. The WR signal is never passed to the target in pause mode.

0 Pass the WR signal to the target during single target bus cycles and in run mode.

1 Pass the WR signal to the target only on non-overlay target cycles. In this state, writes to overlay do not produce a write cycle in the target.

RD Target RD control. This switch determines whether the RD signal is passed to the target during single target bus cycles and run mode.

0 Always pass the RD signal to target.

1 Pass RD to target only on single target bus cycles or in run mode.

BUSREQ Target BUSREQ control. This switch determines whether the BUSREQ signal is passed to the target during run mode.

0 Always pass BUSREQ to target.

1 Pass BUSREQ to target only during run mode.

EXTADDR External high address control (Z80 only).

0 The Z80 only uses 16 address lines, and the emulator has 20 lines available. With this switch set to 0, the four extra external address lines are tied to 0. Unless you are using an external driver for the 4 additional address bits, leave this switch set to 0.

1 Enable the four spare address lines (A:16-A:19) for the external gripper clips that come with your Z80 probe module. These can be used for monitoring input from an MMU or bank switching device. Unconnected signals A:16-A:19 float to a high state.

See Also

Section 5

CONFIGURATION WINDOW: System :CS

Description

The System window is used to enter the path names of your favorite programs for escaping to the shell, editing a file, viewing a file and running a make utility.

The file name fields may be edited:

move cursor	→ ←
insert mode	**<insert>**
overstrike mode	toggle **<insert>** key (default)

You can get to the operating system by typing an exclamation point (!) from the Main Menu or any top level windows.

To run a system process, you use the File Access window:

Edit file	**:FE**
View file	**:FV**
Run make utility	**:FM**

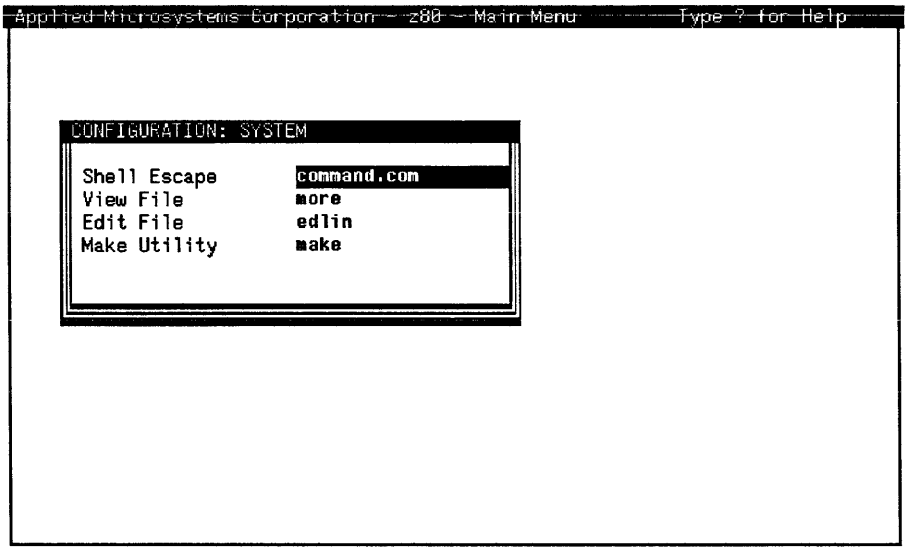
The EL 800 will attempt to load the new process and transfer control to it. When the external process finishes, control will be returned to this program.

SOME CAUTIONS WHEN USING THE SHELL ESCAPE:

1. If you use COMMAND.COM, you must type **EXIT** to return from DOS to the EL 800.
2. Beware of programs that use interrupts or the serial ports, since they may not restore the vectors used by this program.
3. Programs such as DIR should not be used for the shell escape, as they display the information too quickly to read.
4. If the invoked process installs a process that will remain resident after it exits (such as a printer or other device driver), then that process may interfere with the loading of other programs after this program exits.

Your changes to the system processes are saved in the *???*.CFG file (*???* is either Z80 or 64180) when you leave this window. These settings will be the default next time you enter the EL 800 control software.

System Window



See Also

File Access window, page 6-69

Shell Escape, page 6-5

CONFIGURATION WINDOW: User Interface :CU

Description

The User interface window is used to set defaults for variables that affect the interface between you and the control software: screen colors and the initial memory mode. You can set screen colors from a choice of four options:

Default This option uses a light blue highlight color which can be displayed on all types of monitors except LCD (liquid crystal display). Any invalid color entry in the color field results in the default colors being used. Also, if the color section is missing from the **.cfg** file the default colors are used. Colors are: light blue for the window information, forest green for the command line prompt, red for error messages, and yellow for user input.

Blink With this selection, the colors used are similar to the default colors except the cursor blinks so LCD monitors can also be used.

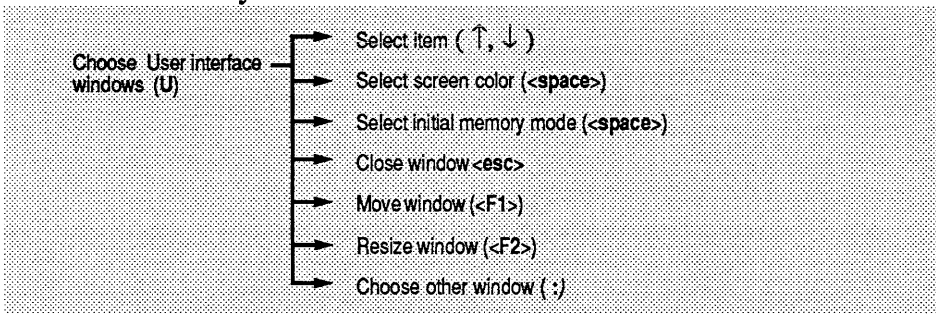
Color1 Prompt, help, error message, and user input colors remain the same as the default, and the highlight and window information colors are a deeper blue.

Color2 Prompt, help, error message, and user input colors remain the same as the default, and the highlight and window information colors are a gray consistent with monochrome monitors.

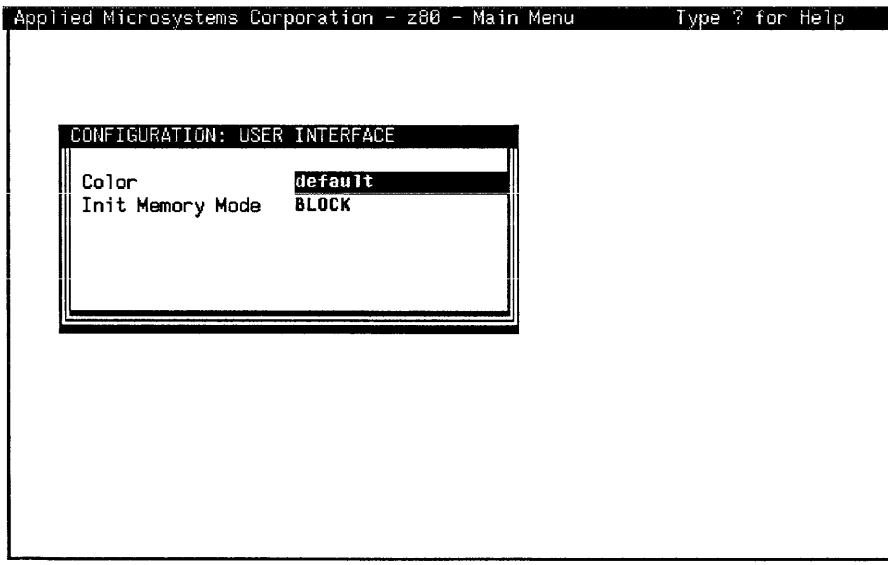
You also use this window to select an *initial* memory mode. When you first start the control software, memory mode is set to either BLOCK, LINE, or ASCII depending on your selection here. You can always change the memory mode by entering the memory window and using the Parameters window (**:M<return>P**).

Any changes you make in this window are automatically saved in the **.cfg** (configuration) file when you leave the window.

Command Summary



User Interface Window



DIAGNOSTICS WINDOW :D

Description

The Diagnostics window provides nine canned test routines for checking target hardware. You can only run one diagnostic at a time, and you cannot perform other target operations when using the diagnostics.

The RAM tests determines if your target or overlay RAM is responding properly to read and write accesses, and the scope loops can help you debug other hardware problems such as inoperative address, data, or control lines.

To use these diagnostics on your target, highlight the desired diagnostic with the cursor keys. Then, enter the starting and ending addresses, if needed. If the diagnostic is a write, enter the data to be written in the appropriate field. If the diagnostic is one of the RAM tests, enter the number of times you want the test to run in the COUNT field, or leave it at 00 for continuous running.

Start a diagnostic by pressing <return> while the cursor highlights the desired diagnostic. A blinking "RUNNING" message will show the test is active. You can press any key to halt the test.

If the test is successful, the "RUNNING" message disappears. If either of the RAM tests fail, an error message is displayed giving you the option of examining information on the nature of the error (**R**), continuing the test (**C**), or quitting (**Q**). Error information includes the error address, expected data, actual data, phase number, current loop number, and the loop number that failed. (The current loop might be different from the fail loop if you continued the test after an earlier failure.) Other tests report any encountered errors but continue running.

The tests are :

RAM TEST 1	A simple RAM test.
RAM TEST 2	A complex RAM test.
READ LOOP	A continuously looping read scope loop.
WRITE LOOP	A continuously looping write scope loop.

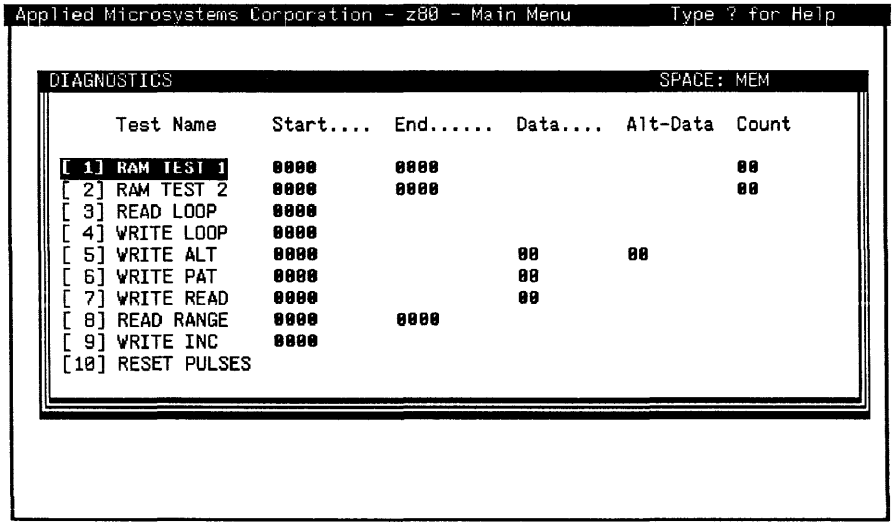
WRITE ALT	A continuous scope loop similar to WRITE LOOP except that data alternates between the two data patterns specified in the "Data" and "Alt-Date" fields.
WRITE PAT	A continuous write scope loop that rotates the data pattern by one bit in each iteration.
WRITE READ	This scope loop writes to the test address and then reads from that address.
READ RANGE	A continuous scope loop that reads data over a range of addresses, returning to the starting address each time the end of the range is reached.
WRITE INC	A scope loop that writes an incrementing data pattern to an address. The count data restarts at zero and the count is reset after the maximum count (determined by the grain size) is reached.

The columns shown on the screen are:

Start	The address to start the test at.
End	The ending address of the test.
Data	The data pattern you want written (write tests only).
Alt-Data	The alternate write data pattern (Write Alt test only).
Count	The number of times to perform the test on the given address range.

Use the cursor control keys to maneuver to the field you want to change.

Diagnostics Window:



Command Summary

Choose Diagnostic window (D)

- Select item (↑, ↓, ←, →)
- Select memory space (io or mem)
- Select test (<return>)
- Close window <esc>
- Move window (<F1>)
- Resize window (<F2>)
- Choose other window (:)

EMULATE WINDOW :E

Description

Use the Emulate window to control the target CPU. This window shows the current state of the CPU, a disassembled history of the most recently executed instructions, and a preview of the next few instructions. The cursor highlights the instruction at the current PC (program counter). The highlighted instruction has not yet been executed. The window header shows the emulator mode: running, stopped, or stepping.

It is common to also display the Registers window with this window, to allow quick display and modification of the registers. To simultaneously display the Registers window, you must first resize the Emulate window so the Registers window will fit. Of course, you may position both windows, or others, anywhere you choose on the screen using the move (**<F1>**) and resize (**<F2>**) keys.

Your choices in this window are:

Step through code (S) Step through your code one instruction at a time, a predetermined number of instructions, or continuously (but at a speed slow enough to view). The step option provides a way to watch register values or memory locations as each instruction is executed. Usually, the microprocessor executes code too quickly to watch the effects unaided. In this mode, one assembly language instruction is executed, and then the microprocessor is halted. (In continuous step mode, you must halt the processor by pressing any key.) As instructions are executed, they are disassembled and displayed in the Emulate window.

Execute the code (G) Run the target. If breakpoints are enabled, the emulator will automatically stop executing code when it reaches a breakpoint condition you have set with either the Basic Breakpoint System or the Advanced Event System. You can also run with *strobe* enabled, which stops the target at regular intervals to refresh all displayed windows with current data. With this option, you can constantly watch the progress of the program.

You can also use this command to reset the program counter or disable breakpoints before entering run mode. Change the PC by entering the new value before pressing **<return>**. Disable breakpoints by entering an asterisk (*) before pressing **<return>**. For example:

:G 1000*<return>

Note the asterisk (*). Using it in the Go command disables all breakpoints *for this command only*, so you can avoid going to the Break/Event system windows to disable breakpoints. Pressing **<return>** after this example starts code execution (without breakpoints) at the new PC.

Restart program (Z) This restarts your program with known Advanced Event System state variables: event state 1, trace control (on), counter control (off) and current counter values (set to preload comparator values). **Z** also resets the PC register to 0 and sends a reset signal to the pod CPU. The current Advanced Event state variables can be viewed in the Event-state window.

Event-state (E) View, change, and clear state variables used when you run your program using the **Go** command.

To stop emulation, press **<return>**.

Emulate Window

```
Applied Microsystems Corporation -Z80- Main Menu    Type ? for Help

EMULATE                    EMULATOR STOPPED
 120  START:   0000  310009     LD    SP,B1ST  [0900]
 117           0003  C30010     JP    ZERO  [1000]
 114  ZERO:    1000  3E00       LD    A,00
 112           1002  210F09     LD    HL,B2END  [090F]
 109           1005  0610       LD    B,10
 107  ZLOOP:   1007  77         LD    (HL),A          090F<00
 105           1008  2B         DEC   HL
 104           1009  05         DEC   B
 103           100A  C20710     JP    NZ,ZLOOP  [1007]
 100  ZLOOP:   1007  77         LD    (HL),A          090E<00
  98           1008  2B         DEC   HL
  97           1009  05         DEC   B
  96           100A  C20710     JP    NZ,ZLOOP  [1007]
  93  ZLOOP:   1007  77         LD    (HL),A          090D<00
  91           1008  2B         DEC   HL
  90           1009  05         DEC   B
  89           100A  C20710     JP    NZ,ZLOOP  [1007]
  86  ZLOOP:   1007  77         LD    (HL),A          090C<00

_ << Go/Step/Z-restart/Event-state/<return> to single step
```

The columns on the screen refer to:

Cycle # Symbols Address Object Code Instruction Data movements

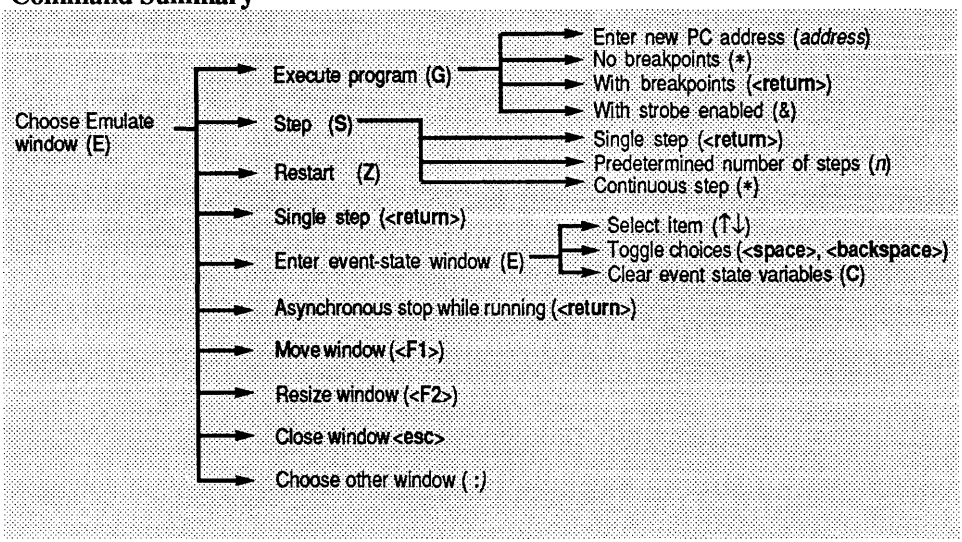
LDIR/LDDR (Z80 and 64180)

The LDIR and LDDR instructions have caused particular confusion for some users when debugging their code. LDIR increments a counter and loops the number of times specified in the first register pair. LDDR decrements a counter and loops the number of times contained in the first register pair. Due to these loops, you may see multiple LDIR or LDDR instructions in the Emulate window as you run your target even if only one is in your code. When either of these instructions is executed, 3 register pairs are loaded:

BC	size of block to be moved
DE	start of first block
HL	start of first block

These instructions will appear in the trace the number of times stored in the BC register.

Command Summary



EMULATE WINDOW: Event State :EE

Description

This window lets you view, change and clear the Advanced Event System state variables. The state variables are modified during emulation by Advanced Event System actions, and can be changed in pause mode with the **Z** command. (See the Emulate Window section for a description of how the **Z** command affects the event state variables.)

The state variables are listed below, with their default values:

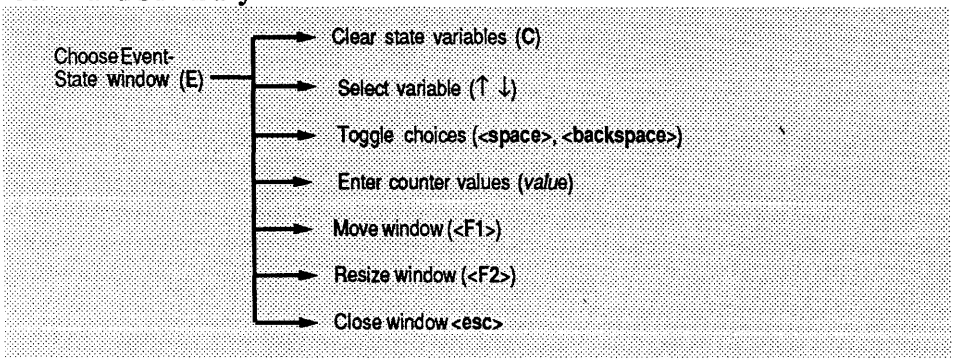
State:	**1**
Counter X:	**OFF**
Counter Y:	**OFF**
Trace:	**ON**
Counter X value:	**0000**
Counter Y value:	**0000**

Clearing the state variables sets them to the following:

State 1
Trace on
Counters off
Counter values set to counter preload comparator values

All variables may be changed from this window. To change the State, Counter X, and Y, or Trace variables, use the cursor control keys to highlight the variable you want to change, then press the space bar to toggle through the options for that variable. To change counter values, simply highlight the appropriate counter and enter the new value. (Change the preloaded counter values from the Break/Event summary window.)

Command Summary



Event-State Window

```
Applied Microsystems Corporation -Z80- Main Menu    Type ? for Help

EMULATE                    EMULATOR STOPPED
54          006B  D303      EVENT STATE VARIABLES
51          006D  3E54
49          006F  D302                State:  1
46          0071  3AE71F          Counter X:  OFF
42          0074  22E81F          Counter Y:  OFF
37          0077  E1                 Trace:  ON
34          0078  22DE1F    Counter X value:  0000
29          007B  22DC1F    Counter Y value:  0000
18          007E  2AE81F
13          0081  ED73D01F
7           0085  31D01F
4           0088  FDE5
0           0014  3078
0           0016  3032
0           0018  204E      JR    NZ,0068
0           001A  41        LD    B,C
0           001B  4D        LD    C,L
0           001C  45        LD    B,L

Event-state: _ << Clear state variables
```

FILE ACCESS WINDOW :F

Description

The File Access window is used for all utilities that manipulate files in the operating system environment. Using this window, you can change directories, upload and download code, edit and view files, use the make utility, and save and restore a variety of control software configurations and files. You can also change the upload/download format using the Parameters option.

The top line of the window shows the current directory and the upload/download format, and the files in the current directory are shown in the window. You select files in the window using the cursor keys, **<home>**, and **<end>** keys. To access files in another directory, you can either change directories (**:FC**) or type the desired filename (with full path information) and press **<return>**.

Change directory This option lets you display the files of a different directory in the File Access window. The change directory command can also be used to list files using * and % as wild cards. Type in the file names using the wildcards as desired and press **<return>**. Type .. to quickly change directories one level up.

Upload file Use this option to save a section of data in target or overlay memory (depending on overlay mapping) to a file. Select the format for the upload with the File Access window's Parameter command (**:FP**). You must specify the starting and ending address of the memory range, and a filename. If you enter the filename without an extension, a default extension indicating the current file format is automatically appended. If a file of this name already exists, a prompt asks whether you want to overwrite the file or stop the upload. (See Appendix C for more information on file formats.) The default extensions are:

.eth	Extended Tek Hex
.mot	Motorola S1, S2 or S3 records
.hex	Intel Hex
.tek	TekHex
.hit	Hitachi S records

To save the symbol table, use the Save command.

Download file This option allows you to download data from a file to the target or overlay memory. Select the format for the download with the File Access window's **P**arameter command (**:FP**). Enter the name of the file to be downloaded. If the filename is entered without any extensions, a default extension indicating the current file format will be automatically appended. The software looks for the file in the current directory, so if it is somewhere else, use the full path name.

Edit file To edit a file, use this option. Enter the name of file to be edited. The editor specified in the Configuration: System Processes window will be used (if available).

View file To view a file, use this option. Enter the name of file to be viewed. The viewing program specified in the Configuration: System Processes window will be used (if available).

Make utility This option runs the make utility specified in the Configuration: System Processes window. Enter the arguments required by the make file.

NOTE: If the utility name is "make", the control software appends a **-f** to this command. This option specifies that the make command use the file named after the **-f**, rather than the default "makefile".

Parameters Use this option to select the object module format used by your linker/loader utility. Your selected format will be used by the upload and download commands. (See Appendix C for more information on file formats.) The default extensions are:

.eth	Extended Tek Hex
.mot	Motorola S1, S2 or S3 records
.hex	Intel Hex
.tek	TekHex
.hit	Hitachi S records

Save

You can save all or part of your emulator configurations on disk files for use in later emulation sessions using this command. Options include:

- **B**reakevents
- **E**mulator switches
- **O**verlay
- **S**ymbols
- **T**race (raw or disassembled)
- **W**indows
- ***A**ll (except trace and symbols)

Each option saves only the appropriate portion of the emulator configuration. The All (*) command saves your breakpoint, emulator switches, overlay mapping, and window configurations, but not symbols or trace. With the Save Trace command (**:FST**), you must enter the start cycle, number of lines, and whether to save trace in raw or disassembled form. Trace is saved in ASCII format.

Use the **R**estore command to recover a saved configuration.

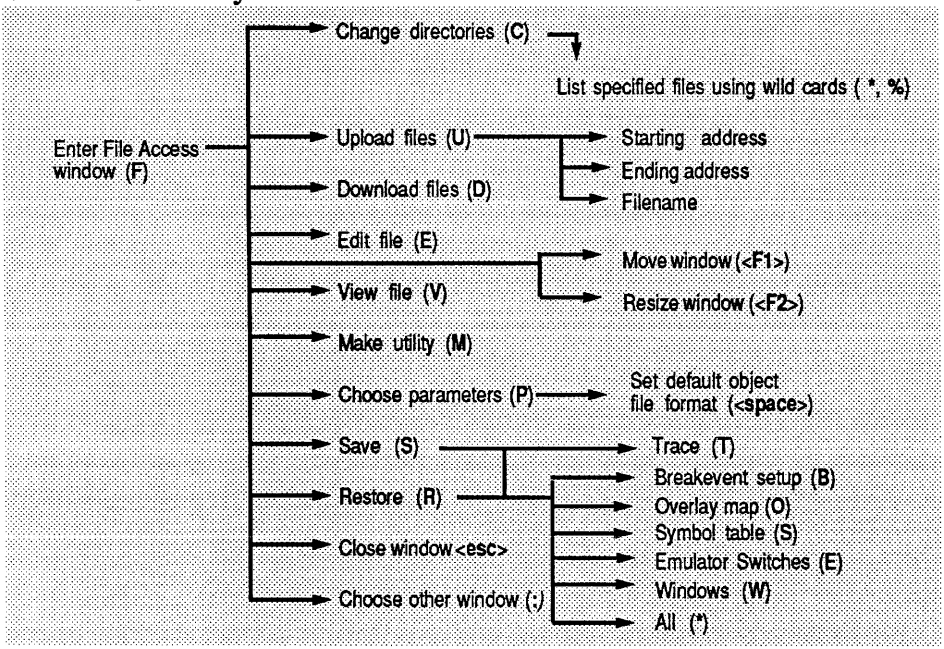
You will be prompted for a file name. If a file of this name already exists, you can overwrite the file or stop the save process.

Restore

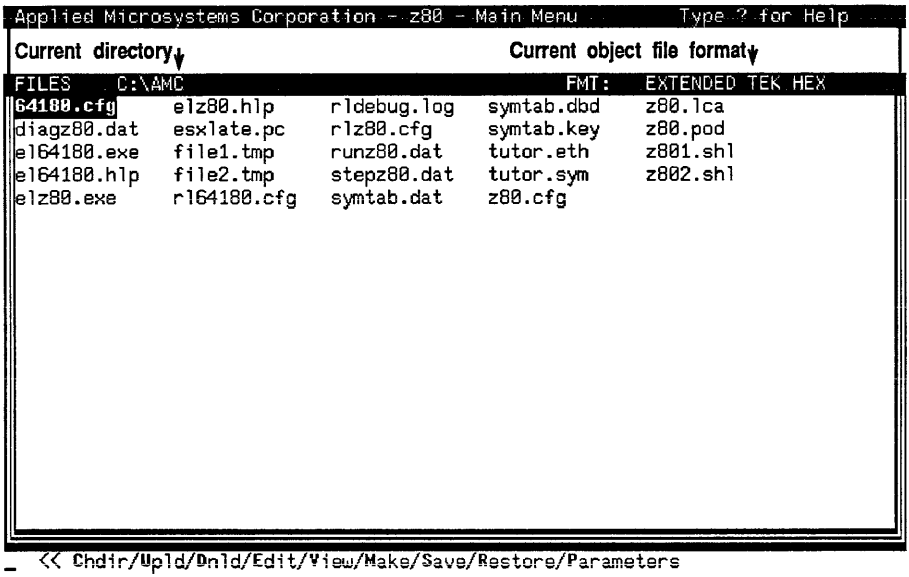
You can restore any of the saved configuration files listed under the Save command except Trace. You must specify the filename of the saved configuration file. The file you name must be of the correct type for the restoration (in other words, you can't restore a saved breakpoint setup to the emulator switches configuration).

When using the Restore All (**:FR***) command, the filename you specify must have been saved using the Save All (**:FS***) command described above.

Command Summary



File Access Window



See Also

Break/Event summary window, page 6-20

Configuration: System, page 6-56

Overlay window, page 6-77

Symbol Table window, page 6-83

MEMORY MODE WINDOW :M

Description

This window displays the contents of memory in one of several formats. Select the format from the Parameters window (default is block). You can display any type of memory space supported by your processor. For the Z80 and 64180, the choices are I/O or memory. To display I/O space, type **io@** immediately before the starting address. To switch back to memory space, type **mem@** at the address prompt.

The Parameters window lets you choose the display mode, display radix, width, and verification status. Changes to the memory display parameters are saved as you make them. When you return to the Memory Mode window, it remembers the display format and parameters from your previous usage unless you initialize the emulator from the Cover window.

You choose the initial memory mode (the memory mode first used when starting the control software) through the Configuration User Interface window (**:CU**). Once set there, the control software will always start in that mode; you change current memory mode with the Parameters window.

The Utilities window lets you move a section of memory or fill a memory section with a constant pattern.

Use the **Refresh** command when you are in block mode and new memory values have been written, especially by I/O input or DMA. For speed's sake, the control software reads the data from memory and stores it in an internal buffer. Therefore, the contents of that buffer are not immediately changed when changes are made to memory. **Refresh** forces a read of all displayed memory locations, and all displayed windows are updated with the most current data. (**Refresh** is not necessary in line mode, since you must specifically read new data with the **Read** command. **Refresh** is also not applicable in ASCII mode, since the **Refresh** command would be interpreted simply as "R.")

Parameters:

The Memory Parameters window lets you choose the mode, display radix, display width and verification status. In all fields, the space bar toggles through the available choices.

Mode: There are three display modes: block, line and ASCII. The block and ASCII modes read a block of memory from the current ad-

dress to display it, and update the display immediately as the cursor is moved or data is entered. In line mode, memory is read and written only when you explicitly direct the emulator to do so (press **R** for read or **W** for write once in line mode). Use the **<return>** key to view the next memory address in line mode.

NOTE: You cannot get to the Parameters or Utilities prompts directly from ASCII mode. Typing a **P** or **U** is interpreted as the ASCII value and simply pokes that letter into memory at the current cursor location. To get out of ASCII mode, type **<ctrl-z>** to enter block mode.

Radix: The radix is the base used to display memory: hex, octal, decimal, or binary. The binary display type is valid only in line mode. If you select binary while in block mode, it will automatically be reset to hex.

Width: This field selects the data width, either byte, word or long. ASCII mode uses only byte wide data.

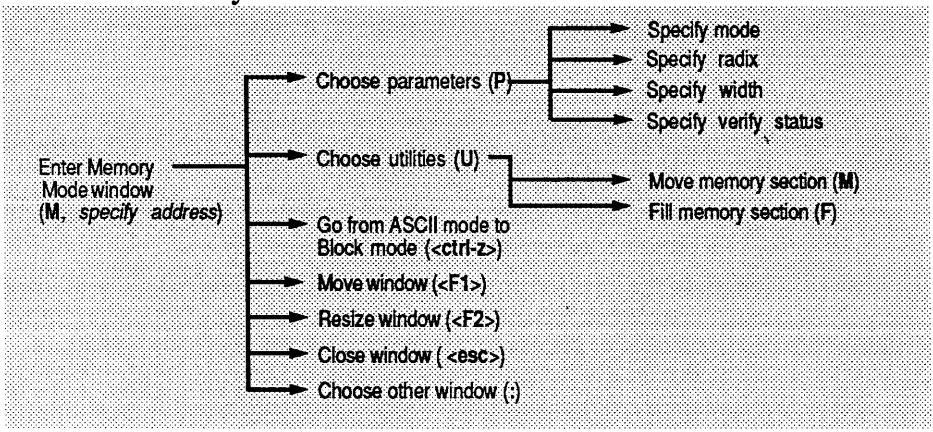
Verify: If verify is on, the emulator performs a read-after-write verify to ensure that the write was successful. You may want to set verify to off when writing large amounts of data to reduce the time required in verification.

Utilities:

Block move This utility copies the data in one memory block to another . You must specify the beginning and ending addresses of the source block and the beginning address of the destination block. The destination block can be within the source block - in this case the source block is read and then shifted to the new starting location. Source and destination locations may be in target or overlay space, depending on how overlay is mapped. You can also specify the memory space for block moves - just preface the appropriate address with the letters **io@** or **mem@**.

Block fill This utility fills a block of memory with a specified hex byte value. Here, you must enter the starting and ending addresses and the desired byte pattern. Source and destination locations may be in target or overlay space, depending on how overlay is mapped.

Command Summary



Memory Mode and Parameters Windows

```
Applied Microsystems Corporation -Z80- Main Menu   Type ? for Help

MEMORY-BLOCK FORMAT  SPACE:MEM      ADDR:1921    VERIFY:ON
1900  3E 00 21 0F 09 06 10 77  2B 05 C2 07 10 3E 07 21  > . ! . . . . w + . . . . .>.!
1910  07 09 06 04 08 D9 3E 00  21 00 09 08 D9 77 2B 3D  . . . . . > . ! . . . . w + =
1920  MEMORY PARAMETERS        00 00 00 00 00 00 00 00  . . . . . .
1930                           00 00 00 00 00 00 00 00  . . . . . .
1940  MODE      BLOCK          00 00 00 00 00 00 00 00  . . . . . .
1950  RADIX     HEX            00 00 00 00 00 00 00 00  . . . . . .
1960  TYPE      BYTE           00 00 00 00 00 00 00 00  . . . . . .
1970  VERIFY    OFF            00 00 00 00 00 00 00 00  . . . . . .
1980                           00 00 00 00 00 00 00 00  . . . . . .
1990                           00 00 00 00 00 00 00 00  . . . . . .
19A0                           00 00 00 00 00 00 00 00  . . . . . .
19B0                           00 00 00 00 00 00 00 00  . . . . . .
```

OVERLAY WINDOW :O

Description

The Overlay window shows the currently mapped overlay segments and types, how much overlay is still available, and the number of wait states for segments specifying wait states. You can add new overlay (**A**), delete existing overlay (**D**), copy data from the overlay memory to your target (**UT**), or copy data from your target memory to overlay (**UO**).

The use of overlay memory is highly recommended in targets with electrically uncertain memory, and required in targets without functioning memory of their own. The overlay memory, an optional part of the emulator hardware, logically replaces the target memory. Using this window, you *map* portions (or all) of the target memory to overlay. In other words, you redirect target memory accesses to the emulator's overlay memory. This is done to simplify debugging, since the overlay memory is known to be good, whereas in many cases the target memory is uncertain.

Mapping overlay refers to the process of assigning overlay memory locations certain attributes such as whether a section of overlay should emulate RAM (read and write), ROM (read only), or illegal target memory. The contents of overlay memory are automatically retained between sessions if you have a battery backed up overlay module.

Your overlay map can be permanently saved using the File Access Save Overlay command (**:FSO**), and restored using the Restore Overlay command (**:FRO**). To save multiple maps, use a different file name for each map.

The smallest memory segment you can map is one kilobyte (1024 bytes). Note that if you map a section of memory that is already mapped, your new map overwrites your previous map. You can also undo a specific overlay mapping by remapping that section of overlay back to the target.

The following mapping types are supported:

Target	Map memory accesses to your target memory.
Read/Write	Map overlay as read and write memory.
Read only	Map overlay as read only memory. With this mapping, read accesses go to overlay and write accesses from the target will halt the emulator and display an error

message. This is useful for protecting PROM space. You can also modify read only memory from the Memory window if desired.

Illegal — Map an address range as illegal, i.e. reads/writes are not permitted. If the target attempts to access an illegal address, emulation is halted and an error message is displayed.

Use the Add command to map target memory segments to overlay. If you did not specify a mapping type, the segment will be mapped as read and write memory. Starting and ending addresses are modified as necessary to conform to the 1 KB minimum mapping resolution. For instance, if you try to map target addresses 0 to 100 to overlay, the actual address range mapped will be 0 to 3FF.

With read only and read/write memory mapping, you must also specify if you want wait states enabled for the segment you are mapping. The actual number of wait states is set in the Configuration Emulator window (**:CE**); the entry allows a number of wait states greater than zero to occur for this memory segment. Pressing **<return>** at this prompt results in wait states *disabled* for this memory segment. The number of wait states is shown in the top line of the Overlay window.

The **D**elete command re-maps the specified address range to target memory. You can delete any overlay mapping entry individually by entering the item number, or all of them using the asterisk (*). There is no undo function to restore a deleted overlay mapping entry.

The total amount of overlay you have depends on the number of overlay modules you have with your EL 800 emulator. The Overlay Available information in the top line of the window should match what you have installed (unless you have already performed mapping). If it does not, see Section 2 for more information on installing the Overlay Modules.

There are two types of Overlay Modules: those with battery backup, and those without. If you have battery backed up modules, code loaded into overlay memory is automatically saved between sessions. To check to see if you have battery backed up overlay, type **<ctrl-c>** to look at the Cover window. After the size of the module, you'll see a -B if the module has battery backup.

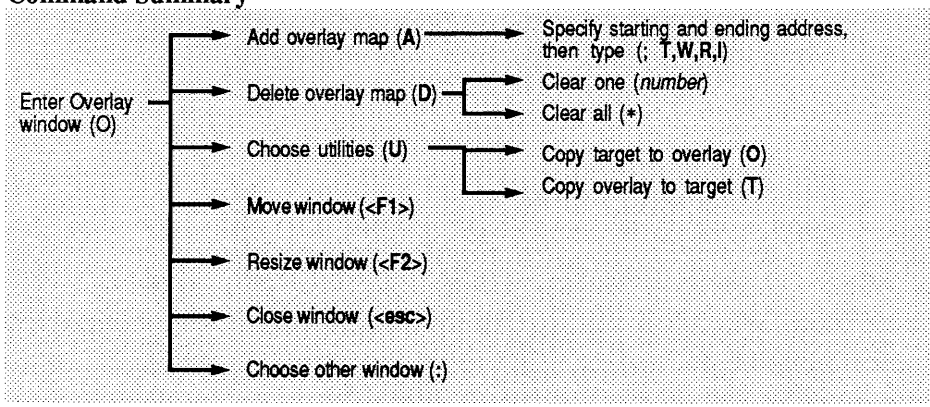
If you plan to use data restored from battery-backed up overlay with newly loaded overlay mapping, perform all desired overlay mapping, save the mapping (**:FSO**), restore the mapping (**:FRO**), and then restore the data to overlay, in that sequence. Following any other sequence can result in incorrect data in memory.

Utilities:

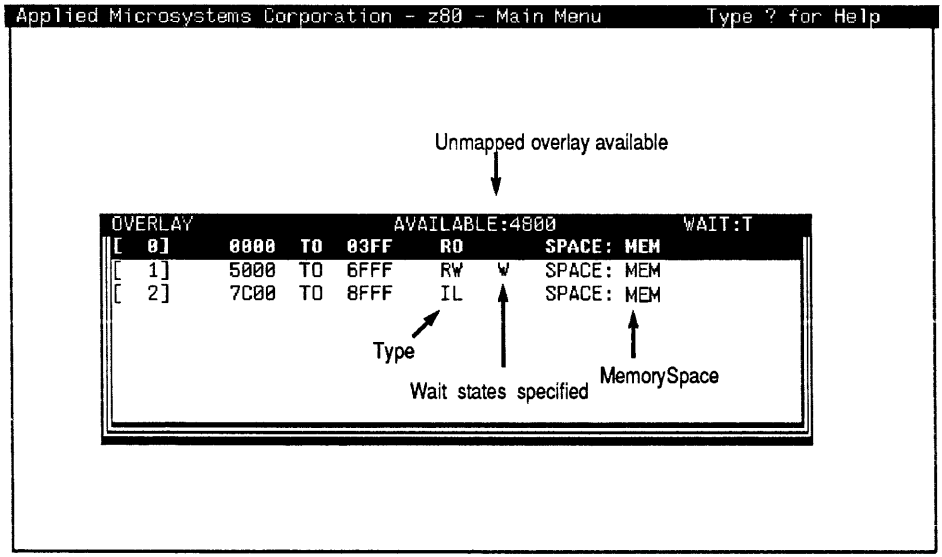
Ovl<-- tgt Copy data from target memory to overlay memory. Specify the starting address for the block of target memory you want to move, the ending address, and the destination address in the overlay module. Without a destination address, the block is copied to the same address range specified with the starting and ending addresses.

Tgt<--ovl Copy overlay memory to target memory. Specify the starting address for the block of overlay memory you want to move, the ending address, and the destination address in the target. Without a destination address, the block is copied to the same address range specified with the starting and ending addresses.

Command Summary



Overlay Window



See Also

File Access window, page 6-69

Configuration Emulator window, 6-50

Section 2

REGISTERS WINDOW :R

Description

The Registers window is used to view and change the CPU registers. The registers are divided into a number of groups depending on the processor (see Section 5 for specific information). The Z80 has two groups (primary and alternate) and the 64180 has six groups (primary and groups 1 - 5). You toggle between groups by placing the cursor on the group name (for instance, "Primary") and pressing the space bar.

Within each group, registers are displayed in pairs following the fashion of the microprocessor reference guide. This means that BC, DE, and HL registers are displayed as 16 bit entities. The flags register (F) is divided into an individual bit display at the bottom of the window.

Individual changes to the CPU registers are written to memory when the cursor is moved to another register. All your changes are saved when you leave the Registers window. When you restart the software, the register values are recalled unless you use the Initialize command from the Cover window.

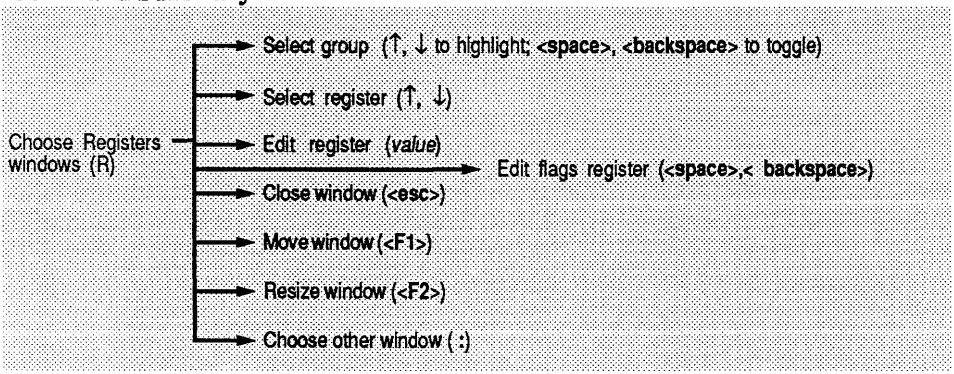
Due to the way the Z80 and 64180 operate, the IFF1 register is write only. It will always display 0 when read, but that doesn't show the true register value. Also, the TMDR1 register (64180 only) tracks the 64180 timers. These timers do *not* stop when emulation is halted. You will, therefore, notice changes to this register value even when the emulator is stopped.

You can also see the value of a register by using the expression analyzer, or use the value of a register in an expression. For instance,

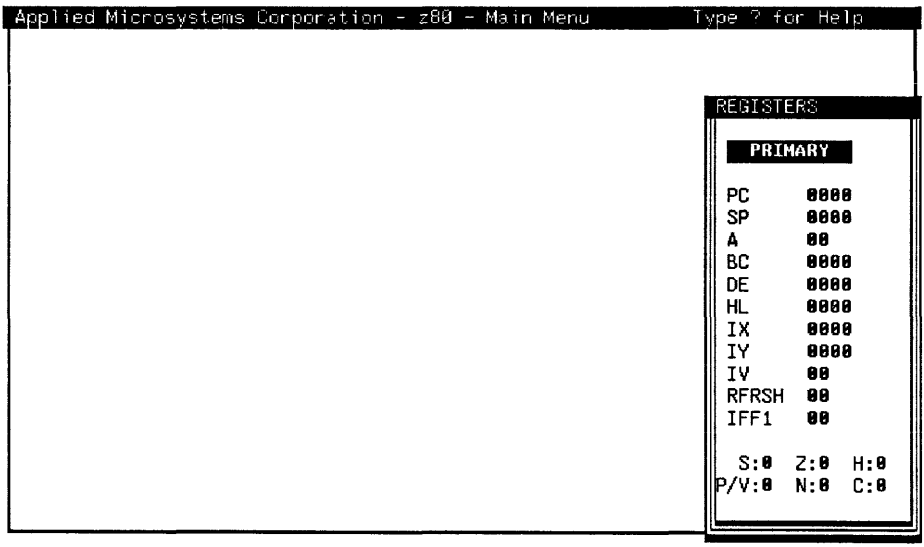
```
*((WORD *) SP +4)
```

entered in the expression analyzer results in the word 4 bytes from the stack pointer in the CPU's stack.

Command Summary



Registers Window



See Also

Emulate window, page 6-63

Expression Analyzer, page 6-15

Section 5

SYMBOL TABLE WINDOW :S

Description

The Symbol Table window is used to add symbols to the symbol table, find symbols, locate all symbols in a particular module and to delete symbols from the symbol table.

Symbols provide a way to reference addresses, data, and numbers used in your program with natural language names, rather than numbers. Most compilers and assemblers also generate symbol information that you can download into the emulator. The use of symbols in your debugging session can reduce the confusion inherent in referring to long numbers and addresses, and can reduce typing by substituting shorter symbol names for large numbers.

You can scroll the symbol table with the **<pg up>**, **<pg dn>** and ↑ ↓ keys.

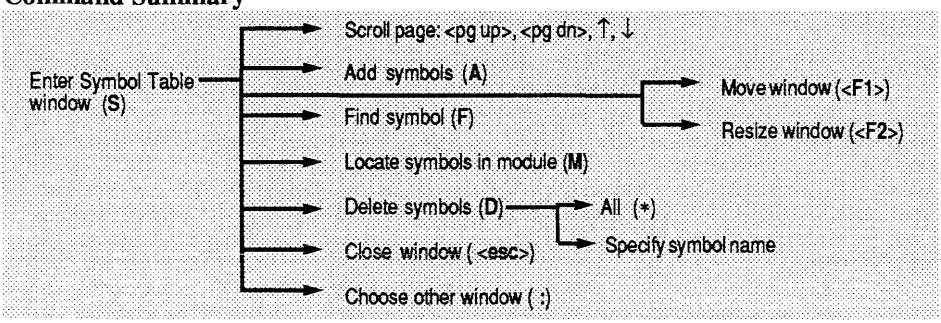
When you **Add** a symbol, you must specify the symbol name and its value. You can also use this command to change the value of an existing symbol.

When you **Delete** a symbol, you can delete one specific symbol by specifying its name, or delete all symbols by entering an asterisk (*).

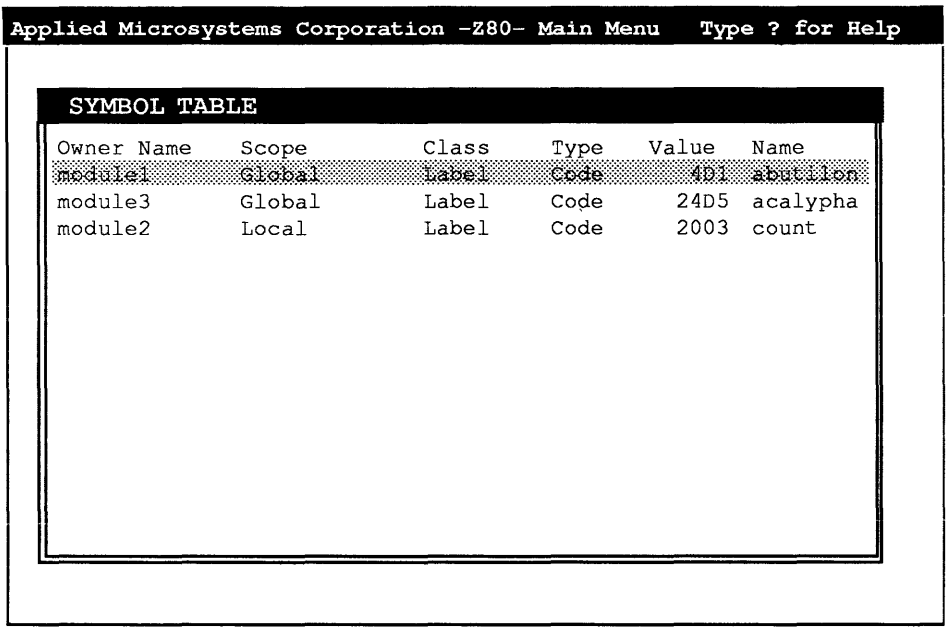
The **Find** command will search the symbol table for the symbol name you specify and display its parameters. The **Module** command searches the symbol table for all symbols that belong to a specified module.

Changes to the symbol table are retained by the emulator until you leave the control software. The symbol table can be saved using the File Access Save Symbols command (**:FSS**). It may be restored using the File Access Restore Symbols command (**:FRS**).

Command Summary



Symbol Table Window



The columns in the table are:

Owner Name	The software module where the symbol is defined.
Scope	The area of execution where the symbol has meaning. When the PC is within the defined scope, the symbol is valid. If global, the symbol is always valid.
Class	The following class types define where or how a symbol is stored: code, data, local (meaning the symbol is probably stack based), section, module, or module end. If none of these classes can be determined for a symbol, it is classed as **UNKNWN** (for unknown).
Type	The following data types define the kind of data defined by the symbol: code, data, local (meaning the symbol is probably stack based), section, label, or module end. If none of these types can be determined for a symbol, it is typed as **UNKNWN** (for unknown).

	The software attempts to further classify "data" types as either label, byte, word, 3 byte, or long. If the data type is indeterminate, you will see "data" in the type field.
Value	The value the symbol is set to.
Name	The symbol name must start with an alphabetic character. It can include numbers, underscores (_), and dollar signs ($).

See Also

File Access window, page 6-69

TRACE WINDOW :T

Description

The Trace window contains a history of your program's execution. If you have not qualified the trace information using the Advanced Event System, a complete history of the last 8192 bytes of information is available. If trace has been qualified, you will see only those cycles you requested. You can display the trace buffer in either of the following formats:

Raw	Display the bits traced for every traced bus cycle
Disassembled	Display trace as disassembled code, showing data movement

Use the **Parameters** command to specify the format desired.

Trace is captured dynamically as your program is running. If you request trace display while in run mode, the Advanced Event system is automatically disabled (for as brief a period as possible), and the trace buffer is displayed. The Advanced Event system is then automatically re-enabled. Note that:

1. You can't acquire trace and read trace at the same time.
2. While reading trace while running, the Advanced Event System is disconnected, so you will miss any events occurring during this time.

To begin the display at a specified line in the trace buffer, use the **Display** command.

You can save trace data to a file with the Save Trace command (**:FST**). There is no practical need to restore trace, so you will not find a matching Restore Trace command. Once trace data is saved, you can view it with the File View command (**:FV**).

Trace Window (Raw Trace)

```
Applied Microsystems Corporation -Z80- Main Menu   Type ? for Help

 TRACE DISPLAY
LINE    ADDR      DATA   R/W Bus    RQ AK  WA  INT  STATE
61              RESET MARK
60      '0066     32     R   FETCH                    1
59       0067     E7     R   DATA           W        1
58       0068     1F     R   DATA       A            N    1
57       1FE7     53     W   DATA                        1
56      '0069     3E     R   FETCH R                     1
55       006A     7E     R   DATA                        1
54      '006B     D3     R   FETCH                  1    1
53       006C     03     R   DATA                        1
52       0303     7E     W   IO                          1
51      '006D     3E     R   FETCH                       1
50       006E     54     R   DATA                        1
49      '006F     D3     R   FETCH                       1
48      '0070     02     R   FETCH                       1
47       0202     54     W   IO                          1
46      '0071     3A     R   FETCH                       1

_ << Display/Parameters
```

The raw trace display columns are:

LINE #: The line number in the display. 0 corresponds to the most recently traced cycle.

ADDR: Z80: One column shows the address of the data fetched. If EXTADDR is set to 1, the extra four bits of address available are also shown. A single quote (') indicates that the address is the first byte of an instruction fetch.
64180: One column shows the address of the data fetched. A single quote indicates that the address is the first byte of an instruction fetch.

DATA: The data on the bus during the cycle.

R/W: Shows whether data is read from or written to memory.

BUS: The type of bus cycle:

INTAK	Interrupt acknowledge
DATA	Data transfers between memory and the CPU
FETCH	Opcode fetch. A FETCH with no single quote next to the address indicates a subsequent fetch of a multiple byte op code in a multiple M1 (LIR) cycle.
IO	I/O cycle. Data was read from or written to an I/O device.

RQ: An R in this column indicates a bus request.

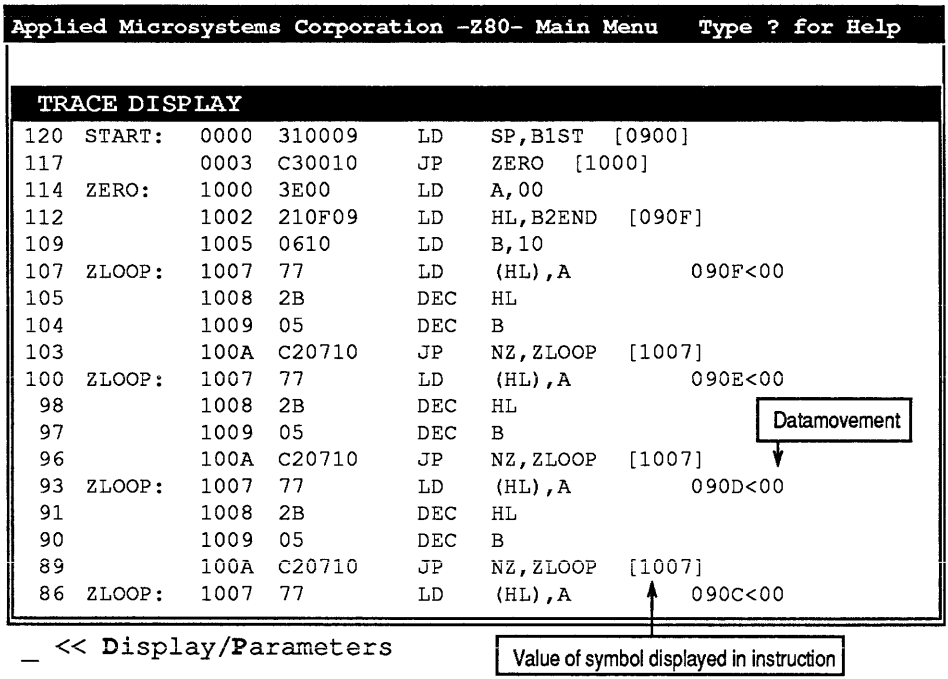
AK: An A in this column indicates the bus acknowledge line was active during this cycle.

WA: A W in this column indicates one or more wait states were inserted during that bus cycle.

INT: Z80: Interrupt type. N is NMI. 1 is interrupt pending, blank means no interrupt is pending.
64180: Interrupt type. N is NMI. 1-7 are interrupts pending, blank means no interrupt is pending.

STATE: This field shows the currently executing Advanced Event System state (1-4).

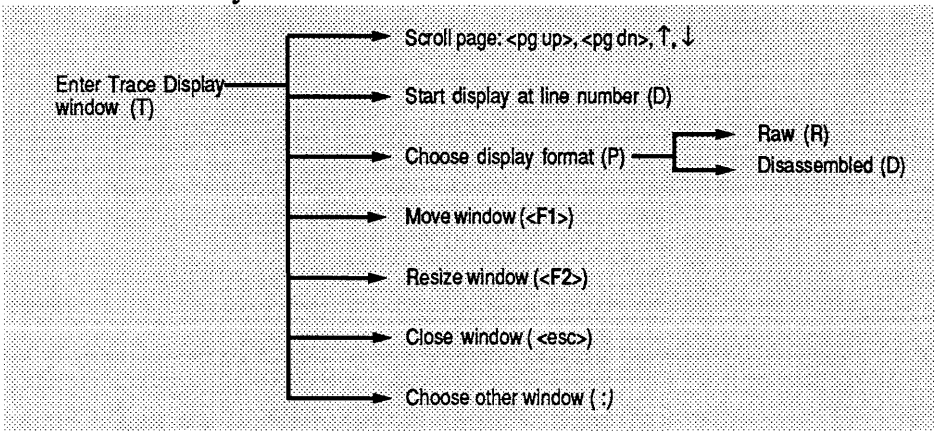
Trace Window (Disassembled Trace)



The disassembled trace columns are:

Cycle # Symbols Address Object Code Instruction Data movements

Command Summary



See Also

Emulate window, page 6-63

Appendix G

WATCH WINDOW :W

Description

The Watch window is designed to let you keep an eye on important expressions, symbols, constants, registers, and memory locations. Any item in the Watch window is updated immediately if its value changes.

You can use the Watch window to display data in a format appropriate to the task at hand. Use the Add command to add items. For example, you can enter an expression to display values in a structured fashion, such as arrays and strings, to avoid searching through a memory dump for these values. You can also display many unrelated objects in the Watch window, or values from non-contiguous section of memory or registers. Typically, you would set up the Watch window and then return to the Emulate window to run your program.

Symbols entered in the Watch window must first be added to the symbol table to be recognized. The use of the Watch window in conjunction with the Emulate Go strobe option (:EG&) slightly interferes with real-time emulation, as the emulator must periodically (once per second) stop to update the Watch window information.

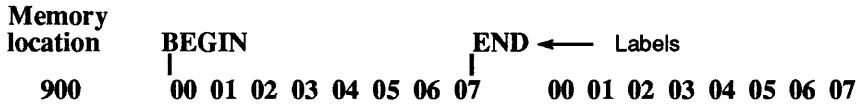
With the REALTIME switch in the Configuration Emulator window set to ON, you can open the Watch window, but you cannot add any elements to it while emulating. Also, the Watch window will not be updated during emulation with the REALTIME switch set to ON.

You can edit items already in the Watch window by typing **E**. The prompt asks you which item you want to edit. Identify the appropriate item with its number. The proper item is displayed in the expression analyzer window for you to edit as you wish. After editing the item, press **<return>** to complete the edit.

Use the **D** command to delete items from the Watch window, either individually by number or all (*) at once.

A few examples of Watch window expression syntax follow. For a complete guide, see Appendix F, *Using Expressions*. Note that addresses used in these examples are four-nibble Z80 addresses, and 64180 addresses will appear as five-nibbles.

All examples below refer to the following memory block contents (0900h to 090Fh inclusive):



The labels BEGIN and END we assume are set equal to the corresponding memory locations in the symbol table.

Expression	Evaluation
BEGIN	0900

The value of the label BEGIN is 0900, as set in the symbol table.

Expression	Evaluation
(long) BEGIN	00000900

The value of the label BEGIN has been cast as a long word and padded with leading zeros.

Expression	Evaluation
* BEGIN	03020100

The * operator dereferences the pointer BEGIN and the contents of 0900, 0901, 0902, and 0903 are 03020100.

Since the expression evaluator does not know the type of * BEGIN, "long" is used as the default.

Expression	Evaluation
(byte *) BEGIN	0900

The value of BEGIN is still the word 0900, but it has been cast as a pointer to a byte, instead of as a pointer.

to a default long.

Expression	Evaluation
* (byte *) BEGIN	00

The first * operator dereferences the pointer BEGIN, which is cast as a pointer to a byte. The byte in address 0900 has a value of 00.

Expression	Evaluation
* (byte *) END + 1	08

The first * operator dereferences the pointer END, which is cast as a pointer to a byte. The byte in address 0907 is 07. 1 + 07 = 08.

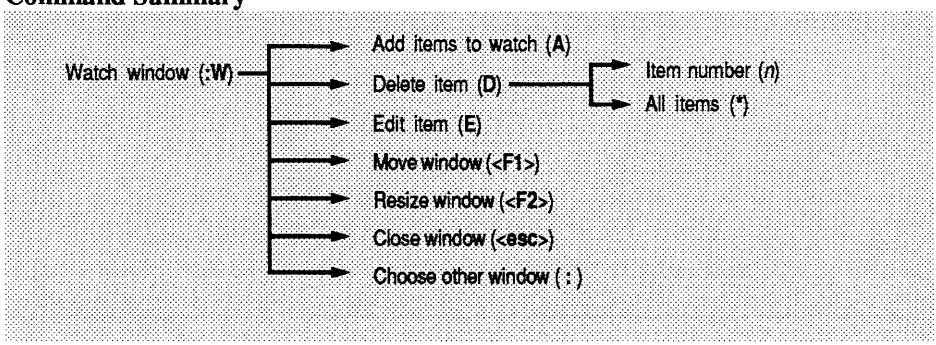
Expression	Evaluation
* (byte *)(END + 1)	00

The first * operator dereferences the pointer (END + 1), which is cast as a pointer to a byte. The byte in address 0908 is 00. Format specifiers may be appended to an expression with a comma. These specifiers can control both number and format of the expression evaluations. It should be noted that the use of a repeat number in the format specifier will dereference a pointer expression. An example of this follows:

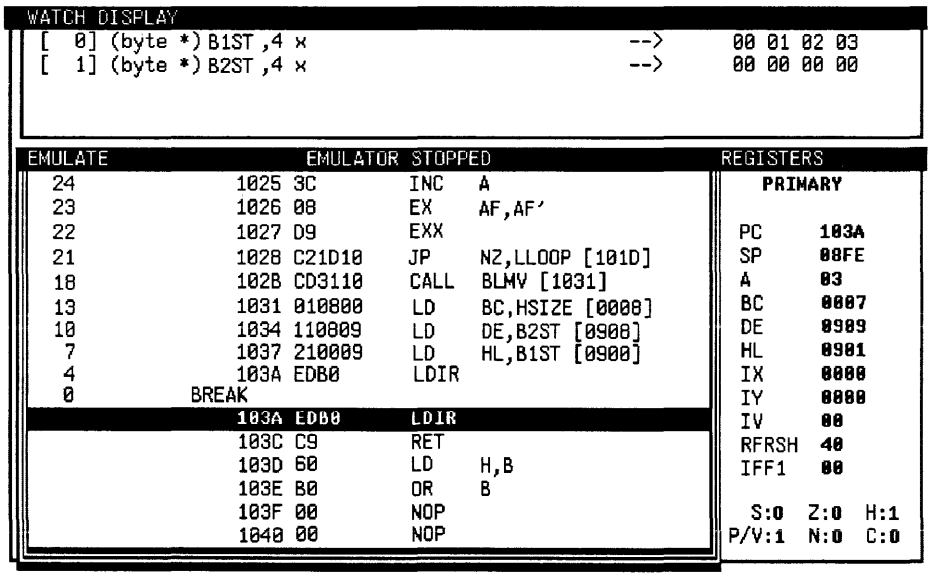
Expression	Evaluation
(byte *) END ,x	0907
(byte *) END ,1x	07
(byte *) END ,2x	07 00

In the first case, END is cast as a pointer to a byte, and evaluated as 0907(hex). In the second case, END is also cast as a pointer to a byte, but is dereferenced by the repeat number "1" in the format specifier "1x", and so is evaluated as the contents of 0907, the byte 07. The third case is essentially the same as the second, but with a repeat number of "2x". In this case, the contents of 0907 and (0907 + 1) are evaluated.

Command Summary



Watch Window



See Also

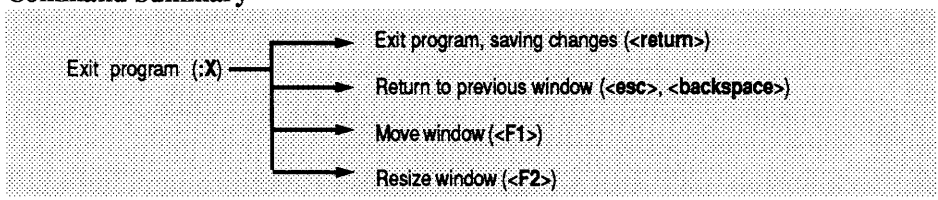
Expression Analyzer, page 6-15

EXIT WINDOW :X

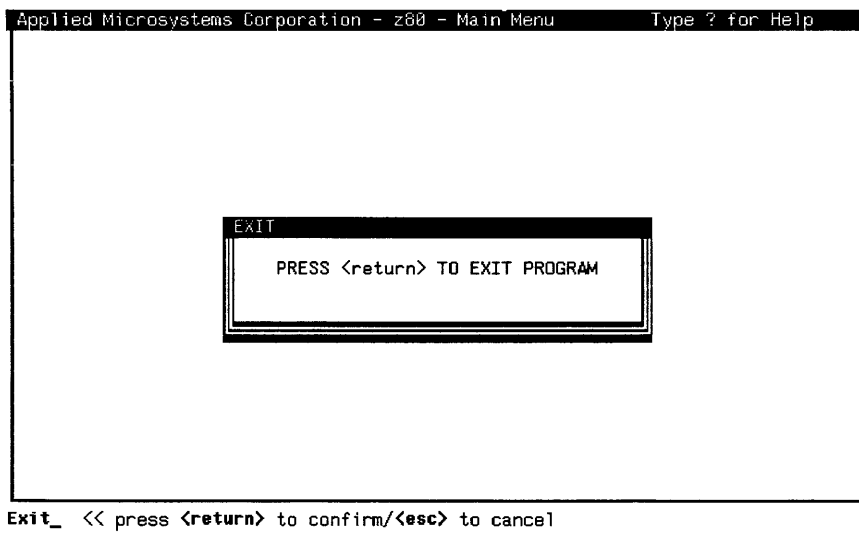
Description

To exit the control software press **:X <return>**. To cancel, press **<esc>** or **<backspace>** . Another method of leaving the control software is to press **<ctrl-c> <ctrl-c>**. To save any portion or all of your emulation configuration, see the Save command in the *File Access Window* section.

Command Summary



Exit Window



APPENDIX A

Table of Contents

Command Summary

The following abbreviations are used within this appendix:

addr	Address in hex or expression (type <space> to enter expression analyzer). When entering addresses that can refer to either I/O or memory space, prefix the address with either **io@** or **mem@** as appropriate. For example, entering **io@10** makes it clear you are referring to I/O address 10, and **mem@3fff** indicates memory address 3fff.
expr	Expression.
file	Filename with optional extension.
n	Decimal number.
dir	Directory path.
value	Hex value or expression (type <space> to enter expression analyzer).
symbol	Symbol name.
[*item*]	Square brackets indicate an optional item. Don't type the brackets.

Commands are shown in upper case, but are case independent.

All commands are shown as you would enter them from anywhere within the program. You may shorten the command if the desired window is already displayed.

?	View help information.	6-1
!	Shell escape.	6-5
<esc>	Exit the current window, saving changes.	6-3
<ctrl-c>	Display Cover window.	6-6
<ctrl-c>I	Initialize emulator.	6-9
<ctrl-c>R	Reload emulator shell code.	6-11
<space>	Enter expression analyzer.	6-15
<F1>	Move current window.	6-4
<F2>	Resize current window.	6-4
<ctrl-c><ctrl-c>	Exit program.	6-93

:A	Prompt for beginning assembler address.	6-17
:A*addr*	Display Assembler window showing disassembly from address given.	6-17
:A*addr line*	Enter new line to be assembled.	6-17
:B	Display Break / Event Summary window.	6-20
:BB	Display Basic Breakpoint window.	6-26
:BBD*n*	Delete breakpoint *n*.	6-26
:BBD*	Delete all breakpoints.	6-26
:BBE*n*	Enable breakpoint *n*.	6-26
:BBE*	Enable all breakpoints.	6-26
:BBI*n*	Disable breakpoint *n*.	6-26
:BBI*	Disable all breakpoints.	6-26
:BBS*addr*	Set breakpoint at address.	6-26
:BBS*addr, addr*	Set breakpoint in range.	6-26
:BC	Clear Advanced Event and Basic Breakpoint Systems.	6-24
:BW	Display WHEN-THEN statements.	6-20
:BX	Change counter comparator X.	6-29
:BY	Change counter comparator Y.	6-29
:B*n*	Enter Advanced Event state *n* window (1-4).	6-31
:B*n***AS***addr*	Set address comparator A in state *n*.	6-35
:B*n***AS***addr*[; *mask*]	Set address comparator A address with optional don't care mask.	6-35
:B*n***AS***addr*,*addr*	Set address comparator A address range.	6-35
:B*n***AD***m*	Delete value *m* in address comparator A, state *n*.	6-36
:B*n***AD***	Delete all values in address comparator A, state *n*.	6-36
:B*n***BS***addr*	Set address comparator B.	6-35
:B*n***BS***addr*[; *mask*]	Set address comparator B address with optional don't care mask.	6-35
:B*n***BS***addr*,*addr*	Set address comparator B address range.	6-35
:B*n***BD***m*	Delete value *m* in address comparator B, state *n*.	6-36
:B*n***BD***	Delete all values in address comparator B, state *n*.	6-36
:B*n***SE***data*	Set data comparator E.	6-37

:B*n*SE*data,data*	Set data comparator E range.	6-37
:B*n*ED*n*	Delete value *n* in data comparator E.	6-37
:B*n*ED*	Delete all values in data comparator E.	6-37
:B*n*ES*value[;mask]*	Set optional don't care mask.	6-37
:B*n*SF*data*	Set data comparator F.	6-37
:B*n*SF*data,data*	Set data comparator F range.	6-37
:B*n*FD*n*	Delete value *n* in data comparator F.	6-37
:B*n*FD*	Delete all values in data comparator F.	6-37
:B*n*FS*value[;mask]*	Set optional don't care mask.	6-37
:B*n*RD	Delete status comparator R values.	6-39
:B*n*RS	Set status comparator R values.	6-39
:B*n*SD	Delete status comparator S values.	6-39
:B*n*SS	Set status comparator S values.	6-39
:B*n*WD*m*	Delete WHEN-THEN statement *m* in state *n*.	6-43
:B*n*WD*	Delete all WHEN-THEN statements.	6-43
:B*n*WW*events*T*actions*	Enter WHEN-THEN statements.	6-43
:C	Display Configuration window.	6-46
:CC	Display communications configuration window.	6-47
:CE	Display emulator soft-switch configuration window and change number of wait states.	6-50
:CU	Display the User Interface Configuration window.	6-58
:CS	Display Configuration System window.	6-56
:D	Display Diagnostics window.	6-60
:E	Display Emulate window.	6-63
:EE	Display Event-state window.	6-67
:EEC	Clear event-state variables.	6-67
:EG	Run program with breakpoints enabled.	6-63
:EG*addr*	Run program beginning at new PC address.	6-63
:EG*	Run program with no breakpoints.	6-63
:EG&	Run program with strobe.	6-63
:ES	Single step once.	6-63

:ES*n*	Step by increments of *n*.	6-63
:ES*	Step continuously.	6-63
:EZ	Restart program.	6-64
:F	Display the File Access window.	6-69
:FC*dir*	Change to directory specified.	6-69
:FD*file*	Download the file name specified.	6-70
:FRB*file*	Restore breakevent settings from file.	6-71
:FRE*file*	Restore emulator softswitch settings from file.	6-71
:FRO*file*	Restore overlay map from file.	6-71
:FRS*file*	Restore symbol table from file.	6-71
:FRW*file*	Restore window settings from file.	6-71
:FR* *file*	Restore all except trace and symbols.	6-71
:FSB*file*	Save breakevent settings to file.	6-71
:FSE*file*	Save emulator softswitch settings to file.	6-71
:FSO*file*	Save overlay map to file.	6-71
:FSS*file*	Save symbol table to file.	6-71
:FSW*file*	Save window settings to file.	6-71
:FS* *file*	Save all except trace and symbols.	6-71
:FST*file*<**return**>*startcycle,#lines;format*	Save trace information to file.	6-71
:FU*addr1*, *addr2*; *file*	Upload memory range specified to file.	6-69
:FV*file*	View file.	6-70
:FE*file*	Edit file.	6-70
:FM*file*	Invoke make utility.	6-70
:FP	Open the Parameters sub-window to change object file format.	6-70
:M	Display Memory window.	6-74
:M*addr*	Display Memory starting at address given.	6-74
:MP	Display Memory parameters window.	6-74
:MR	Refresh Memory window.	6-74
:MU	Display Memory utilities window.	6-74

APPENDIX B

Table of Contents

Serial Interface

SERIAL INTERFACE

Serial Interface for the PC AT

Figure B-1 shows the serial interface for the PC AT.

Figure B-2 shows the cable wiring as if there were only one cable, rather than the two supplied.

Figure B-3 shows the wiring diagrams for the two supplied cables.

Figure B-1. Serial Interface for the PC AT

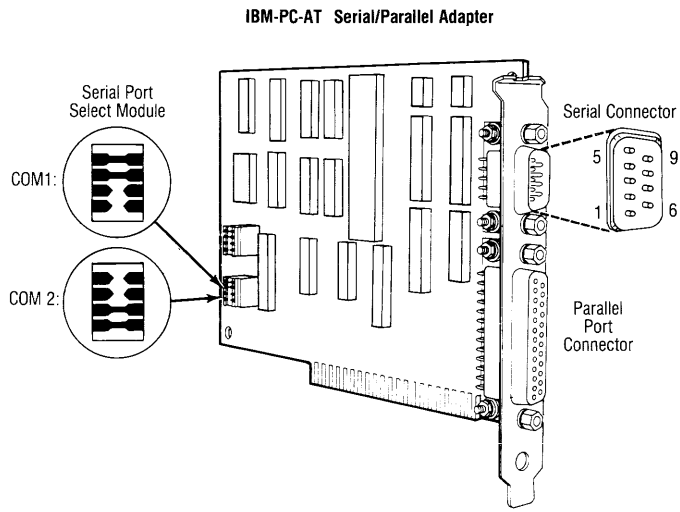


Figure B-2. Wiring Diagrams for Combined Cables: EL 800 to PC AT

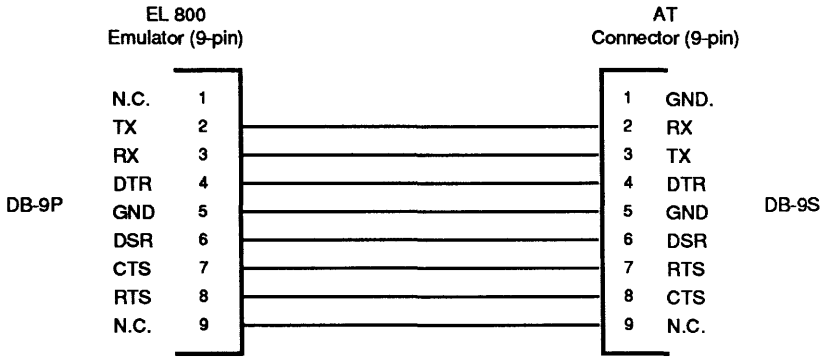
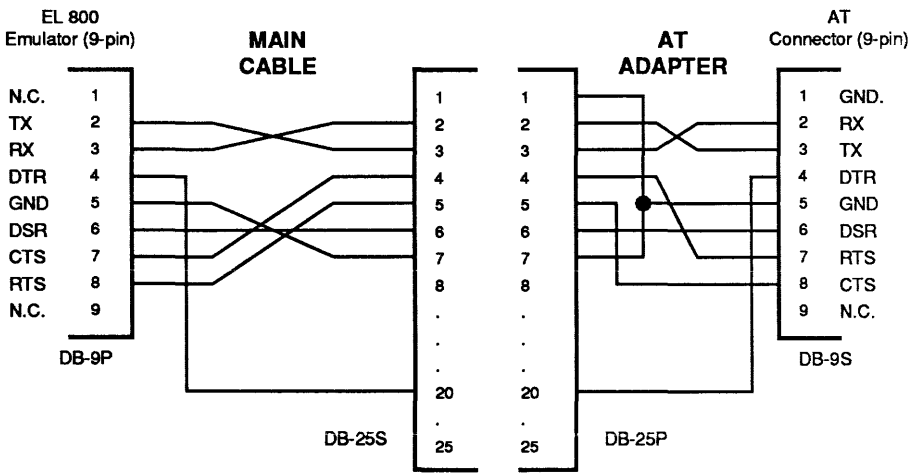


Figure B-3. Wiring Diagrams for Both Cables: EL 800 to PC AT



Serial Interface for the PC XT

Figure B-4 shows the serial interface for the PC XT. Figure B-5 shows the wiring diagram for the 9-to-25 pin cable used to connect the EL 800 to the PC XT.

Figure B-4. Serial Interface for the PC XT

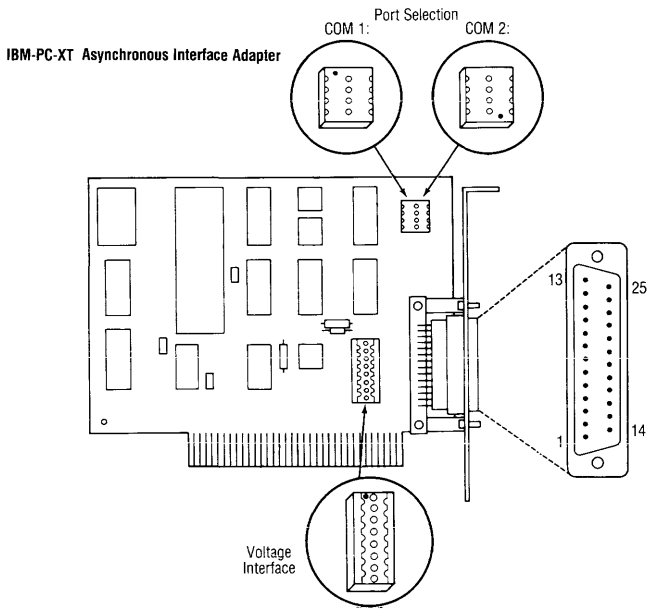
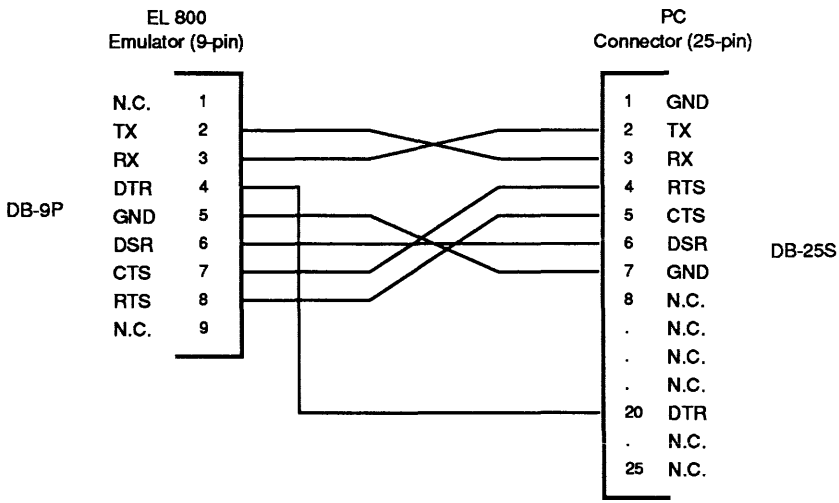


Figure B-5. Wiring Diagrams for EL 800 to PC XT Cable



APPENDIX C

Table of Contents

File Formats for Object Files

FILE FORMATS FOR TARGET OBJECT FILES

Program File Up/Download Formats

The EL 800 will upload and download files using Motorola S Record, Tektronix Hex, Hitachi S Records, Extended Tekhex, and Intel Hex formats. Attempts to upload or download files in a format other than these formats will produce unpredictable results.

Extended Tekhex Format

Extended Tekhex uses three types of message blocks:

1. The data block contains the object code.
2. The symbol block that contains information about a program section and the symbols associated with it. This information is only needed for symbolic debug.
3. The termination block contains the transfer address and marks the end of the load module.

NOTE

Extended Tekhex has no specially defined abort block. To abort a formatted transfer, use a Standard Tekhex abort block.

Each block begins with a six-character header field and ends with an end-of-line character sequence. A block can be up to 255 characters long, not counting the end-

of-line character. The header field has the format shown in the following table.

Extended Tekhex Format		
ITEM	NUMBER OF ASCII CHARACTERS	DESCRIPTION
%	1	A percent sign specifies that the block is in Extended Tekhex format.
Block Length	2	The number of characters in the block: a two-digit hex number. This count does not include the leading % or the end-of-line.
Block Type	1	6 = data block 3 = symbol block 8 = termination block
Checksum	2	A two-digit hex number representing the sum, mod 256, of the values of all the characters in the block, except the leading %, the checksum digits and the end-of-line. The following table gives the values for all characters that may appear in Extended Tekhex message blocks.

Character Values for Checksum Computation	
Characters	*Values (Decimal)*
0..9	0..9
A..Z	10..35
$	36
%	37
. (period)	38
_ (underscore)	39
a..z	40-65

Variable-Length Fields

In Extended Tekhex, certain fields may vary in length from 2 to 17 characters. This practice enables you to compress your data by eliminating leading zeros from numbers and trailing spaces from symbols. The first character of a variable-length field is a hexadecimal digit that indicates the length of the rest of the field. The digit 0 indicates a length of 16 characters.

For example, the symbols **START, LOOP,** and **KLUDGESTARTSHERE** are represented as **5START, 4LOOP,** and **0KLUDGESTARTSHERE.** The values **0, 100H,** and **FF0000H** are represented as **10, 3100,** and **6FF0000.**

Data and Termination Blocks

If you do not intend to transfer program symbols with your object code, you do not need symbol blocks. Your load module can consist of one or more data blocks followed by a termination block. The following tables show the format for a data block and a termination block.

Extended Tekhex Data Block Format

ITEM	NUMBER OF ASCII CHARACTERS	DESCRIPTION
Header	6	Standard header field Block type = 6
Load Address	2 to 17	Address where the object code is to be loaded: a variable length number.
Object	2*n*	*n* bytes, each represented as two hex digits

Extended Tekhex Termination Block Format

ITEM	NUMBER OF ASCII CHARACTERS	DESCRIPTION
Header	6	Standard header field Block type=8
Transfer Address	2 to 17	Address where program execution is to begin: a variable-length number

Symbol Blocks

A symbol used in symbolic debug has the following attributes:

1. The symbol itself: 1 to 16 letters, digits, dollar signs, periods, a percent sign, or symbolize a section name. Lower case letters are converted to upper case when they are placed in the symbol table.
2. A value: up to 64 bits (16 hexadecimal digits).
3. A type: address or scalar. (A scalar is any number that is not on address.) An address may be further classified as a code address (the address of an instruction) or a data address (the address of a data item). As symbolic debug does not currently use the code/data distinction, the address/scalar distinction is sufficient for standard applications of Extended Tekhex.
4. A global/local designation. This designation is of limited use in a load module, and is provided for future development. If the global/local distinction is not important for your purposes, simply call all your symbols global.
5. Section membership. A section may be thought of as a named area of memory. Each address in your program belongs to exactly one section. A scalar belongs to no section.

The symbols in your program are conveyed in symbol blocks. Each symbol block contains the name of a section and a list of the symbols that belong to that section. (You may include scalars with any section you like.) More than one block may contain symbols for the same section. For each section, exactly one symbol block should contain a section definition field, which defines the starting address and length of the section.

If your object code has been generated by an assembler or compiler that does not deal with sections, simply define one section called, for example, MEMORY, with a starting address of 0 and a length greater than the highest address used by your program; and put all your symbols in that section.

The following table gives the format of a symbol block. Tables that follow give the formats for section definition field and symbol definition fields, which are parts of a symbol block.

Extended Tekhex Symbol Block Format		
ITEM	NUMBER OF ASCII CHARACTERS	DESCRIPTION
Header	6	Standard header field Block type=3
Section Name	2 to 17	The name of the section that contains the symbols defined in this block: a variable-length symbol.
Section Definition	5 to 35	This field must be present in exactly one symbol block for each section. This field may be preceded or followed by any number of symbol definition fields. The table on the next page gives the format for this field.
Symbol	5 to 35	Zero or more symbol definition fields as described in the next table.

Extended Tekhex Symbol Block Format: Section Definition Field		
ITEM	NUMBER OF ASCII CHARACTERS	DESCRIPTION
0	1	A zero signals a section definition field.
Base	2 to 17	The starting address of the Address section: a variable-length number.
Length	2 to 17	The length of the section: a variable-length number, computed as: 1 + (high address -base address)

Extended Tekhex Symbol Block Format: Symbol Definition Field		
ITEM	NUMBER OF ASCII CHARACTERS	DESCRIPTION
Type	1	A hex digit that indicates the global/local designation of the symbol, and the type of value the symbol represents: 1 = global address 2 = global scalar 3 = global code address 4 = global data address 5 = local address 6 = local scalar 7 = local code address 8 = local data address
Symbol	2 to 17	A variable-length symbol.
Value	2 to 17	The value associated with the symbol: a variable-length number.

The following figures show how the preceding tables of information might be encoded in Extended Tekhex. The information for the Extended Tekhex Symbol Block illustration (see Figure C-3) could be encoded in a single 96-character block. It is divided into two blocks for purposes of illustration.

Figure C-1. Extended Tekhex Data Block

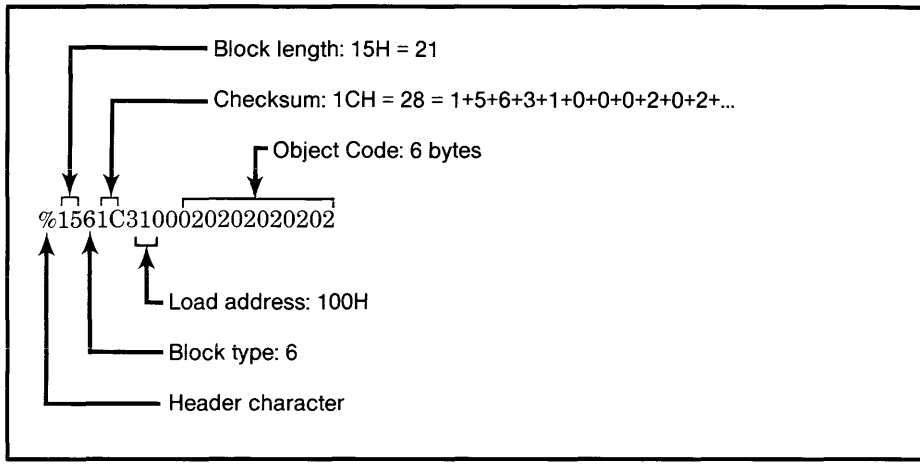


Figure C-2. Extended Tekhex Termination Block

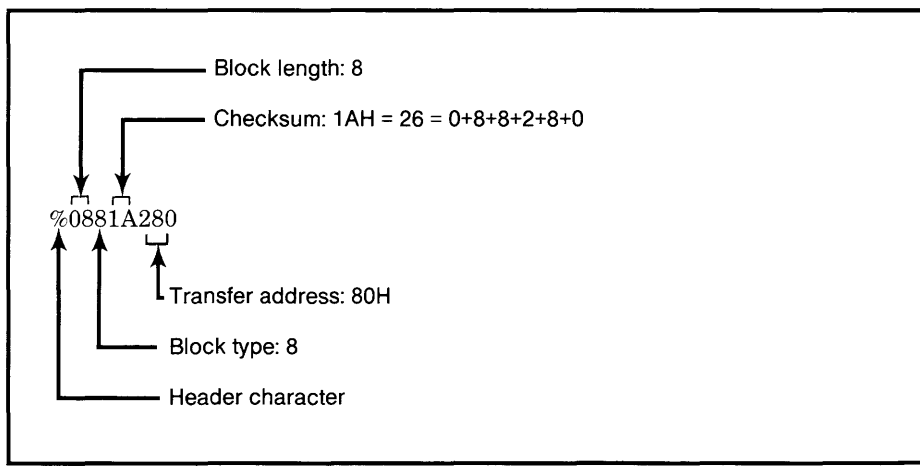
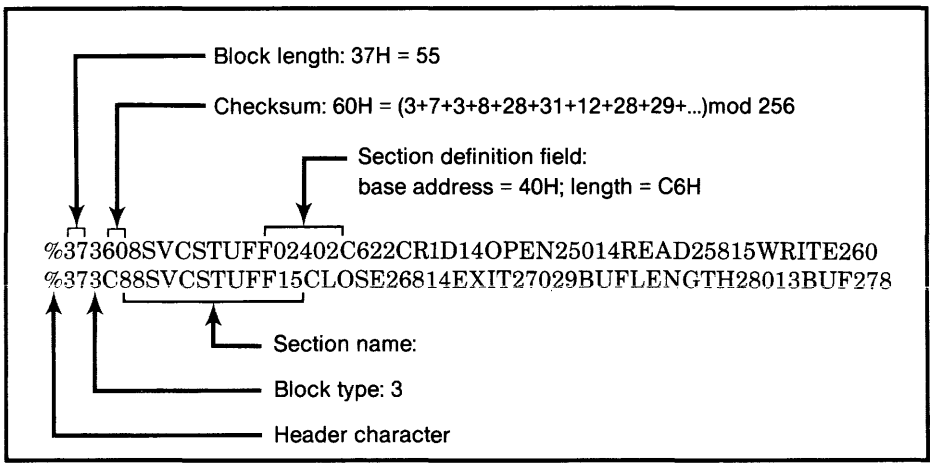


Figure C-3. Extended Tekhex Symbol Block



Motorola EXORciser Format

Motorola data files may begin with a sign-on record, initiated by the code S0. Valid data records start with an eight-character prefix and end with a two- character suffix.

Figure C-4 demonstrates a series of valid Motorola data records.

— Each data record begins with the start characters (**S1**). The emulator will ignore all earlier characters.

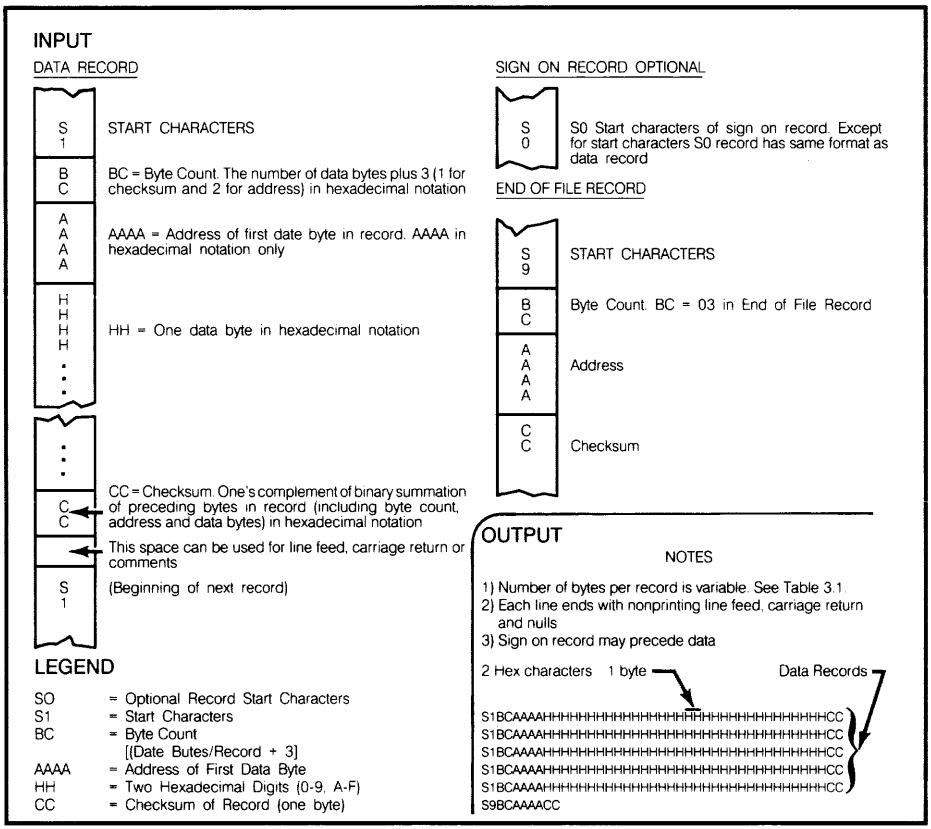
— The third and fourth characters represent the byte count, expressing the number of data, address, and checksum bytes in the record.

— The address of the first data byte in the record is expressed by the last four characters of the prefix.

— Data bytes follow, each represented by two hexadecimal characters. The number of data bytes occurring must be three less than the byte count.

— The suffix is a two-character checksum.

Figure C-4. Specifications for Motorola EXORciser 16-Bit Data Files



NOTE

S2, S3, S7 and S8 records are also accepted. S2 and S8 records have 24 bit addresses (six address characters) rather than the 18 shown. S3 and S7 records have 32 bit addresses (eight address characters) rather than the 18 shown.

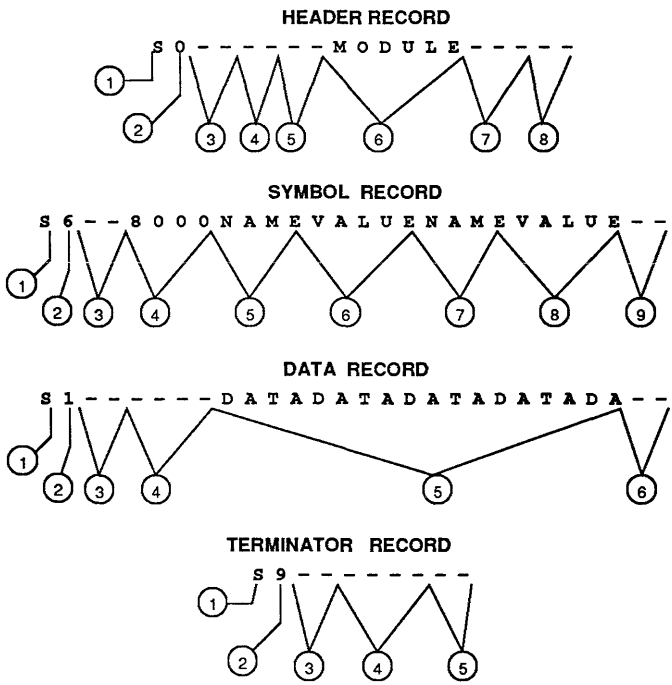
Microtec-Hitachi S-Record Format

Microtec-Hitachi S-record files may contain up to nine object module segments, one symbol segment (optional) and a maximum of eight data segments.

Each segment must begin with a header record, contain symbol records or data records, and end with a terminator record.

The structure of, and the information contained in each type of type of record, is shown in Figure C-5.

Figure C-5. Specifications for MRI-Hitachi S-Record Files



In the header record, 1 is always an "S", 2 is always a 0, 3 is the record byte count, 4 is 01 if more data segments follow and 00 if no data segments follow, 5 is the segment base address, 6 is the module name, 7 is "SYM" if a this is a symbol segment and 000-007 if this is a data segment, and 8 is the record checksum.

In the symbol record, 1 is always an "S", 2 is always a 6, 3 is the record byte count, 4 is always 8000, 5 and 7 are symbol names, 6 and 8 are symbol values, and 9 is the record checksum.

In the data record, 1 is always an "S", 2 is always a 1, 3 is the record byte count, 4 is the load address, 5 is a maximum of 16 bytes of data, 6 is the record checksum.

In the terminator record, 1 is always an "S", 2 is always a 9, 3 is the record byte count, 4 is the program starting address, and 5 is the record checksum.

Intel Hex Format

There are four types of records which can be used in an Intel hex object file:

1. Extended address record
2. Start address record
3. Data record
4. End of file record

Records begin with a colon (ASCII 3AH) and end with a checksum field. The checksum is the ASCII value of the two's complement of the eight-bit sum of the eight-bit bytes resulting from converting each pair of ASCII hex digits to 1 byte of binary. The checksum uses the values beginning with the record length and ending with the last byte of the data field. The binary sum of all the ASCII pairs in a record (including the checksum and excluding the leading:) is zero.

Extended Address Record

Record mark	*Byte count*	*Zeros*	*Record type*	*Upper segment base address*	*Check sum*
:	02	0000	02	xxxx	ss

Data Record

Record mark	*Byte count*	*Load address*	*Record type*	*Data*	*Check sum*
:			00	dd...dd	ss

Start Address Record

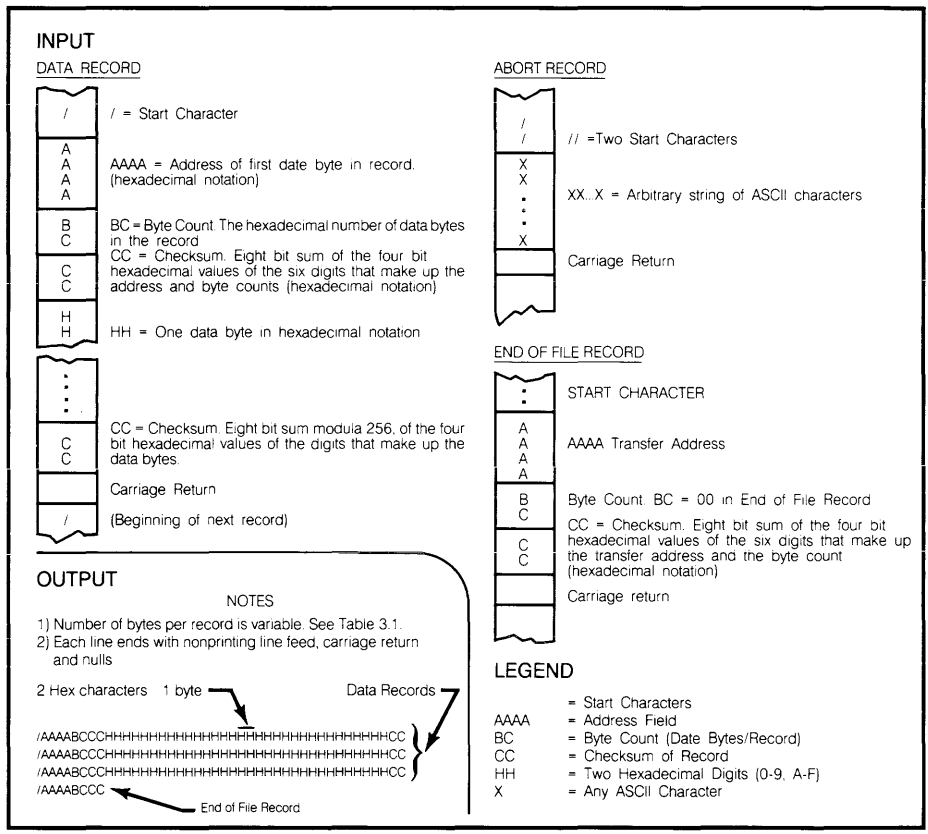
Record mark	*Byte count*	*Zeros*	*Record type*	*CS*	*IP*	*Check sum*
:	04	0000	03	xxxx	yyyy	ss

End of File Record

Record mark	*Byte count*	*Zeros*	*Record type*	*Check sum*
:	00	0000	01	FF

Tektronix Hexadecimal Format

Figure C-6. Tekhex/Copyright 1983, Tektronix; reprinted by permission



APPENDIX D

Table of Contents

What Happens When ...

WHAT HAPPENS WHEN ...

This appendix summarizes what happens in your EL 800 hardware and software when you turn on your emulator, press the hardware reset button, start your software, initialize the emulator (from the Cover Window), reload the shell code, exit the software, and turn off the emulator.

Hardware Power

Power for the base module of the EL 800 is backed up by a battery, so most settings are preserved between sessions. Software configuration settings are usually saved as you leave the configuration window in the control software. Even if the hardware loses power, the battery will ensure that your configuration settings are secure. See the "Software Startup" information on the next page for a description of which settings from your previous session remain available.

Power-On-Reset Sequence

Power-On-Reset Sequence with Uninitialized RAM

The emulator boot code, the emulator operating system kernel, and a configuration table reside in the emulator's boot ROM.

When you turn on the emulator's power or press the hardware reset button, the emulator's processor begins executing boot code out of the boot ROM, performing system hardware initialization and cyclic redundance checks (CRC) on various memory locations, starting with the the system memory configuration table in RAM. Because RAM has not been initialized, the CRC of the system memory configuration

table fails.

When the system memory configuration table CRC fails, program control is passed to the ROM-based emulator operating system kernel, which establishes communication with the host, allowing the host to download RAM-based emulator code, data, and the system memory configuration table.

CRCs are performed on all code and data sections defined by the newly-downloaded system configuration table, and control is transferred to the RAM-based emulator operating system.

Power-On-Reset Sequence with Initialized RAM

When you turn on the emulator's power or press the hardware reset button, the emulator's processor begins executing boot code out of the boot ROM, performing system hardware initialization and CRCs on various memory locations, starting with the the system memory configuration table in RAM. Because the data in this table passes the CRC, the boot code knows that RAM based code has already been downloaded, and continues to verify the rest of the code in RAM against its lists of CRC values.

If all the CRCs match, program control passes directly to the RAM-based code, which establishes a communications channel with the host.

Software Startup

When you start the EL 800 control software, the following events occur:

1. The host keyboard, screen, and search paths are initialized, the database is opened, and the host-emulator link is initialized.
2. The host opens the **???.cfg** file and loads the configuration parameters (communications, user interface parameters, system process configuration). The status message "Loading Configuration Parameters" is displayed.
3. The emulator interface layers are initialized and opened. The emulator's runtime status monitor file STATUS.LOG is initialized. The status message is "Opening emulator interface"
4. The emulator is reset to a known state. The message "Booting Emulator, Please Wait" is displayed.

5. The Advanced Event system setup is cleared; the state is reset to state 1, the counters are set up in the "stop" counting mode, the trace buffer is cleared, trace is set up to the "begin" tracing mode, and the counter values are set to 0. The message "Clearing Advanced Event System" is displayed.

If you wish to restore the emulator configuration from a previous session, you can do so at this point using the **File Access Restore** command.

Emulator Initialization

When you initialize your emulator, using the **I** (Initialize emulator) command from the Cover Window, the following actions are performed.

1. The emulator is stopped (if running).
2. The Advanced Event system setup is cleared, as described above.
3. The emulator soft-switches are set to their default values.
4. The overlay map is cleared.
5. The target reset vectors are loaded into the CPU registers. (Same as the Z-restart command from the Emulate window).

Shell Code Reload

The shell code is loaded at the factory before the EL 800 is shipped. It contains the information necessary to initially start the EL 800 with a particular probe module. You only need to reload shell code if you receive a shell code update or switch probe modules.

Exit from Software (:X) or <ctrl-c<ctrl-c>

If you want to save the current emulator configuration so that you can restore it later, use the **File Save** command before exiting the control software.

Exit the control software before turning off your emulator. When you exit the software using either **:X** (Exit) or **<ctrl-c><ctrl-c>**, the following occurs:

1. The common channels between the host and the emulator are closed.
2. The EL 800 control program is terminated.

If you have battery-backed up overlay modules, any code you store in overlay is preserved between sessions, but you must restore the overlay map manually. See the *Overlay Window* part of Section 6 for more information.

Exit from Windows

The following table describes the types of information saved as you exit various windows in the control software. Window size and position is saved for all displayed windows as you exit the control software.

Assembler	New lines are written to memory as you type them. Symbol changes are entered into the symbol table, but are not saved between sessions unless you use the Save Symbol table command from the File Access Window.
Break/Event	Basic and Advanced breakpoint settings are preserved for the duration of the software session, and may be saved between sessions using the Save Breakevents command from the File Access Window.
Configuration	Communications, user interface parameters, and system process configurations are saved in the configuration file when you exit the Configuration windows. Emulator soft-switch configurations may be specifically saved using the Save Emulator Switches command in the File Access window.
Diagnostics	Nothing is saved from the Diagnostics window.
Emulate	Changes to the Event State Window are preserved during the session. The emulation parameters can be changed while running if the appropriate Advanced Event System statements are specified.
File Access	The default file format is preserved during the session.
Memory Mode	Memory mode changes are saved as you make them, even before you leave the Memory Mode window.

Overlay	The overlay map is only preserved between sessions if you have battery backed-up overlay modules, or if the Save Overlay command is used in the File Access Window.
Registers	The register values are updated when you leave the Registers window, and are not preserved in the emulator between sessions.
Symbol Table	The symbol table is preserved for the duration of the session. To save it between sessions, use the Save Symbol table command from the File Access Window.
Trace Display	The position in the trace buffer is maintained during a session, but is reset when you press the hardware reset button. You can save the trace buffer contents with the Save Trace command in the File Access Window. You cannot restore the trace buffer.
Watch	Changes to the Watch window are preserved during the emulation session.

Hardware Power Off

Make sure that you have saved any emulator configurations you want to restore later with the File Save command before turning off the power to your emulator. The only information automatically preserved between emulating sessions is the data residing in battery-backed up overlay memory. Even with this data, when you return power to the emulator the overlay memory must be remapped for the data in overlay to be meaningful.

APPENDIX E

Table of Contents

Error Messages

ERROR MESSAGES

Error Messages

Many times you will find yourself in a situation that does not seem to be described in the help files or the body of this manual. In these situations, this appendix might help you. It contains a list of error messages, with possible causes and recommended resolutions. The messages are listed in alphabetical order so you can quickly find the resolution to any error displayed on your screen.

Error Message	Description
A Virtual Connection has Failed and One or More Messages Have Been Lost	The communication link between the host and the emulator has failed. Check all cable connections and baud rates, and make sure that the emulator power is on.
Attempted Diagnostic In Run	You cannot execute a diagnostic during run mode. Go to the Emulation window and stop the processor, then execute the diagnostic.
Bad Address Attribute	You have entered an address space that is not valid. The valid memory spaces are **mem** and **io**.

Bad Interface to Emulator

This error will occur if you did not successfully establish communication with the emulator, but you asked to proceed into the control program anyway by responding to the continue dialog box with Ignore. At this point you cannot execute commands that must communicate with the emulator.

Boot Failed

An error was detected while attempting to download or run the emulator RAM-based software. Verify that the "rlconfig.dat" file has not been corrupted. Make sure that the files referenced in rlconfig.dat are in the proper directory, readable, and uncorrupted. If corrupted, reload the files listed in **rlconfig.dat** from the distribution disk.

Can't Delete Module

You cannot delete a module without first deleting all of the symbols associated with it. Delete symbols from the Symbols window (**:SD***).

Can't Find LCA Download Data
Can't Find Pod Download Data

Data in the battery backed RAM has been corrupted. Verify that the proper revision of emulator code is downloaded by entering a **<ctrl-c>** to bring up the Cover window and select the "Reload shell code" option.

Cannot Execute System Process

The process you have selected in the Configuration/System window can not be executed. Either the file could not be found in the directory or there is not enough memory to run the process.

Cannot Process Trace Data Request	The trace line number requested resides outside the currently valid trace buffer. Select a smaller trace line number.
Communications Link Not Found	The requested physical communication device could not be found. Make sure the DIP switches on the adapter cards in the host are set properly. Verify that the "*???*.cfg" (*???* is either z80 or 64180) file is in the proper directory, readable, and uncorrupted.
Communications Service Receive Message Timeout	A message expected from the emulator has not been received. Verify the communication link between the host and the emulator by checking all cable connections and baud rates, and make sure that the emulator power is on.
Communications Service Send Message Timeout	An acknowledgement that a message transmitted to the emulator has not been received. Verify the communication link between the host and the emulator by checking all cable connections and baud rates, and make sure that the emulator power is on.
Configuration Database Not Found	The configuration database file *???*.cfg (*???* is either z80 or 64180) cannot be found. Make sure the file is in the proper directory, readable and uncorrupted.

Doesn't Own Any Symbols	The module for which you have requested the symbols displayed does not have any symbols associated with it. Make sure you have typed the module name correctly or that you have downloaded the symbols.
Emulator Communications Protocol Error	The emulator is not responding as expected. Check all cable connections and baud rates, and make sure that the emulator power is on. If necessary, reload the proper revision of emulator code by entering a **<ctrl-c>** to bring up the Cover window and selecting the "Reload shell code" option.
Emulator RAM Configuration Table Error	Verify that the **rlconfig.dat** file has not been corrupted. Make sure that the files referenced in **rlconfig.dat** are in the proper directory, readable, and uncorrupted.
Entry Can't be Interpreted	The value entered cannot be processed. Make sure you have not mistyped the value. A character in place of a digit will cause this error.
Event System Limitation Error	There are not enough event system resources to satisfy the last event system request. Delete all unnecessary comparator values and WHEN/THEN statements and try again. If that doesn't work, try to combine two or more statements into a single complex statement or try to break a complex statement into two or more simpler statements.

Expression Syntax Error	The expression analyzer cannot process the expression entered. You may have entered an illegal operator or mistyped something.
Help File Not Found	The help file **el***???***.hlp** (*???* is z80 or 64180) cannot be found. Make sure the file is in the proper directory, readable and uncorrupted.
Illegal Access Breakpoint Detected	Emulation was stopped because the target program attempted to read from or write to a memory address mapped as illegal.
Illegal Assembler Label	You have tried to assign an assembler label to a reserved string or have entered an illegal character in the label name.
Illegal Digit in Number	You have entered a digit that is not valid in the current radix. For example, the digits 8 and 9 are not valid if the radix is octal (base 8).
Illegal File Format	You have tried to download a file that is not in the correct object module format or you have tried to restore a file of the wrong window type. Check the object format specified in the **File/Parameters** window. You cannot, for example, restore a file saved from the **Breakevents** window to the **Overlay** window.
Illegal Register Access In Run	An illegal attempt was made to access target registers during run mode.

Improper Operand	The line entered is not valid syntax for the assembler. Make sure you have entered the op code and operands correctly.
Improper Operand Type	You have tried to perform an operation that is not valid on the operand specified.
Insufficient Memory	The host operating system was unable to allocate sufficient memory to process the command. The memory in the host system may have become overly fragmented, or you may want to terminate some of your terminate-and-stay-resident programs if they use a lot of memory.
Insufficient Overlay	There is not enough overlay memory to complete this request. Clear all memory segments you are not using and try again. You can order additional overlay modules by calling an Applied Microsystems Corporation sales office.
LCA Download Error *LCA Program Error* *LCA Verify Error*	The emulator hardware is not responding as expected. Turn off the emulator power for a few seconds and try again.

Mismatch: Probe, Software

The EL 800 control program and the software in the emulator are not of the same type. Make sure you are running the correct control program, elz80.exe for the Z80 microprocessor and el64180.exe for the HD64180 microprocessor. If you have changed the emulator probe module you must re-download the emulator software by entering a **<ctrl-c>** to bring up the Cover window and select the"Reload shell code" option.

No Configuration File

The file "*???*.cfg" (*???* is either z80 or 64180) could not be opened or an error occurred while reading it. Make sure this file is in the proper directory, readable, and uncorrupted.

No Memory Left for Operation

The host operating system was unable to allocate sufficient memory to process the command. The memory in the host system may have become overly fragmented, in which case you must re-boot the host. Another possibility is that you have too many terminate-and-stay-resident programs, which you may want to remove if they use a lot of memory.

No Response to Close Request

This message will appear if you turned off the emulator power or disconnected the communication cable between the host and emulator before exiting the EL 800 control program. The emulator cannot respond to a host request if it is not running.

No Response to Open Request	The emulator is not responding as expected. Turn off the emulator power for a few seconds and try again. If it fails again, re-download the emulator code by entering a **<ctrl-c>** to bring up the Cover window and select the "Reload shell code" option.
No Target VCC	The emulator detects no power in you target system.
NOT a Real Time operation	The REALTIME emulation switch is turned on and the requested command would have halted emulation. You must stop emulation before you enter this command, or you must turn off the REALTIME switch in the Configuration/Emulator window.
Out of Memory	The host operating system was unable to allocate sufficient memory to process the command. The memory in the host system may have become overly fragmented, or you may want to terminate some of your terminate-and-stay-resident programs if they use a lot of memory.
Overlay Map Exhausted	There is not enough overlay memory to complete this request. Clear all memory segments you are not using and try again. You can order additional overlay modules by calling an Applied Microsystems Corporation sales office.

Overlay Module not Installed

You must have an overlay module in order to map overlay memory or to set basic breakpoints. You can order overlay modules by calling an Applied Microsystems Corporation sales office.

Overlay Not Mapped Here

The command you requested attempted to access overlay memory at an address that has not been mapped.

Physical Device Sync Failure

The emulator has not responded to the EL 800 control program. Check all cable connections and baud rates, and make sure that the emulator power is on. See page 2-20.

Pod Fault
Pod Idle Error

The emulator pod did not respond to a request as expected. Reset the pod with one of the following commands. Note that these commands may also change the state of the emulator and its hardware.

- "Z-restart" (Emulation window)
- "Initialize emulator" (Cover window **<ctrl-c>**)
- Push emulator reset button
- Turn off emulator power

If you still get a pod error, re-download the emulator code by entering a **<ctrl-c>** to bring up the Cover window and select the "Reload shell code" option.

Restart Message Received from Emulator

The emulator has unexpectedly re-booted itself and the current emulation session is lost. The host will attempt to re-synchronize with the emulator, but if this does not

	work, turn off the emulator power or push the emulator reset button and restart the control software.
Shell Escape Not Permitted	You cannot execute a shell escape while the emulator is in run mode. Go to the Emulation window and halt emulation, then try again.
Switch to Boot Command did not Execute Properly *Switch to Soft Shell Command did not Execute Properly*	The emulator is not responding as expected. Turn off the emulator power or push the emulator reset button and try again. If you try the above and still get a switch error, re-download the emulator code by entering a **<ctrl-c>** to bring up the Cover window and select the "Reload shell code" option.
Target Address Out of Range	The address specified is larger than the target CPU's address range. You must enter a smaller address value. If you are using a Z80 emulator, check the state of the EXTADDR (Extended Address) soft-switch in the Configuration/Emulator window.
Target Diagnostic is Running	This command is illegal while a target diagnostic is running. Go to the Diagnostics window and terminate the test, then try again.
Target DREQ0 Asserted	The emulator is reporting that the target DREQ0 line is being held asserted.
Target DREQ1 Asserted	The emulator is reporting that the target DREQ1 line is being held asserted.

Target HALT Asserted	The emulator is reporting that the target HALT line is being held asserted.
Target RESET Asserted	The emulator is reporting that the target RESET line is being held asserted.
Target WAIT Asserted	The emulator is reporting that the target WAIT line is being held asserted.
The Target Power is Off	The emulator detects no power in your target system.
Unable to Find the Specified Bootfile	Make sure that the files referenced in **rlconfig.dat** are in the proper directory and are readable.
Unable to Open ECL Channel *Unable to Open the ERROR/STATUS Channel*	The emulator is not responding as expected. Turn off the emulator power or push the emulator reset button and try again. If you try the above and still get an open channel error, re-download the emulator code by entering a **<ctrl-c>** to bring up the Cover window and select the "**R**eload shell code" option.
Unable to Read Bootfile	Make sure that the files referenced in the **rlconfig.dat** file are uncorrupted.
Unexpected Restart Message From the Emulator	The emulator has unexpectedly re-booted itself and the current emulation session is lost. The host will attempt to re-synchronize with the emulator, but if this does not work, turn off the emulator power or push the emulator reset button and restart the control software.

User Break Requested	The requested command was interrupted by the user so the command will not be completed.
Value Out of Range	The value entered is too large for the register. Make sure you are assigning a value to the proper register, and that you haven't mistyped the value.
Verify Fail *Verify Error Detected*	The emulator tried to read back data it wrote to memory, and it did not match. Make sure the memory you are trying to modify resides in target RAM or is mapped to overlay RAM.

APPENDIX F

Table of Contents

Using Expressions

USING EXPRESSIONS

This appendix describes the type of expressions that the expression analyzer can process. Any address or data value can be referred to by an expression. Expressions are symbols, register values, and constants combined arithmetically or using C language constructs. To use the expression analyzer, press the space bar when the control software prompts you for address, symbol, or data value.

Values

Numerical values used in an expression must start with a numerical digit or a decimal point (.). The control software assumes that the number is hexadecimal unless you specify a different radix. You can specify the radix of a number by preceding the value with a 0 (zero), and a classification character. The following forms are recognized:

0Xnnnn	A hexadecimal constant
0Lnnnn	A decimal constant
0Onnnn	An octal constant
0Nnnnn	A binary constant.

The classification character (**X, L, O, or N**) may be upper or lower case. For historical reasons, another override specification is recognized:

.nnnn	A decimal constant

You can also use character constants in expressions by placing the character of interest in single quotes. Non-printable characters may be used by placing their octal representation, preceded by a backslash, in single quotes. The common C language escape sequences are also recognized. Some valid character constants:

'A'
'\060' (The character '0')
'\t' (The tab character)

The data type of a value is inferred from its size in binary notation. A value that may be represented in one byte is considered a BYTE, a two byte value is a WORD, longer values are LONGs.

Symbols

You may insert a symbol into an expression anywhere a value could be used. If the symbol is the name of a label, whether data or code, then its value is directly inserted into the expression. If the symbol is a scalar, the value at the referenced memory location in the target is inserted into the expression.

One of the symbol's attribute's is its *data type*. Certain operators expect a symbol to be either a label or scalar. The symbol's data type may be overridden with a cast operator.

Symbol names are case sensitive; **data**, **Data**, and **DATA** are three different symbols.

Register names

The control software recognizes the names of the target CPU and can use these values in expressions. When you enter a register name, the expression analyzer fetches that register's value from the emulator. Registers have data types just like symbols.

If a symbol is defined with the same name as a register, the registers value will no longer be accessible. Note that register names are not case sensitive, for example, "pc", "Pc", "PC", "pC" all refer to the same register. In this example, a symbol named **PC** would eliminate the possibility of referring to the PC register, but referring to the same register through another variation of capital and lower case naming, such as **Pc** would still be legal.

Operators

The expression analyzer recognizes the usual C language operators shown in the following chart, in order of precedence (with operators on the same line having equal precedence):

1.	( )					
2.	!	~	-	(type)	*	&
3.	*	/	%			
4.	+	-				
5.	<<	>>				
6.	&					
7.	^					
8.	|					

Consult Kernighan and Ritchie's *The C Programming Language* for complete descriptions.

Type Casting

The operator shown as **type** in the list above is a typecast operator. It tells the expression analyzer that the following value in the expression should be treated as if it were of the specified type. Examples:

(byte) 1234	Treat 1234 as a byte (it is truncated to 34)
(long) 23	Treat 23 as a long (it is padded to 00000023)
(word *) 0	Treat 0 as a pointer to a word in the target memory (it is transformed into the appropriate address representation for the target)

Memory Access

Access to target memory is provided through the C pointer operators * and &. * dereferences a pointer, making the expression equal to the value at the address pointed to. In the following examples, assume that **xPW** is the address of a WORD in target memory with a value of 0x1234.

*xPW	→	1234	
* (byte *) xPW	→	34	(Assuming "little-endian" order, where the least significant byte precedes the most significant byte)
* (byte *) 0	→	12	(Assuming 12 is at address 0)

The unary & operator causes insertion of its operand's address into the expression. For example, if TIME is a variable stored at location 0x0002 in target memory, then:

& TIME

has the value 2.

Expressions

Expressions are entered normally, and read left to right (infix notation). You can enter any C expression that uses the operators above with one of the control software's unique modes. Examples:

1 + 2 * 4	→	9
- - 2	→	2

Formats

In some cases, you can append format specifiers to an expression. Expression results may be viewed in a variety of formats. The format notation is adopted from the C standard library function **printf( )**:

d	decimal integer
o	octal integer
x	hexadecimal integer
c	character
s	C style string
n	binary integer

Note that length specifiers are not used or accepted; the length of the expression is known by the time it is printed.

The expression analyzer zero-pads non-decimal values. The number of zeros used in non-decimal values depends on the mode of the value.

Format specifiers are added to expressions by preceeding them with a comma.

Examples:

2 + 3, x	→	05
100, d	→	64
3, n	→	00000011
strAB, s	→	hello (strAB points to "hello")

Repeat Counts

When the control software allows format specifiers, it will also allow you to specify a repeat count. Repeat counts are always coupled with format specifiers; the count values are placed immediately before the format specifier in the expression. Repeat counts can only be used with expressions that evaluate to pointers. The following examples assume that there is a string of bytes with values **0, 1, ...** at location 0 in target memory.

(byte *) 0, 2 x	→	01 02
(word *) 0, 2 x	→	0201 0403
0, 4 x	→	ERROR (0 is not a pointer)

A repeat count may be coupled with the string format specifier. In this case, you are implying that an array of pointers resides at the specified value. Each element of that array points to a character string for the "s" format. As an example, suppose the control software is in use on a C program. If **argv** is an array of pointers to strings "cc", "-c", and "test.c", then the following expression could be set up:

(byte **) argv, 3 s	→	cc -c test.c

APPENDIX G

Table of Contents

Debugging Multiprocessor Systems

DEBUGGING MULTIPROCESSOR SYSTEMS

This section provides suggestions for using the Advanced Event System with other instruments and for multiprocessor debugging. Basic information on the structure of the Advanced Event System and on how to use it is described in the *Break/Event Window* part of Section 6.

Trigger Inputs and Outputs

The EL 800 can output a trigger signal used to trigger another instrument, such as a logic analyzer, oscilloscope, or another emulator. In addition, it can recognize input triggers from another instrument or your target system. There are two trigger outputs and two trigger inputs on the base module. The following figures show how to configure the EL 800 to trigger an oscilloscope and how to use the trigger as an additional line from a target board.

Figure G-1. Using the Trigger Output with a Oscilloscope

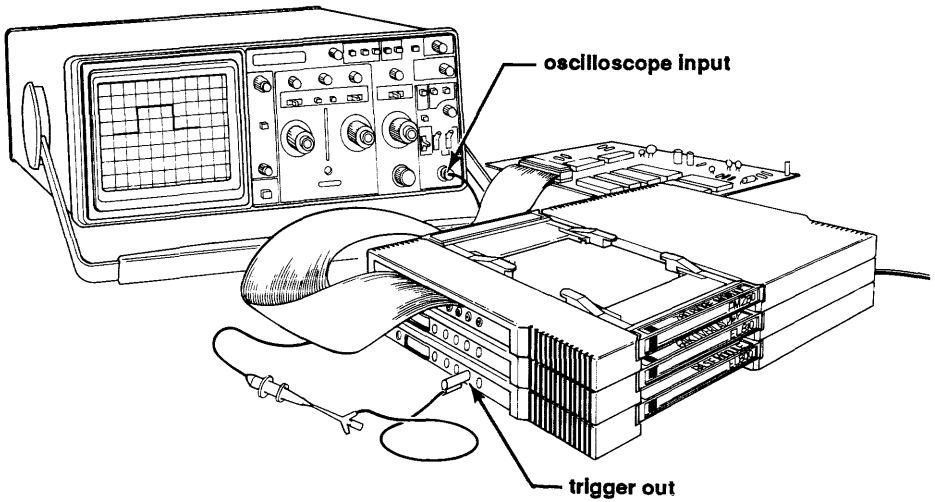
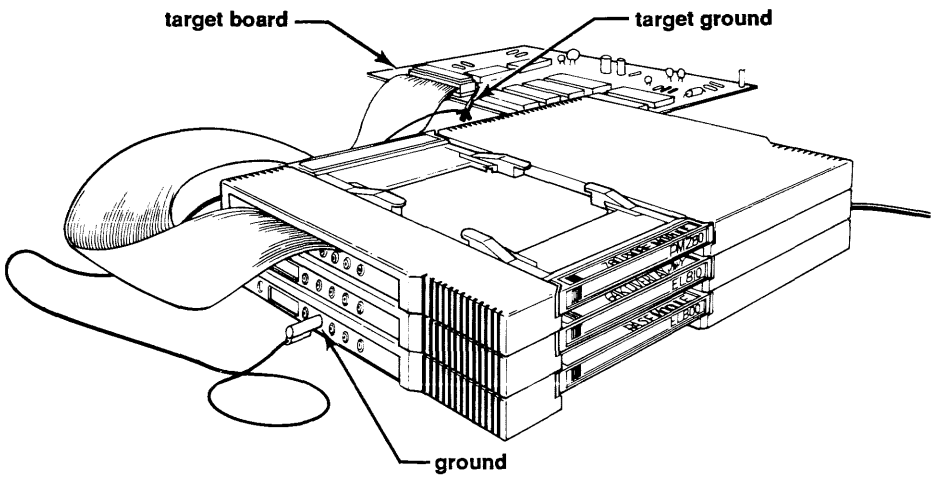


Figure G-2. Using the Trigger Input with a Target Board



Debugging MultiProcessor Systems

Debugging multiprocessor systems can be difficult, since knowledge of the state of one processor does not usually give you complete information about the state of the other processors. The only information available to each processor about the state of the others is contained in the data passed between them. Debugging is easier if you have a mechanism for determining, at any given instant, the state of all processors in the system.

If you have in-circuit emulators for all microprocessors you gain visibility and control over the processors and their environments, as well as the capability to allow events in one processor's environment to qualify or control the events in the environment of another.

This section describes using the EL 800 with another emulator, the ES 1800, to debug designs combining 8 and 16 bit microprocessors. It includes two sample scenarios, with complete information on setting up the event systems of each emulator:

1. Debugging interrupt handling done by a slave processor
2. Debugging disk handling done by a series of slave processors

Advanced Event System Use

The ES 1800 has one output trigger. With the optional Logic State Analyzer pod, 16 input triggers are available. The EL 800 has two input and two output triggers.

Asynchronous trigger input signals can be logically combined with address, data, status and counter information to qualify event system actions. For example, upon reaching a specified condition including a trigger signal from an emulator or logic analyzer, the event system can perform any of its actions.

When a condition based on address, data, status, counters and trigger inputs is reached, one possible action is to send a trigger signal out to another instrument.

Common Multiprocessor Debugging Situations

Problems in multiprocessor systems are often related to the transfer of commands and data between the processors. Care must be taken that two processors do not try to access memory simultaneously, that the handshake signals are correctly interpreted on both sides and that the command/data formats and locations are defined properly for both processors.

Two common kinds of problems found in multiprocessor systems are described in the following pages. Following each problem description is a way to use the emulators' event systems to catch the problematic situation and quickly determine its cause.

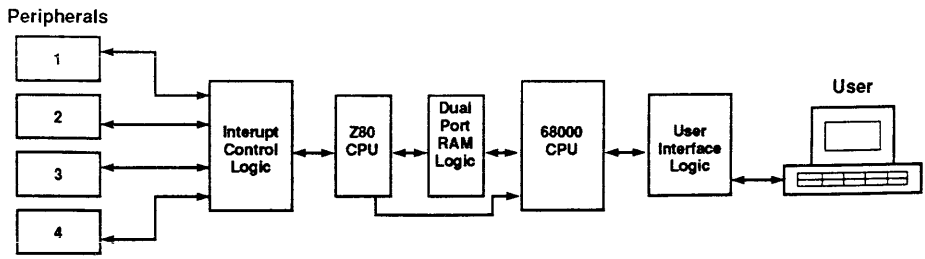
Each example describes cross-triggering an ES 1800 emulator and an EL 800 emulator. The examples shown are emulator specific, but the principles are general and can be applied to multiple processor designs involving any microprocessors currently supported by the ES 1800 and EL 800 series of emulators.

Let's assume in the following examples that you have a system in which there are multiple CPUs: one 68000 and one or more Z80s. The processors must communicate with each other in order to get a job done.

Scenario 1

Let's look at an example of a data acquisition scheme in which a Z80 is being used to off-load the serial interrupt task from a 68000. In the diagram below you can see that there are four peripheral interrupt sources. The Z80 intercepts these interrupts and does the processing necessary to package up a complete command before interrupting the 68000 from its task of processing the user's commands. When interrupted, the 68000 will retrieve the information from the dual port RAM.

Figure G-3. Block Diagram of Scenario 1



Symptom

You have determined that the 68000 is receiving the interrupt from the Z80, but the resulting action on the 68000 side seems to indicate that the information it receives is incorrect.

Solution

One approach is to look at the data in the dual port RAM and the data in the Z80's and 68000's storage buffers to see if it looks correct in all three places.

There are four steps:

1. Connect the ES 1800 output trigger to an EL 800 input trigger.
2. Set up the ES 1800 event system.
3. Set up the EL 800 event system.
4. Run your programs and when a breakpoint occurs, observe the data in both processors' memory regions. This will help you determine:
   - if data is being corrupted somewhere during the processing cycle
   - if the Z80 incorrectly packages the incoming data
   - if the 68000 is misinterpreting the data it receives
   - if there seems to be a hardware problem

Step 1:

Electrically connect the ES 1800 output trigger to one of the EL 800 input triggers.

Step 2:

A good place to stop execution would be the last instruction in the 68000's interrupt service routine. At this point we know that the information written into the dual port RAM by the Z80 will have been read from the dual port RAM by the 68000 and stored into its buffer area.

We want the ES 1800 to output a trigger signal before it stops executing. To set up the event system to do this, define the condition to be the 68000's interrupt return instruction and the actions to be an output trigger signal and a break. The output trigger will be used to stop the Z80's execution.

ES 1800 Event Statement	*Description*
AC1='isr +$54	Set an address comparator (AC1) to the last instruction in the interrupt service routine. (The symbol 'isr is the beginning of the interrupt service routine, and $54 is the length of the routine).
WHEN AC1 THEN TGR, BRK	When the address (AC1) is reached, output a trigger signal (TGR) and break execution (BRK).

Step 3:

Now set up the EL 800 event system to break execution of the Z80 when the trigger input is received. From event state 1, enter the EL 800 event specification using the keystroke sequence indicated in the following chart.

EL 800 Event Statement	*Key Sequence*	*Description*
WHEN trigIn1 THEN Break	**wwiltb**	When a trigger signal is received, break execution.

Step 4:

After both systems have broken emulation, the emulators can be used to compare the data in dual port RAM with the data that the 68000 read from the dual port and saved in its own buffer. In addition, if the data has not been overwritten by more incoming peripheral data, we may still be able to see the original data received from the peripheral in the Z80's storage buffer.

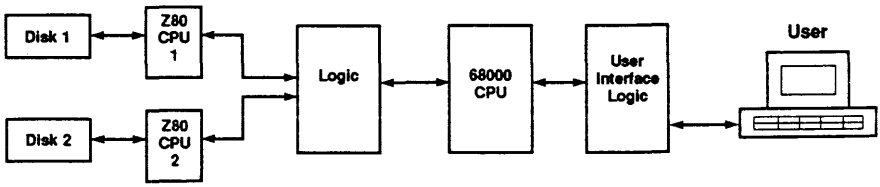
From this data you can determine:

- if data is being corrupted somewhere during the processing cycle
- if the Z80 incorrectly packages the incoming data
- if the 68000 is misinterpreting the command or data
- if there seems to be a hardware problem

Scenario 2

Let's assume you have a system like the one shown below, with a master 68000 that processes user input and two slave Z80 disk driver subsystems.

Figure G-4. Block Diagram of Scenario 2



Symptom

The behavior you are seeing is that every now and then one of the disk drivers returns an 'illegal command' error to the 68000.

To make it interesting, let's say that if an error occurs, the whole system must be halted in order to avoid crashing any disks. With the ES 1800 emulator, you can use the "forced special interrupt" feature to execute a safe shutdown routine before stopping program execution.

Solution

To debug this problem, you need three emulators: one EL 800 for each Z80 and one ES 1800 for the 68000.

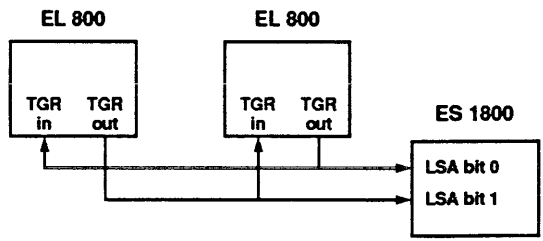
There are four steps:

1. Connect the EL 800 output triggers to the ES 1800 LSA pod input triggers.
2. Set up each EL 800 event system.
3. Set up the ES 1800 event system.
4. Run your programs. When the breakpoints occur, observe the data in the emulator which triggered the break condition. This data will show which Z80 is causing the problem, as well as help identify what exactly is causing the problem.

Step 1:

Electrically connect one output trigger from each EL Z80 emulator to the lowest two bits of the ES 1800's LSA pod, and to one of the input triggers of the other EL Z80 as shown in the diagram below.

Figure G-5. Block Diagram of Trigger Connections for Scenario 2



Step 2:

There are two parts to setting up each EL 800's event system:

1. The EL 800 emulating the processor that receives the erroneous command must send a trigger signal to the other two emulators and then break.
2. The other EL 800 emulators must have an event specification that causes a break when an input trigger is received.

For the first part, set up each EL Z80 to output a trigger signal if it gets an illegal command. We know that an illegal command was received if we begin executing the error procedure "err_proc".

For the second part, set up the event system to break when an input trigger is received.

From event state 1 in each of the EL 800 Z80 emulators, enter the event specifications using the keystroke sequences indicated in the following chart.

EL 800 Event Statement	*Key Sequence*	*Description*
A= err_proc	**as err_proc**	Set address comparator A to the beginning of the error procedure (symbol err_proc).
WHEN addrA THEN Break trigOut1	**wwatbo1**	When execution of the error procedure begins, send out a trigger signal and break execution.
WHEN trigIn1 THEN Break	**wwi1tb**	When an input trigger is received, break execution.

Step 3:

Set up the ES 1800 event system to break if bit 0 or 1 of the LSA trigger inputs goes active. This is done by using two event system groups to specify two distinct program states, one to track each of the LSA bits. We will specify that we do not care about the state of the other bits of the LSA trigger inputs by specifying a "don't care" mask. As long as neither of the LSA bits becomes active, the emulator will toggle between state 1 and state 2 every other cycle.

The ES 1800 can be used to safely shut down the disk drives, using the forced special interrupt (FSI) action to jump to a prewritten soft shutdown program before stopping program execution. The ES 1800 has a special interrupt address register (SIA) to store the address of the prewritten soft shutdown routine.

ES 1800 Event Statement	*Description*
SIA = 'shutdown	Set the special interrupt address register (SIA) to refer to your safe shutdown routine, shown by the symbol 'shutdown.
LSA = 1 DC $FFFE	Set the Logic State Analyzer comparator (LSA) to detect when bit 0 goes "high". We don't care about bits 1-15, so the expression 'DC $FFFE' indicates a don't care mask for these bits.
AC1 = 'end_shutdown	Set an address comparator in group 1 to the end of the shutdown routine, indicated by the symbol 'end_shutdown.
1 WHEN LSA THEN FSI	When the LSA input condition for group 1 is true (bit 0 high), execute the special shutdown routine.
1 WHEN AC1 THEN BRK	When the end of the safe shutdown routine is reached, break emulation.
1 WHEN NOT LSA THEN GRO 2	When the LSA input condition is false (bit 0 low), go to group 2 to check for the bit 1 value.

ES 1800 Event Statement	*Description*
LSA.2 = 2 DC $FFFD	Set the Logic State Analyzer comparator in group 2 (LSA.2) to detect when bit 1 goes high. We don't care about bits 0,2-15, so the expression 'DC $FFFD' indicates a don't care mask for these bits.
AC1.2 = 'end_shutdown	Set up an address comparator in group 2 to the end of the shutdown routine.
2 WHEN LSA THEN FSI	When the LSA input condition for group 2 is true (bit 1 high), execute the special shutdown routine.
2 WHEN AC1 THEN BRK	When the end of the safe shutdown routine is reached, break emulation.
2 WHEN NOT LSA THEN GRO 1	When the LSA input condition is false (bit 1 low), go to group 2 to check for the bit 0 value.

Step 4:

When the emulators break, the reason for the illegal command can be determined by evaluating the trace memory of the machine that triggered the break condition and determining what was wrong with the command.

INDEX

- L -

- M -

- N -

- O -

- T -

Applied Microsystems Corporation

Applied Microsystems Corporation maintains a worldwide network of direct sales offices committed to quality service and support. For information on products, pricing, or delivery, please call the nearest office listed below. If you are unsure which office to contact, call 1-800-426-3925 for assistance.

CORPORATE OFFICE
Applied Microsystems Corporation
5020 148th Avenue Northeast
P.O. Box 97002
Redmond, WA 98073-9702
(206) 882-2000
1-800-426-3925
TRT TELEX 185196
Fax (206) 883-3049

EUROPE
Applied Microsystems Corporation
Chiltern Court
High Street
Wendover
Aylesbury, Bucks
44 (0) HP22 6EP England
296-625462
Telex 265871 REF WOT 004
Fax 44 (0) 296-623460

JAPAN
Applied Microsystems Japan, Ltd.
Nihon Seimei
Nishi-Gotanda Building
7-24-5 Nishi-Gotanda
Shinagawa-Ku
Tokyo T141, Japan
3-493-0770
Fax 3-493-7270

U.S. REGIONAL SALES OFFICES

Western Region
Applied Microsystems
Corporation of Washington
3333 Bowers Avenue
Suite #220
Santa Clara, CA 95054
(408) 727-5433
Fax (408) 727-9011

Applied Microsystems
Corporation of Washington
2101 Business Center Drive
Suite #140
Irvine, CA 92715
(714) 476-3177
Fax (714) 476-8546

Central Region
Applied Microsystems Corporation
Suite #142
Richardson, TX 75081
(214) 235-8827
Fax (214) 238-0719

Eastern Region
Applied Microsystems
Corporation of Washington
6 Cabot Place
Stoughton, MA 02072
(617) 341-3121
Fax (617) 341-0245

P/N 922-00021-02
October 1988