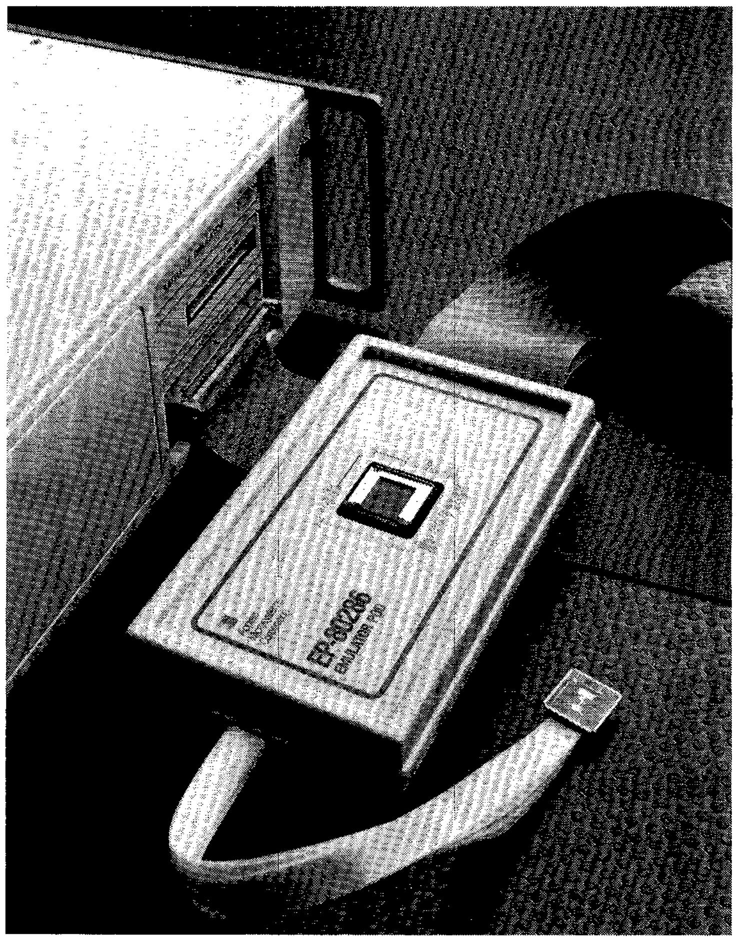
ERRATA:
Page 1-5 of this manual should be corrected. The minimum level for using the Time Stamp Module with an ES 1800 for an Intel 80286 is ESL 2.02 not ESL 2.10.

Applied
Microsystems
Corporation



ES 1800 Satellite Emulator Reference Manual For the 80286 Microprocessor

Applied
Microsystems
Corporation

ES 1800 Satellite Emulator Reference Manual For the 80286 Microprocessor

920-10435-02
March 1988

Table of Contents

Page

Page

Table of Contents

Preface

PREFACE

APPLIED MICROSYSTEMS CORPORATION is proud of its role in the systems development industry and conscious of its important contribution. However, it assumes no liability for errors or for any damages that may result from use of this manual or the equipment it accompanies.

We have made every effort to document this product accurately and completely. We reserve the right to make changes to this manual without notice.

The ES1800 Emulator is intended for use in developing, debugging, and testing Intel 80286 microprocessor-based systems. This manual assumes the user is familiar with the terminology of the 80286 microprocessor.

Unpacking and Inspection

Your Emulator has been inspected and tested for electrical and mechanical defects before shipping, then configured for the line voltage you requested. Although the Emulator was carefully packed, check it for possible transit damage and verify that the following units are present. If you find any damage, file a claim with the carrier and notify Applied Microsystems Corporation (Customer Service 1-800-426-3925).

STANDARD EQUIPMENT

- Emulator chassis with power cord
- Main control boards and pod assembly
- ES1800 Emulator Reference Manual for the 80286 Microprocessor

OPTIONAL EQUIPMENT

- Control Boards
    - overlay memory
    - symbolic debug
- Logic state analysis pod assembly
- Carrying case

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.

Service

If the ES1800 unit needs to be returned for repairs, Applied Microsystems Customer Service will issue a Return Authorization number. To obtain the necessary Return Authorization number and shipping information call 1-800-426-3925, and ask for Customer Service. 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 are 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 transferable.

To obtain warranty service, the customer must notify Applied Microsystems Corporation of any defect prior to the warranty expiration and make arrangements for repair and for prepaid shipment to Applied Microsystems Corporation. Applied Microsystems Corporation will prepay the return shipping to US locations. For international shipments, 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 sales office.

SECTION 1

Table of Contents

Introduction

INTRODUCTION

The ES1800 Emulation system allows you to analyze and control a target environment, consisting of hardware or software, in real time. To use the ES1800 with your target hardware, simply remove the target system's microprocessor and plug in the ES1800 Emulator. Your system uses the Emulator in place of the microprocessor and behaves as if the target microprocessor were there. It continues to run until you manually stop it or it encounters a user-defined stop condition. This predefined condition can be in the form of single-step operation statements or more complex event monitoring (WHEN/THEN) statements.

During the debugging or integration process you can read and write to the microprocessor registers or memory locations and execute programs contained in the target system memory. The ES1800 Emulator also allows you to debug software without being physically connected to the target system. In this configuration, the Emulator uses its own real-time clock feature combined with overlay memory capabilities.

Information in this manual applies to the Intel 80286 microprocessor only. For more complete information on this chip, refer to the *Intel iAPX286 Hardware Reference Manual* and the *Intel iAPX286 Programmer's Reference Manual* published by Intel Corporation.

How to Use This Manual

This manual is your guide to using the Applied Microsystems Corporation's ES1800 for the 80286 microprocessor. For your first time using the ES1800, read through the Introduction and Getting Started sections and refer to the Hardware section to make sure your hardware is set up correctly.

Once you are familiar with the Emulator, Chapters 4, 5, 6 and 7 provide information on all of the available commands. The comprehensive index and Appendix A: ES Language Mnemonics are useful for finding specific information in the manual.

The manual is organized as follows:

Section 1: Introduction introduces Applied Microsystem Corporation's ES1800 Emulator for the 80286. It explains emulation, setup, and configuration requirements, and provides an overview of the features of the ES1800.

Section 2: Getting Started provides a checklist for setting up the Emulator and target system, starting and testing the Emulator, and storing customized system variables in EEPROM.

Section 3: Hardware contains all the information on the Emulator, the control boards, the rear panel, the pod, and the serial ports, as well as information on maintenance and troubleshooting.

Section 4: ES Language explains the structure of the language that controls the Emulator, with explanations of the help menus, prompts, special modes and characters, and language related error messages.

Section 5: System Commands provides a reference to commands that control the Emulator system. It is divided into sections on setup, serial communications, download operations, registers, trace memory, macros, and symbols.

Section 6: Target Commands provides a reference to commands that directly control the target system. It is divided into sections on running the target program, overlay memory commands, the line assembler, the memory disassembler, memory and I/O modes, and special functions.

Section 7: Event Monitor System explains the powerful breakpoint and control system, including the structure of the system, breaking emulation, counting events, using special interrupts, and tracing events.

The **Appendices** include a quick reference to ES Language mnemonics, explanations of the hardware error messages and serial data formats.

System Setup

The ES1800 can debug and integrate software and hardware. Setups for each system may be different. In every combination, there is a "target" system, which can be hardware, software alone (if you are using the Emulator's overlay memory to debug software), or a combination of the two. The target system is the environment you intend to emulate.

The ES1800 Emulator consists of a chassis assembly which houses the control boards and an Emulator pod which houses the emulating microprocessor. The Emulator can be controlled with a terminal, which can be your development system CRT or another device set to function in terminal mode. You can enhance this basic system by adding the optional logic state analyzer (LSA) pod. This provides 16 additional input lines, giving access to signals other than the normal address, data, and control signals of the microprocessor. You may also add an optional overlay memory board. Overlay memory can be mapped anywhere in the address space of the target system. The overlay memory board provides additional capabilities, including the ability to debug software with or without a target system.

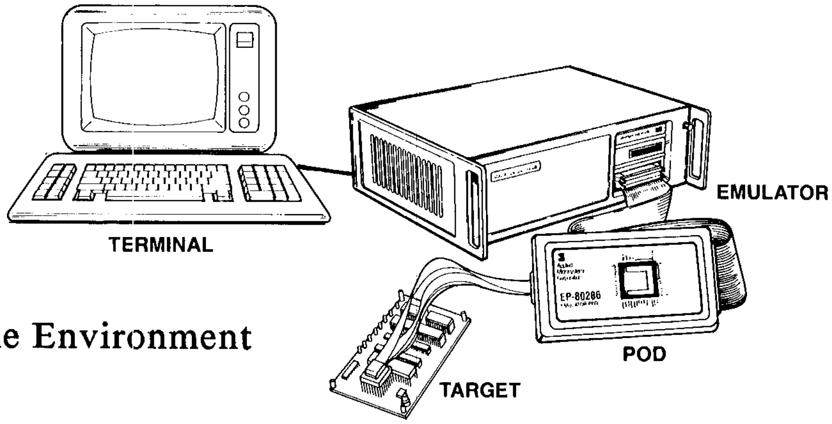
The stand alone environment (refer to diagram on following page) consists of the Emulator and a dumb terminal or equivalent connected to the terminal port. This configuration can debug target systems with software already installed or short, hand-entered routines. The stand-alone configuration is common in manufacturing test and service facilities.

The Emulator can also use data stored in a host development system by setting up a hosted environment (refer to diagram on following page). The Emulator is still under the direct control of the CRT but can load data from the host system's data files.

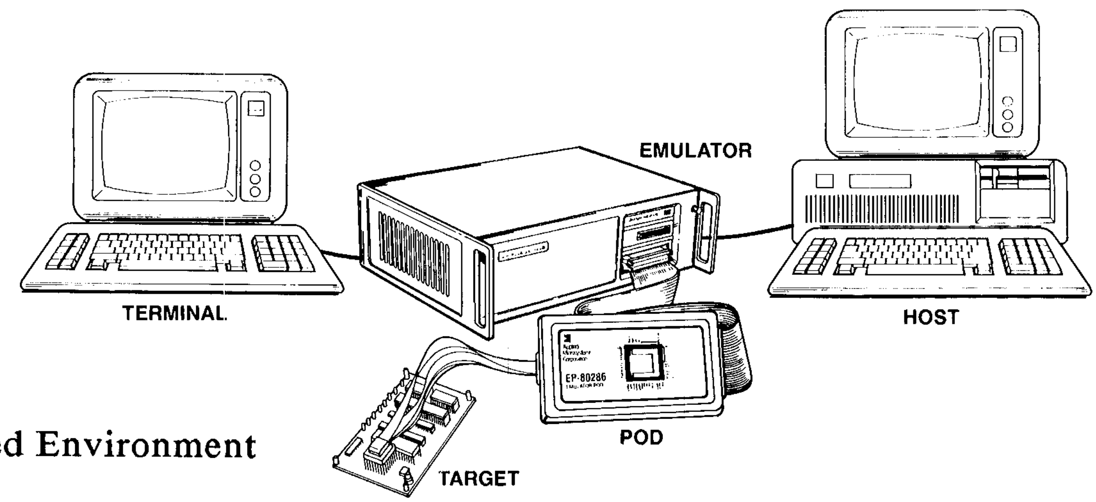
By attaching a printer, data and code from the target system can be printed out in assembly language. You can also print all Emulator commands and their results. The Emulator system has two serial ports and uses standard RS232C serial port protocol. Each port can be independently configured for baud rate, data length, and number of stop bits.

Software for driving the Emulator is available from Applied Microsystems Corporation for the IBM PC and compatibles, SUN, APOLLO and VAX.

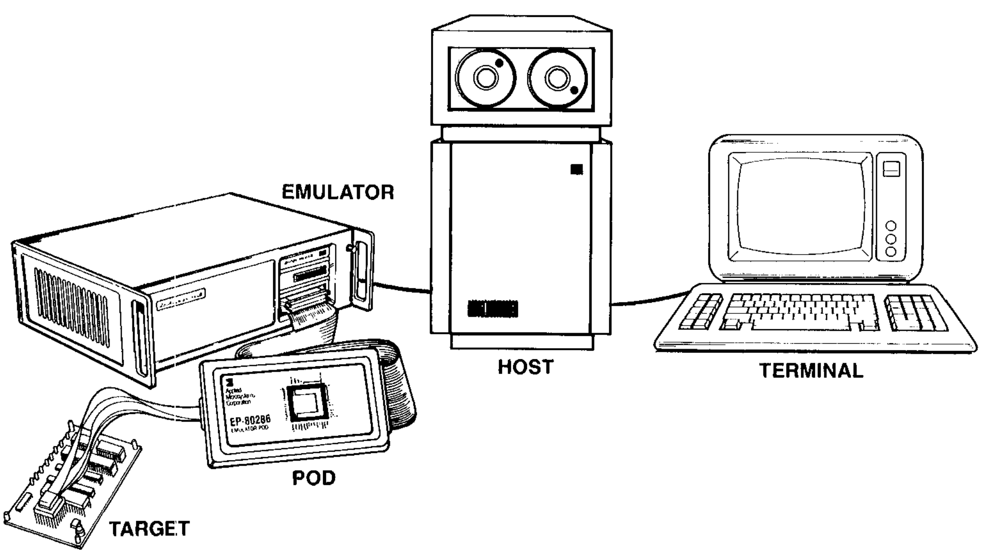
Figure 1. Environments



Stand Alone Environment



Hosted Environment



Hosted Software Environment

The Emulator can also be totally controlled by a host system. This hosted software environment (refer to diagram on previous page) requires special host resident software. Drivers and high level debuggers are available from Applied Microsystems for most languages and host systems.

System Operation

OVERVIEW

The ES1800 has two basic operational modes: emulation and pause. Pause mode is generally used to set up the system configuration and to display information after exiting emulation. System setup is accomplished from two menus. The first menu contains all external communication variables; the second contains the control switches for emulation. Both setups can be saved to EEPROM and automatically loaded at power-up.

Emulation, or run mode, means that the microprocessor in the Emulator pod is running a program in the target system, allowing you to see what is happening within the target system. Emulation stops when (1) you stop it, (2) user-defined breakpoints are enabled and occur, (3) you reset the system, or (4) errors occur in the target system.

When you manually stop emulation or a breakpoint is reached, you enter pause mode. All registers and addresses are then available for examination, along with a trace history of performance of the microprocessor. A command language allows you to enter emulation mode in the desired state and leave emulation when the desired combination of events are detected in the target.

ES LANGUAGE

The ES1800 uses its own command language. To benefit from the sophisticated operations of the Emulator, you must understand the general concepts of this language. The Emulator operates in response to command statements composed of command mnemonics and, for some commands, arguments. An argument to a command is an additional value entered as part of the command sequence, such as an address range or base value. Arguments can consist of single values, expressions, or lists.

The command statements form a control language, similar to higher-level computer languages. And, like a computer language, the operators and values can be combined to form complex expressions. Statements have a maximum length of 76 characters and can be extended by the use of macros.

The ES Language contains registers, counters, and conditional statements allowing the user full control over the operation of the target system. To complete the language, a full set of error messages is provided for (1) target hardware, (2) Emulator hardware, (3) target software, and (4) ES command language syntax.

REAL-TIME

Since the pod processor is identical to the target microprocessor, the target system runs in real time. No wait states are inserted by the Emulator during run mode.

NULL TARGET

When there is no target system, you may select the internal clock feature, which places the Emulator in null target mode. In this mode a 12.5 MHz clock is supplied to the CPU, and unterminated inputs are set inactive. Overlay memory can then be used to develop code as if a target system were attached.

TRACE MEMORY

Trace memory functions as a history of the target system program's execution. This memory can record 2046 bus cycles and display these in assembly language. All address lines, data lines, processor status lines, and 16 bits of external logic input are traced. If something unexpected happens during program execution, trace memory can be reviewed to determine the sequence of instructions executed by the CPU prior to the unexpected event. When used in conjunction with the trace disassembler, hardware and software problems can be quickly tracked down.

Trace memory can be selectively switched on by the Event Monitor to trace events only when certain conditions are met. Program execution can be stopped at any point, either manually or using the Event Monitor System. The address, data, and control signals of the most recently traced cycles can then be critically reviewed.

OVERLAY MEMORY

Overlay memory is Emulator working memory, which can be used in a variety of ways. When debugging software without target hardware, the target program is loaded into overlay memory, where it can be edited and positioned in the target system address space as desired (null target mode). The program executes in real time as if it resided totally in the target system. Overlay memory is also useful when a target is connected, for loading portions of software, making patches, and checking programs not yet committed to PROM.

The overlay memory is RAM with appropriate address and control logic, ranging in size from 32K to 2M bytes and locatable in 2K-byte segments throughout the system. Each segment can be assigned one of four attributes: target, read/write, read-only, or illegal. Unmapped memory is assigned the target attribute by default. Overlay memory mapped as read-only can always be modified by the Emulator operator. However, if a program tries to write to read-only overlay, emulation stops and an error message is displayed. Overlay memory mapped as read/write can be written to or read from. If a program attempts to read or write to memory mapped as illegal, emulation stops and an error message is displayed.

When a segment of memory is mapped, program accesses in that memory range are directed to the overlay instead of the target. Overlay memory accesses occur in real-time, with no wait states added by the Emulator.

EVENT MONITOR SYSTEM

The ES1800's Event Monitor System provides unprecedented breakpoint and system control, enabling the user to isolate and break on any predefined series of events and then perform actions defined by WHEN/THEN conditional statements. The user controls and monitors the target with the Event Monitor System by defining statements that specify exact or multiple events through logical combinations of address, data, status, pass counter, and optional logic field states. When those events are encountered in the target system program, the ES1800 can break emulation, trace specific sequences, count events and trigger outputs all independently, allowing the user to analyze the cause-effect relationship established by the event/action sequences defined.

The Event Monitor System uses four groups containing eight registers each to let the user monitor a complex series of events through multiple actions and combinations of comparator registers. The system uses one group at a time, with each WHEN/THEN statement active in a specific group. WHEN/THEN statements can switch to different groups and access conditional statements and registers for that group. The user can control the tracing of 2046 machine cycles, selecting the desired instructions to be recorded in the trace memory.

OPTIONAL SYMBOLIC DEBUGGER

The symbolic debug option allows you to assign frequently used values to symbol names that make sense. Features include:

- Reference to an address by a name instead of a value
- Display of all symbols and sections with their values
- Editing (entry and deletion) of symbols and their values
- Automatic display of symbolic addresses during disassembly
- Section (module) symbols that can be used as range arguments and for section offsets in trace disassembly
- Upload and download of symbol and section definitions using standard serial formats

Because symbols are a powerful extension of the Emulator, they are frequently used in examples throughout Section 5, System Commands. Please note that if you have not yet purchased the symbolic debug option, you may need to modify these examples.

OPTIONAL LOGIC STATE ANALYZER (LSA)

LSA inputs can qualify event specifications in the Event Monitor System. In the simplest form, specific bit patterns at the LSA inputs can cause a breakpoint. The LSA comparator can detect arbitrarily complex event specifications as well. The LSA allows tracing of additional signals in the target system. This is useful when monitoring (1) buffers suspected of failure, (2) decode logic, (3) memory management circuit translations, and (4) for asynchronous external events.

DIAGNOSTIC FUNCTIONS

Diagnostics available in the ES1800 Emulator include both RAM/ROM tests and scope loops. RAM test routines verify that RAM is operating properly. They can be run on the target or Emulator overlay memory and may be executed in either byte or word mode. ROM tests include a built-in CRC algorithm.

High speed memory and I/O scope loops for troubleshooting with an oscilloscope are built into the Emulator firmware. They can be used for locating stuck address data, status or control lines, and generating signatures using signature analysis equipment.

The firmware that generates the scope loops is optimized for maximum speed of execution. This short cycle time allows the hardware engineer to review the timing of pertinent signals in the target system without using a storage oscilloscope. The scope loops can be executed in either byte or word mode.

SECTION 2

Table of Contents

Getting Started

Page

GETTING STARTED

Introduction

This section provides a checklist for setting up the Emulator and target system, starting and testing the Emulator and storing customized system variables in EEPROM.

Emulator Setup

1. Refer to page 3-1 and verify that proper grounding and power requirements have been met.
2. Remove the front cover of the Emulator by turning the thumbscrews counterclockwise. The pod and LSA pod may need to be unplugged in order to do this.
3. Verify that the main control board and the memory control board are in the top two slots of the Emulator chassis. (See page 3-4 for board positions.)
4. Verify that the trace/break board is in the third bus slot of the Emulator chassis. (See page 3-4 for board positions.)

5. If you are using overlay memory, verify that the RAM overlay master, and/or master and slave boards, if needed, are inserted. (See page 3-4 for board positions.)

6. Verify that the correct Emulator board for your target microprocessor is in the bottom slot. (See page 3-4 for board positions.)

7. Verify that all boards are firmly seated in their motherboard connectors. (See page 3-4 for board positions.)

8. Set the thumbwheel switch on the main control board for your particular system variables (see page 3-4 for thumbwheel switch location).

   System default variables in switch position 0 are:

- 9600 baud	- 8-bit word length
- One stop bit	- No parity
- Full duplex	- No echo
- Terminal control	- XON and XOFF are recognized
- 8th data bit set to 0 (space)	

9. Verify that the three-position toggle switch on the memory control board is in the center position. See page 3-4 for location illustration.

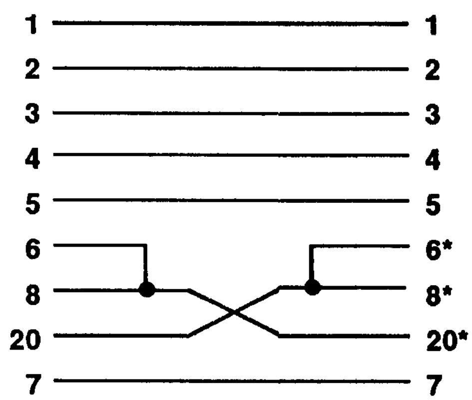
10. Replace front panel and attach the correct pod assembly (see page 3-8). A pod assembly must be connected to the Emulator even if you connecting it to a target system.

11. OPTIONAL: Connect logic state analyzer pod (see page 3-9, LSA).

12. Verify that the RS232C cable connections are correct for the system configuration you plan to use (see page 3-10, Pin Configurations). (See page 2-3, System Configuration.)

13. Verify that the RS232C baud rates and data requirements are set the same on both the Emulator and the terminal. See page 3-5 for thumbwheel switch settings.

14. If using communications without a modem, you may need a null modem cable. If you purchase a null modem cable, it is likely to have the following configuration.



Check the specifications in your terminal manual before reversing the pins.

*Note that pins 6, 8, and 20 are not used, and are unaffected by the cable configuration.

System Configuration

STAND ALONE ENVIRONMENT

Connect the terminal port of the Emulator to a terminal, using a null modem RS232 cable. Insure that the Emulator and terminal are using the same baud rate and data lengths (see page 3-5).

HOSTED ENVIRONMENT

Connect the terminal port of the Emulator to a terminal.

Connect the computer port of the Emulator to the host system. (See page 3-7).

HOSTED SOFTWARE ENVIRONMENT

Initial configuration of the Emulator requires that the Emulator terminal port be connected to a terminal. Insure that the thumbwheel switch and the three-position toggle switch are set to 0 and center, respectively. (See page 3-4.) Power-up the Emulator. Configure the Emulator to meet the host requirements (see **SET** menu, page 5-3). After you have configured the Emulator, type **SAVE;SET 1,1;SAVE**, followed by a [RETURN]. This saves the new configuration. After the [>] prompt is returned (this may take up to four minutes), power-down the Emulator. Set the thumbwheel switch to position 3 (see page 3-4). Connect the computer port of the Emulator to the host system (see page 3-7).

Target System Setup

Check that the target has a lead-less chip carrier (LCC) socket. If this socket is not present, refer to page x for more information.

1. Turn off target system power and Emulator power.
2. Remove the probe tip protector and save for later use. (See page 3-14 for probe tip precautions.) Plug in the probe tip.
3. Check that a good ground exists at the microprocessor socket using an ohmmeter. Measure from pin 9, 35, and 60 to power supply ground on the target board.
4. Turn on target power.
5. Validate power supply at the microprocessor socket in the target. Use a voltmeter and measure from pin 30 and 62 to ground (pins 9, 35, or 60).
6. Check for a valid clock signal at the target microprocessor socket (pin 31).

Power-Up Sequence

TARGET SYSTEM PRESENT

1. Turn on the Emulator.
2. Turn on the target system.
3. Reset the target system (see page 6-29).

NO TARGET SYSTEM

1. Verify that the pod is connected to the Emulator (see page 3-8).
2. Remove the conductive probe tip protector. Save for later use.
3. Power-up the Emulator.
4. The power-up banner should be displayed. Select the internal clock source.

When you power-up the Emulator, all registers, maps, event clauses, and system variables are either cleared or set to default values. Examine the **SET** and **ON** menus (see pages 5-3 and 5-9) and configure the system to your liking. Your special setup can then be stored in EEPROM (see page 5-36). By setting the rotary switch on the controller board to the proper position, your set-up can be autoloaded on power-up (see page 3-4).

The ES1800 Emulator system is now running and ready to accept ES Language commands.

Test Run of System

Use this test guide after the system configuration is correct and the ES prompt is displayed ([>]).

A system test run consists of the following 9 steps:

1. Initialize Emulator.
2. Map overlay memory.
3. Test overlay memory.
4. Enter a program.
5. Verify a program.
6. Run the Emulator.
7. Stop the program.
8. Display the trace buffer.
9. Set a breakpoint.

This test requires an optional overlay memory board. This demonstration does not need a target system.

If you suspect trouble with the ES1800 hardware, call the Applied Microsystems Corporation Customer Service hotline at *1-800-426-3925* for assistance.

1. INITIALIZE THE EMULATOR

Enter the following to initialize the Emulator:

```
>SET 1,0;SAV;SET 1,1;SAV;SET 1,0
```

This will save any changes you have made to the six categories of variables, which include the SET menu (see page 5-3). This operation can take up to four minutes if major changes have been made.

Do not interrupt the operation.

2. MAP OVERLAY MEMORY

Map all of the overlay memory available to the Emulator. Use either of the following procedures:

```
>MAP 0 TO XXXX
```

(Where *XXXX* is the ending address (in hex) of the amount of overlay memory installed.) The following table provides a quick reference for hex values corresponding to overlay memory sizes:

Hex value	Overlay memory
7FFF	32K
0FFFF	64K
1FFFF	128K
3FFFF	256K
7FFFF	512K

For example, to map 64k, enter:

```
>MAP 0 to 0FFFF                    Map memory
>DM                                Display memory
```

A shortcut method is as follows:

```
>MAP 0 TO -1
```

The shortcut will cause an error message, but it maps all overlay memory.

For more information, refer to page 5-66.

3. TEST OVERLAY MEMORY

Test all overlay memory installed by entering:

```
>SF 1,0 TO XXXX
```

(Where *XXXX* is the ending address (in hex) of the amount of overlay memory installed.) If there is a failure, repeat mapping and testing.

For more information, refer to diagnostic functions, page 6-64.

4. ENTER PROGRAM

Enter a short program by invoking the line assembler and entering 80286 op codes (see page 6-44).

```
 >ASM 10
**** 80286 LINE ASSEMBLER VX.XLA ****
CSEG = XXXX
 0010>NOP
 0011>/
 0012>/
 0013>/
 0014>/
 0015>JMP 10
 0017>X
```

NOP is a null operation. Each time you type the slash ([/]), you repeat the previous command, so you have entered the equivalent of five lines of NOPs. The [X] at the end exits the assembler.

5. VERIFY THE PROGRAM

Single step through the program to verify that it works, by entering:

```
>CS = 0
>IP=10
>STP;DT
>/
>/
>/
>/
>/
```

The disassembled trace should show that NOPs were executed and that the jump was taken correctly.

For more information on the **STP** command, refer to page 6-23.

6. RUN THE EMULATOR

Enter **RUN**.

```
>RUN
R>
```

The R> prompt should be displayed with no error messages. This indicates the Emulator is running in real time, executing the program.

7. STOP THE PROGRAM

Enter **STP** to stop.

```
R>STP
```

The Emulator should stop running and display the **CS:IP** register value and group 1. The **CS:IP** value should not exceed 15.

8. DISPLAY THE TRACE BUFFER

Enter **DRT** to display the execution history of the program.

```
>DRT
```

The display should show sequence numbers between 0 and 20, and address values between 10 and 30.

```
>DTB
```

This should show a disassembled trace of the program with NOPs and JMP 10s.

9. SET A BREAKPOINT

Verify that the Event Monitor System halts execution when a defined condition is met by setting a breakpoint. In this case, the Emulator executes 100 (hex) bus cycles, then breaks.

Enter:

```
>WHEN DC1 THEN CNT
>WHEN CL THEN BRK
>DC1=0XXXX
>CL=100    CTL=100
>RBK
R>
```

This causes the counter to be incremented each time data comparator 1 sees a data bus value between 00000 and 0FFFF. When the count limit of 100 is reached, emulation breaks.

If a break does not occur:

1. Enter a **STP**.
2. Set CS and IP to 0 and 10.
3. Enter **DES 1** and verify that you have entered the WHEN/THEN statement and comparator values as shown above.
4. Type **RBK** again.

If no break occurs call Applied Microsystems Customer Service at 1-800-426-3925 for assistance.

SECTION 3

Table of Contents

Hardware

Page

HARDWARE

Emulator Chassis Assembly

The Emulator chassis is the metal enclosure housing the control boards for the target system. This rack-mountable chassis houses up to six boards as shown in the figure on page 3-4.

The Emulator power supply is also in this chassis. A power switch on the rear panel is the only external panel control.

WARNING!

A cooling fan and vent for the Emulator are located on the left side panel of the chassis. The warm air exhaust vent is in the right side panel. Blocking either of these panels may cause the Emulator to overheat.

SYSTEM GROUNDS

The ES1800 Emulator has three grounding systems:

1. A chassis ground from the metallic enclosure of the unit to the power filter.
2. An AC protective ground from the green ground wire of the AC power cord and the chassis ground at the power filter.

3. A signal ground connected by means of a jumper at the power supply terminal strip to the chassis ground. The Emulator has a three-wire power cord with a three-terminal polarized plug. The ground terminal of the plug is connected internally to the metal chassis parts of the Emulator.

WARNING!

Failure to ground the system properly may create a shock hazard.

Emulator Control Boards

Removing the front panel of the Emulator chassis exposes the chassis card cage as shown in Figure 2 on the following page. Open this panel by turning the two knobs in the upper corners of the front panel counterclockwise.

Main Control Board

The main control board holds the controlling 6809 CPU for the Emulator, the EEPROM, two serial ports, and RAM. The 16-position thumbwheel switch on this board determines the system variables and serial line baud rates for autoloading on power-up. Refer to page 3-5 for each switch position setup. Switch position 0 autoloads default system variables.

Memory Control Board

The memory control board holds the memory management logic and optional symbolic memory. The three-position toggle switch below the main control board thumbwheel switch must be in the center position. If the toggle switch is in either of the other two positions, the Emulator will not work properly.

Trace/Break Board

The trace/break board holds trace memory, the Event Monitor System, and the logic state analyzer (LSA) interface.

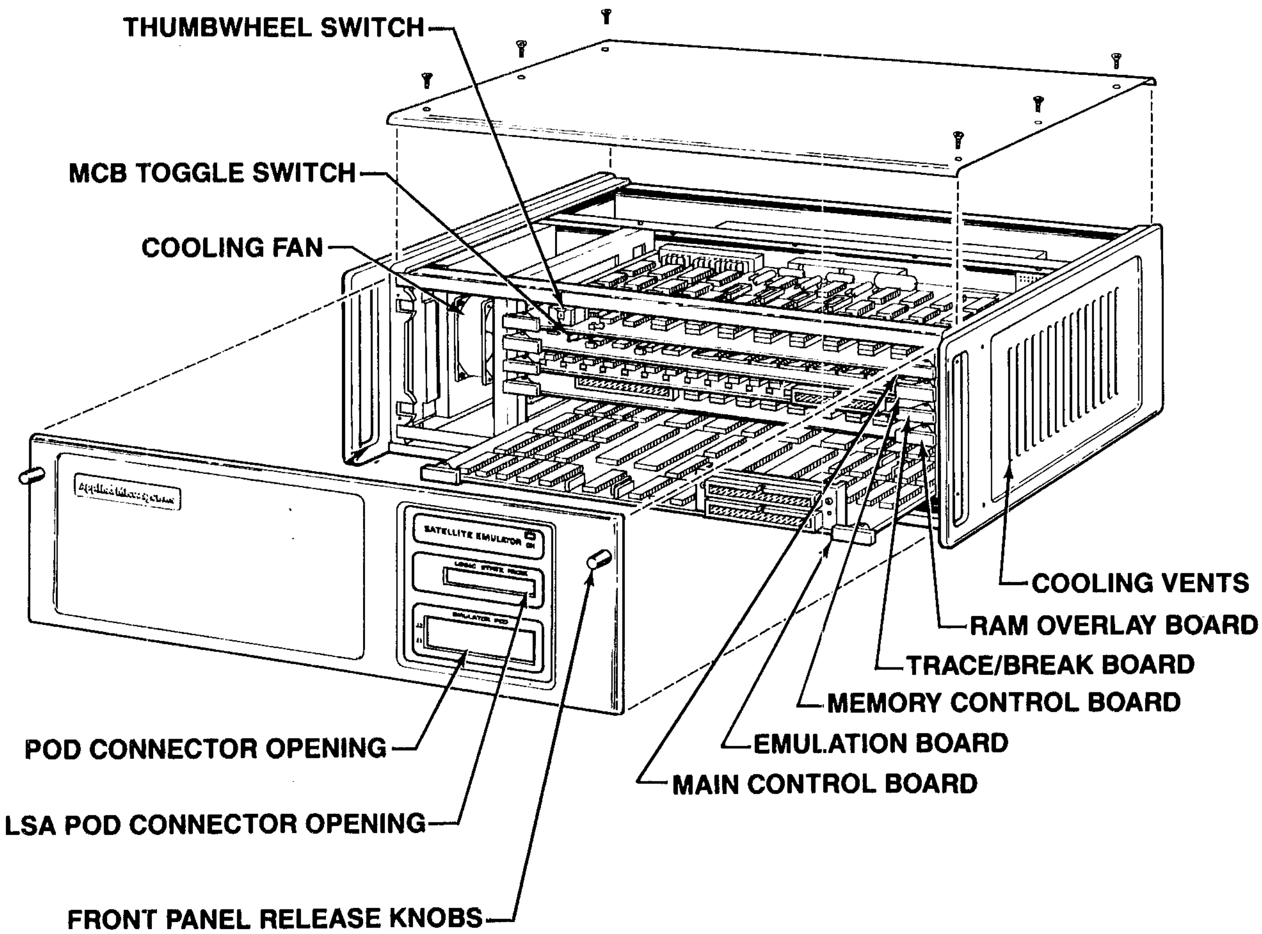
Overlay Memory Board(s)

The overlay memory board set is optional and can hold 32K, 64K, 128K, 256K or 512K of memory. (512K of memory requires a slave board.)

Emulation Board

There are six different emulation boards, depending on the target microprocessor you are using. (Refer to the diagram on the following page for the location of the label indicating the processor type.)

Figure 2. Control Boards



Emulation Board
Thumbwheel Switch Settings

POSITION	PARAMETERS	BAUD RATE
0	Factory Default*	9,600
1	User "0" defined	User defined Terminal control
2	User "1" defined	User defined Terminal control
3	User "0" defined	User defined Computer control
4	User "1" defined	User defined Computer control
5	Factory Default*	110
6	Factory Default*	300
7	Factory Default*	1,200
8	Factory Default*	2,400
9	Factory Default*	4,800
A	Factory Default*	7,200
B	Factory Default*	19,200
C,D,E,F	Reserved for factory use	

----

*Factory Default Parameters

- 8-bit word length
- no parity
- Terminal control
- no echo
- one stop bit
- full duplex
- XON and XOFF are recognized
- baud rate the same for both terminals
- 8th data bit set to 0 (space)

Emulator Chassis Rear Panel

The rear panel of the Emulator mainframe is shown in Figure 3 on page 3-7.

Serial Ports

The two serial ports are RS 232C ports labeled [TERMINAL] and [COMPUTER]. Pins are discussed on page 3-10.

System configuration determines which port your peripheral equipment connects to (see pages 1-4 and 2-4).

Trigger Output

The ES1800 Emulator provides a TTL trigger strobe output controlled by the Event Monitor System. The trigger output is available at a BNC connector on the rear panel of the chassis and on a clip lead attached to the optional logic state analyzer (LSA) pod. Refer to Section 7 for information on Event Monitor System actions.

The trigger can be used for such things as:

- Synchronizing an oscilloscope to the execution of an I/O routine.
- Measuring the duration of a routine by asserting the trigger for its duration and using a timer-counter.
- Cross-coupling two or more Emulators so that an event in one can control events in the others.

Power Switch

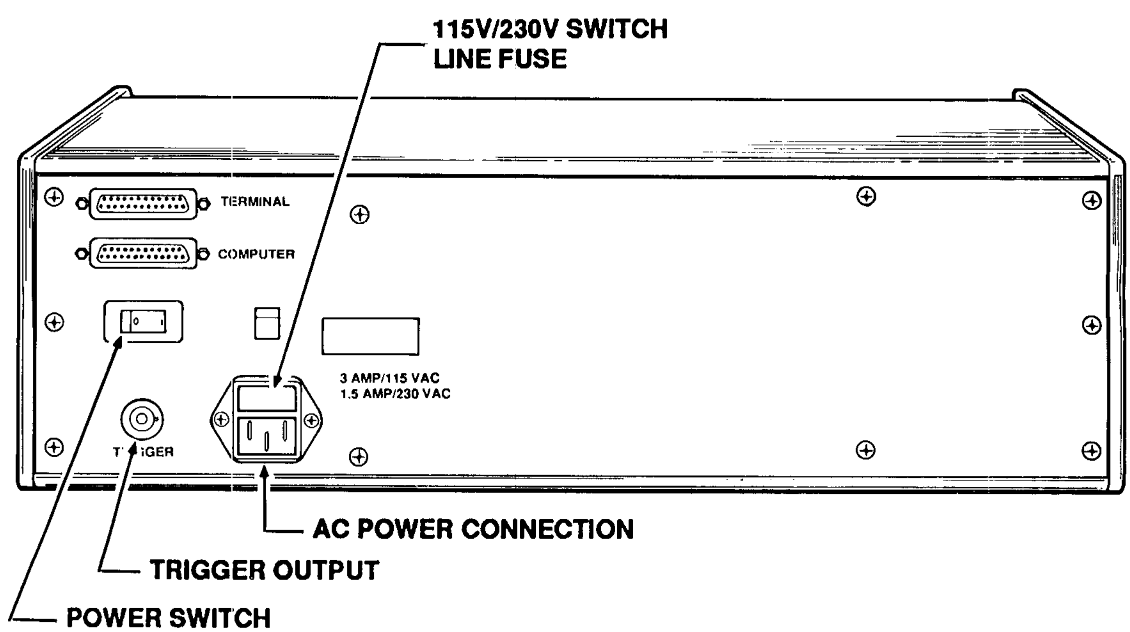
Before powering up, two items should be checked:

1. Proper grounding of power cable (see page 3-1).
2. Proper power-up sequence of Emulator, target system, and/or peripheral equipment. (See Power-Up, page 2-6.)

Line Fuse

A 3 amp slow-blow fuse for 110V operation or a 1.5 amp slow-blow fuse for 220V operations. Remove the fuse by turning the fuse holder counter clockwise.

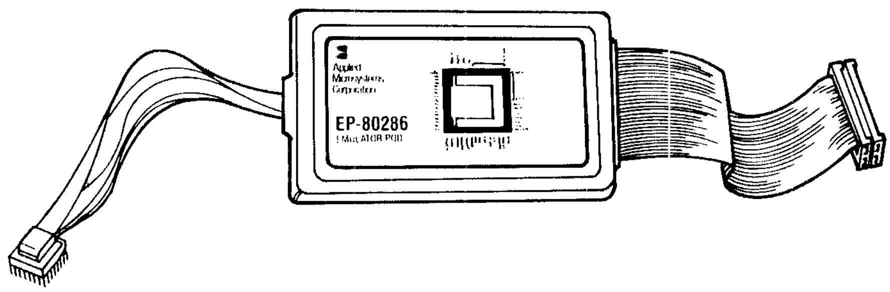
Figure 3. Rear Panel



Pod Assembly

The pod assembly is the link between the ES1800 Emulator and the target system. A 40-inch ribbon cable assembly connects the pod assembly to the Emulator board. An 11-inch ribbon cable assembly ends in a probe tip that is inserted into the microprocessor socket in the target system.

Figure 4. Pod Assembly



With the pod assembly is a set of jumpers (J5 and J6) that lets the CPU send status lines S0~ and S1~ to the target either buffered or unbuffered. J5 and J6 come set to buffered mode for shipping. *Un*buffered, the ES1800 sends S0~ and S1~ three to five ns faster.

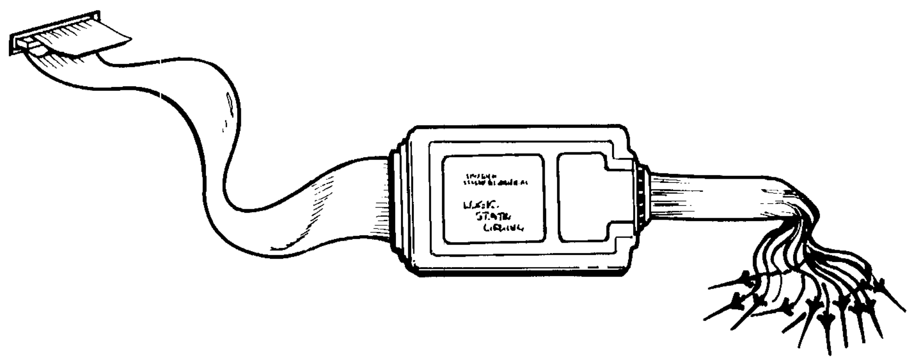
To change the jumpers to unbuffered mode: (1) Disconnect the power. (2) Disconnect the pod probe tip from the target. (3) Remove the four screws from the bottom of the pod case. (4) Remove the bottom PC cover and board. (5) J5 and J6 are on the top board next to the probe tip interface board, and are set to pins 1-2. Set J5 and J6 to pins 2-3. (Pin 3 is the one closest to the 150-pin DIN connector.)

Logic State Analyzer (LSA) Assembly

An optional feature, the logic state analyzer (LSA) pod assembly connects directly above the Emulator pod assembly. The LSA assembly includes a pod, cables, and probe clips. The LSA pod provides 16 input lines and one trigger output line.

The one trigger output line behaves the same as the BNC signal on the rear panel of the Emulator and can be used with an oscilloscope.

Figure 5. Logic State Analyzer Pod Assembly



Serial Ports

Both the terminal port (TERMINAL) and the computer port (COMPUTER) end in standard RS232C female connectors. Make sure peripheral hardware is connected to the correct port.

Baud Rate — Baud rates and data lengths for each port are independent. Refer to the **SET** command (page 5-3) for available baud rates on each port.

Port Control — Only one port can be the controlling port. Either port can give control to the other port. (See pages 5-39, 5-43, and 5-44.)

Upload/Download — The Emulator accepts commands to begin uploading/downloading from either port. However, the Emulator uploads/downloads hex format data files only through the computer port.

Your system configuration determines which port should be in control. Refer to pages 1-4 and 3-5.

PIN CONFIGURATIONS

The pin configuration of your equipment (terminal, PC or host) may not match that of the Emulator. It is important to be familiar with the pin configurations of all peripheral equipment you intend to use with the ES1800 Emulator.

The ES1800 Emulator is configured as "Data Terminal Equipment" (DTE). Before powering up, make sure the ES1800 Emulator system and peripheral hardware are compatible. Pins 1, 2, 3 and 7 must be connected to peripheral hardware. Pins 4 and 5 need to be connected if peripherals attached to the Emulator use these pins. This situation may require a null modem cable with pins 2 and 3 (and pins 4 and 5) crossed.

Both ports use the same pin assignment. All pin assignments and voltage levels conform to Electronics Industries Association RS232C standards. The following chart lists the signals on each pin.

Pin	Name	Description
1	Protective Ground	Connected in the Emulator to logic ground.
2	Serial Data Out	This signal is driven to nominal ±12 voltage levels by an RS232C compatible driver.
3	Serial Data In	Data is accepted on this pin if the voltage levels (±12V) are as specified by RS232C specifications.
4	Request to Send (Output)	This signal is driven to nominal ±12V levels by an RS232C compatible driver. It signals other equipment that the Emulator is ready to accept data at this port.
5	Clear to Send (Input)	An input signal to the Emulator indicates that another piece of equipment in the system is ready to accept data. This signal is terminated so the Emulator operates with the signal disconnected.
6	Not Used	
7	Signal Ground	Connected in the Emulator to the system logic ground.
8-25	Not Used	These pins are not used by the ES1800 Emulator, but may be required by your peripheral hardware.

DATA REQUIREMENTS

Stop Bits

The Emulator software transmits and receives 8-bit ASCII characters. The number of stop bits is determined by **SET** parameter #11 for the [TERMINAL] port and #21 for the [COMPUTER] port (see page 5-3).

Parity

The Emulator sends and checks parity according to system **SET** parameter #12 for the terminal port and #22 for the computer port. These two **SET** parameters are listed on the **SET** menu (page 5-3).

Each character consists of a start bit followed by 8 data bits. When no data is being transmitted, the serial data out pin (pin #2) will be at the 12V level.

Hardware Handshake

When the Emulator is ready to receive data, it asserts the Request To Send line (pin #4). When a receive buffer is nearly full, the Emulator deasserts the Request To Send line.

When the Emulator is ready to transmit data, it checks the status of the Clear To Send line (pin #5). Data is transmitted only when Clear To Send is high.

Software Handshake XON XOFF

The ES1800 uses normal flow control codes to control software handshaking. The default values are XON (CTRL Q) and XOFF (CTRL S). You can change these values (see page 5-3).

The ES1800 serial I/0 system contains internal buffers to smooth the transmission of data via the serial ports. If an input buffer becomes nearly full, the system immediately transmits an XOFF character. When the software empties the input buffer, the system transmits an XON character.

Although the user cannot overfill the input buffer from a controlling terminal, a controlling computer is quite capable of doing so. The input buffer for the computer port is 64 characters deep. When eight characters have been placed in the computer input buffer, the XOFF character is transmitted. Allowing two character times for skew, the computer must transmit no more than 54 characters until the next XON from the ES1800.

The RTS hardware handshake follows the software handshake described above. When an XOFF is transmitted, RTS is dropped on that I/O port; when an XON is transmitted, RTS is reasserted.

Maintenance

Maintenance of the ES1800 Emulator has been minimized by the extensive use of solid-state components throughout the instrument. There are only three areas where you need be concerned.

CABLES

The interconnect cables are the most vulnerable part of the instrument, due to constant flexing during insertion and extraction. First, inspect the cables for any obvious damage, such as cuts, breaks, or tears. Even if you have thoroughly inspected the cables and cannot find any damage, there may be broken wires within the cables (usually located close to the ends). A broken wire within the cable will cause the instrument to run erratically or intermittently if the cables are flexed during emulation (run mode). By swapping the cables in question with a known good set of cables, you can easily isolate the faulty cable.

PROBE TIP ASSEMBLY

The probe tip assembly consists of a ceramic lead-less chip, four ribbon cables and an adapter board. The adapter board is inside the pod case. When the Emulator is not in use, the protective cover should be installed over the ceramic chip to prevent cable abrasion and to protect it from being damaged by other objects. Folding or kinking of the ribbon cables may result in premature failure.

CLEANING THE FAN FILTER

The fan filter should be cleaned regularly. The recommended interval is every 90 days. If you are working in a dusty environment, you may need to clean the filter more frequently.

1. Unplug the ES 1800.

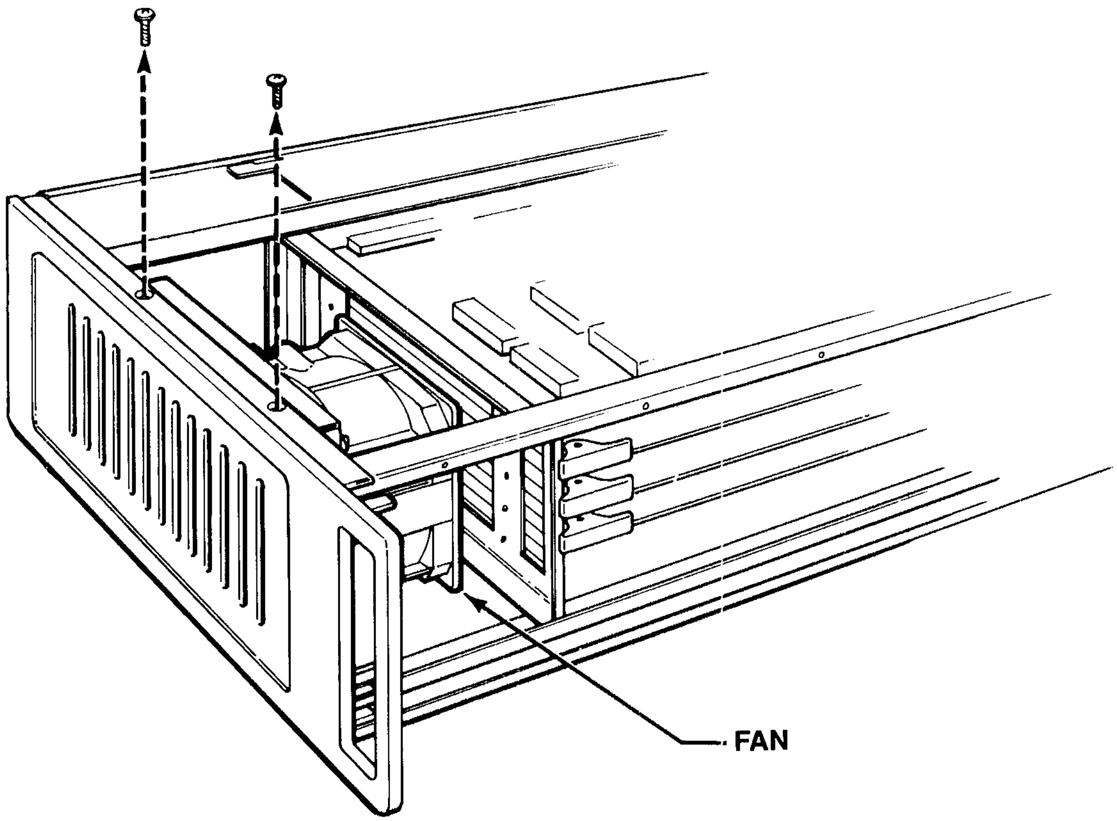
WARNING!

Electrical shock and moving fan parts are dangerous. Make sure you unplug the unit before proceeding.

2. Remove the front cover of the ES 1800. (Loosen the two captive fasteners.)
3. Remove the top cover of the ES 1800. (Unscrew six screws and lift the cover off.)

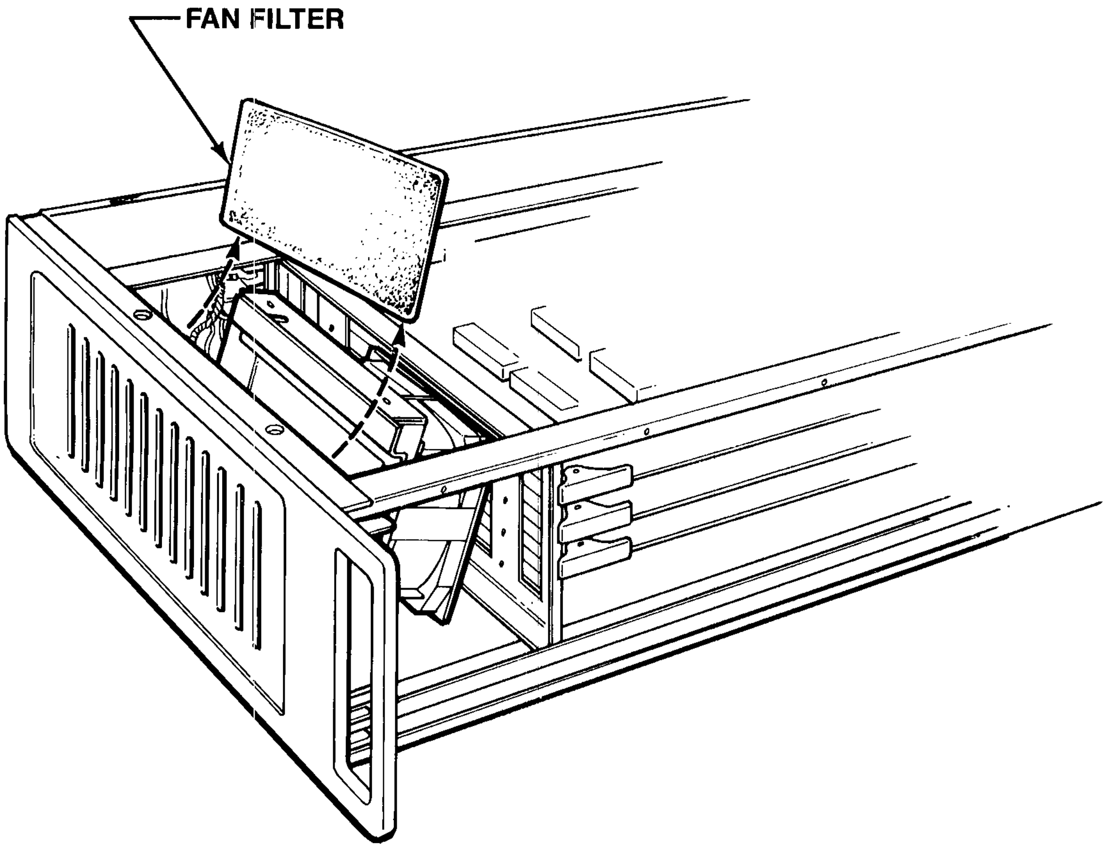
4. Unscrew the two screws at the top of the chassis which hold the fan in place.

ES 1800 Fan Mounting



5. Tilt the fan towards the boards in the chassis.

ES 1800 With Fan Tilted for Easy Access to Filter



6. Remove the fan filter.

7. Rinse the fan filter in cold water. Thoroughly shake out the excess water.

8. Replace the fan filter.

9. Tilt the fan back into the correct position.

10. Replace the screws connecting the top of the chassis to the fan.

11. Replace the top and front covers.

PARTS

The following parts are available for you to order:

```
        Probe tip assembly
        Long cable set
```

Troubleshooting

Check that the interconnect cables are installed properly in a compatible target system, with power applied to both the target system and the Emulator before starting troubleshooting procedures.

The most common problems encountered are listed below. We recommend that you contact Customer Service at Applied Microsystems Corporation if you experience any problems that do not fall within this range of items. Before you call our service department, display your software revision number by typing **REV** (page 5-126). Also, record the serial number on the back of the chassis. You will be asked for the revision number and serial number when you call.

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

Troubleshooting

SYMPTOM	POSSIBLE CAUSES
Target system runs erratically	1. Faulty interconnect cables
	2. Broken ceramic probe tip
	3. Poor seating of probe tip assembly
	4. LDV not executed before RUN (vector not loaded).
Emulator will not communicate over RS232	1. Baud rate set incorrectly.
	2. Target system requires "null" modem cable (pin 2 and pin 3 of RS232 connector reversed).
	3. Emulator configuration incorrect.

ES 1800 Emulator Specifications

INPUT POWER

Standard	90 to 130 VAC 47 to 60 Hz consumption less than 130W
Optional	180 to 260 VAC 47 to 50 Hz consumption less than 130W

ENVIRONMENTAL

Operating Temperature	0 °C to 40 °C (32 °F to 104 °F)
Storage Temperature	-40 °C to 70 °C (-40 °F to 158 °F)
Humidity	5% to 95% relative humidity, non-condensing

PHYSICAL

Mainframe	13.2 cm x 43.18 cm. x 34.29 cm. (6.2 in. x 17 in. x 13.5 in)
Emulator Pod	22.6 cm. x 12.9 cm. x 4.1 cm. (8.9 in. x 5.1 in. x 1.6 in.)

Target System Connection (total length including pod)	1.5 m (60 inches)
LSA Pod	12.4 cm. x 7.9 cm. x 2.3 cm. (4.9 inc. x 3.1 in. x .9 in.)
Total Weight	9.1 kg. (20 lbs.)
Shipping	10.9 kg. (24 lbs.)

SECTION 4

Table of Contents

ES Language

Page

Page

ES LANGUAGE

Structure of the ES Language

The command language that controls the ES1800 Emulator is a formal language. Once you understand the basic concepts of this language, you can apply the full debugging power of the Emulator. An overview of the ES language structure is presented in the accompanying table. A more detailed description of the language elements, the help menu, prompts, special operating modes, and ES language error messages are also included in this section. Items in angle brackets (<>) are mandatory and must be entered as part of the command.

Items shown in square brackets ([]) are optional. Do not type the angle or square brackets when typing a command.

If the ESL command interpreter detects an illegal statement, it beeps and places a question mark under the command line near the position the error was detected. Enter a ? following an error to display the appropriate error message.

ES Language Syntax

Language Element	Example
Command Line	
[Repeat] Command Statement [;Command Statement] ... <RETURN>	
Single Character Instant Command	
Repeat	
<*>	*STP;DT
<*><Repeat limit>	*9 STP;DT
Repeat Limit:	
Decimal number only (1 to 2^{32}-1)	87651234
Command Statement	
Command Mnemonic	DTB
Command Mnemonic <Expression>	MM CS:IP + 4
Command Mnemonic <Expression List>	SET #20,#14
Assignment Command	CS = 0FA9
Expression	2 * GR5
Event Monitor System Control Statement	WHEN AC1 THEN BRK
Single Character Instant Command	
</> (repeat previous command line)	
<,> (execute macro 1 or decrement scroll in memory mode)	
<.> (execute macro 2 or increment scroll in memory mode)	
<?> (help)	
Command Mnemonic	
<1 or more alpha characters>[1 or more decimal characters]	ASM
Expression	
[Unary Operator] Ivalue	-2473
Ivalue <Operator> Expression	2 - 3F6C90
<@> Expression	@240;@@@SS:SP
<(> Expression <)>	2 * (-2 + 3)
Nvalue <:> Nvalue	CS:1234

ES Language Syntax *(cont)*

Language Element	Example
Ivalue:	
Symbol	`'main`
Nvalue	
Symbol:	
<'><1 or more printable characters><space or return>	
Nvalue:	
Number	`7FA36`
Register Name	`IP`
Register Name:	
<1 - 3 Alpha characters>[0 - 2 decimal digits]	
Number:	
[Base]<1 or more digits>	`%0101001`
Base:	
<%> (binary)	
<\> (octal)	
<#> (decimal)	
<$> (hexadecimal)	
Expression List	
Expression <,> Expression [,Expression list]...	`1,CS:IP,2+2,-6`
Assignment Command	
Svalue <=> Expression	`IP = @0FFFF0`
<@> Expression <=> Expression	`@SS:SP = CS:IP`
Svalue:	
Symbol	`'Test_result`
Register Name	`MMP`

ES Language Syntax *(cont)*

Language Element	Example
Event Monitor System Control Statement	
[Group] <WHE[N]> Event <THE[N]> Action List	WHEN AC1 THEN BRK
Group:	
<1>	
<2>	2 WHE AC1 THE BRK
<3>	
<4>	
Event:	
[Disjunctive] <Event Comparator>	NOT AC1
Event <Conjunctive> <Event>	DC2 OR NOT AC1
Disjunctive:	
<NOT>	
Event Comparator	
<AC1>[.Group]	AC1.3
<AC2>[.Group]	
<DC1>[.Group]	
<DC2>[.Group]	
<S1>[.Group]	
<S2>[.Group]	
<CTL>[.Group]	CTL.4
<LSA>[.Group]	
Conjunctive:	
<AND>	
<OR>	
Action List	
<Action>[,Action]...	TRC,TGR,FSI

ES Language Syntax *(cont)*

Language Element	Example
Action:	
<BRK>	
<TRC>	
<TOT>	
<CNT>	
<TOC>	
<RCT>	
<TGR>	
<FSI>	
<GRO Group>	GRO 3
Unary Operator	
<ABS>	ABS GD3
<!>	!0AA
<->	-3
Operator	
Mul.op	
Add.op	
Shft.op	
<&>	GD4 & 0FF
<^>	DC2.3 ^ 0FF00
Mul.Op	
<*>	2 * 3
</>	0FAC / %01001
<MOD>	GD5 MOD 7
Add.op	
<+>	GRO + IP
<->	@(SS:SP - 4)
Shft.op	
<<<>	DC1 << 3
<>>>	

Notes on ESL

Command Line

A command line is created by entering one or more characters after the ESL prompt (see page 4-24 for a description of the various prompts). One or more command statements can be placed on a single command line. Multiple command statements must be separated by a semicolon. The command line is limited to 76 characters and must be terminated with a return. The only way to extend command lines is by using macros (see page 5-105).

Backspace or delete characters may be used to delete the previous character entered on a command line. CTRL X deletes the entire line. CTRL R redisplays the current line (useful for hardcopy terminals).

Repeat

If an asterisk (*) is the first character on the command line, the entire command line will be repeated indefinitely. If the asterisk is followed immediately by a decimal number, the command will be executed that many times. A repeating command line may also be terminated by setting the TST register to zero within the command line. This allows you to repeat something until a condition is met.

Command Statement

There are several special modes in which the above command statement rules do not apply. In memory mode (see page 6-52) entering a RETURN on an empty line causes the next location to be read. Entering a value followed by RETURN will cause that value to be written to memory. I/O mode (page 6-52), the line assembler (see page

6-44), memory disassembler (page 6-50), and main help menu (page 4-19) all have special modes that alter the normal execution of ESL commands.

Single Character Instant Commands

These commands are processed immediately when they are the first character entered on a command line. The forward slash character ([/]) causes the previously entered command line to be repeated.

```
>STP
>/
>/
```

This command single steps three times.

The comma ([,]) executes macro 1, and the period ([.]) executes macro 2. However, if you are in memory or I/O mode, the period moves you to the next higher memory address, while the comma moves you to the next lower address.

The question mark ([?]) also has two uses. It can be entered after the command interpreter detects an error and beeps. If you are "beeped," enter a [?] and the command processor will give you an error message describing the problem it detected.

A [?] entered at any other time (i.e., not after an error), causes a two-page Help Menu to be displayed. A [RETURN] moves you from the first page to the second. Any other character terminates the Help Menu.

Command Mnemonics

Command mnemonics are the alpha-numeric character strings that identify a specific ESL command. Command mnemonics are formed from 1 to 3 alpha characters followed by 0 to 2 numeric characters. Extra characters in between are ignored. For example, WHEN is the same as WHE and GR12345 is the same as GR45. See the Appendices for a list of all ES language mnemonics.

Expression

An expression can be an integer value, an alpha/numeric value, or an equation.

Parentheses may be used to alter the normal precedence of operations. The Emulator recognizes parentheses just as they are treated in algebraic equations. You can use as many levels of parentheses as you need. The only limitation is that statements can be no more than 76 characters long.

Parentheses are not allowed in WHEN/THEN clauses. Parentheses are also not allowed in conjunction with the colon operator (when used as an address expression operator).

The expression processor can resolve arbitrarily complex expressions.

```
@(GD0 +3) = IP + #100 * (DX >> 4) +0AF34
```

This example retrieves the value of the DX register, shifts it right 4 bit positions (divide by 2^4), multiplies the result by 100 decimal, adds 0AF34 and the contents of the IP register, and writes the result to the location 3 bytes above the address in GD0.

A more common and useful example might be:

```
ASM CS:IP
```

This computes the address CS:IP and starts up the line assembler at that address. The expression:

```
'Interrupt + 1A6
```

by itself will add 1A6 to the current value of the symbol, Interrupt, and display the result. If you do not assign the results of an expression to a location or register, the result is displayed as a 32-bit value.

The `@` operator is an indirection operator. `@ Exp` (where `Exp` is an expression) refers to the value in memory at the address `Exp`. If the `@ Exp` is on the left side of an `=` then the value from the right side of the `=` will be loaded into memory at the address `Exp`. At all other times, `@ Exp` simply reads a value from memory. `@SS:SP` is a simple way to read something from the stack pointer. In physical mode (see page 6-4) it is legal to have multiple indirections, e.g., `@@GRO = @@@(SS:SP + 6)`. Byte mode and word mode affect the length of data transferred to or from the target by the `@` operator.

In physical mode (see page 6-4) the colon operator (`:`) mimics the arithmetic combination of segment and pointer registers in the 8086 microprocessor. The value on the left side of the colon is shifted left 4 bits, added to the value on the right side and, finally, the total is masked to 20 bits. In virtual mode the expression

```
16 bit value:16 bit value
```

is interpreted as

```
selector:offset
```

The colon operator is handled at the preprocessor level and thus has higher precedence than normal math operators. The colon operator must be used only between actual numbers or register names; e.g., `CS:IP` is fine but `CS:(IP+3)` is illegal.

All other math or logic operations are evaluated according to the order given in the following section on operators. Parentheses may be used to alter the normal precedence. Unary operations must be enclosed in parentheses if they occur within another expression; e.g., `2+-1` is illegal, but `2+(-1)` and `-1+2` are legal.

Certain combinations of expression types and operators are illegal or have complex results. See Results of Dyadic Operator Combinations table on page 4-19.

Some commands can accept a variety of argument types. The display block (**DB**) command accepts an integer, a range, or no argument at all. Other commands require that a certain argument type be used. The upload (**UPL**) command requires a range argument. See the discussion on Numbers, below, for types.

Symbols

If the symbolic debug option is installed in the ES1800 Emulator, you can use symbolic references. Every symbol must begin with a single quote ('). Symbols are composed of 1 to 64 printable characters followed by a space or RETURN. Symbols can be used anywhere a register or a number is used, with the exceptions that symbols are not valid with the colon (:) operator or the repeat (*) operator.

Numbers

The ES1800 has a default base register. It is assumed that numbers entered without a leading base character are being entered in the default base. Generally, the default base is hexadecimal (factory default). See page 5-83 for more information on changing the default base register.

There are three different types of numbers.

- An integer is a 32-bit signed value.
- A don't care is a 32-bit value with a 32-bit mask. For each bit set in the mask, the corresponding bit position in the value is ignored during Event Monitor comparisons. Don't cares can be entered in two ways. 1234 DC 0FF0 is explicit. 1XX4 is equivalent to 1FF4 DC 0FF0. Don't cares are useful for setting the Event Monitor System event comparators (see page 7-2).
- A range is specified by entering a start address and

a length or an endpoint. `200 LEN 20` is the same as `200 TO 21F`. Ranges can be either internal (default) or external. An explicit range type can be specified by using the prefix IRA or XRA. `0 LEN 100` is the same as `IRA 0 LEN 100`. The `!` operator inverts the type of a range value. `!(0 LEN 100)` is the same as `XRA 0 LEN 100`, which means everything but addresses 1 to 00FF. The endpoints are always included in the range. Regardless of the method of entering (TO, LEN), range values are always displayed as "start TO end."

Ranges, don't cares, and integers are not generally interchangeable. Certain registers can only hold certain data types. All registers can hold integers. Address type registers cannot be loaded with don't care values. Status and data registers cannot be loaded with range values. See page 5-75 for a list of all registers and their data types.

Base

To enter a character in any base other than the default, use a leading base character: `%` = binary, `\` = octal, `#` = decimal, and `$` = hexadecimal.

Expression List

Lists are required by a few commands. They can also be used for implicit evaluation. For example, in pause mode, entering the three numbers `%010011010, #128, \77347` causes the Emulator to display their equivalent in the default display base (usually hexadecimal). Lists are limited to nine elements. Lists are used in memory and I/O modes as well.

Indirection Operator

In physical mode, the form `@Expression = Expression` will cause the left side expression to be calculated and used as an address at which to store the value of the right side expression. Note that since `@Expression` is itself an expression, in physical mode, commands such as `@@SS:SP = 0` are legal and useful.

Parentheses may be used to affect the processing of the `@` operator:

```
>@ GD4 + 6
>@ (GD4 + 6)
```

In the first example the indirection operator is applied to `GD4`. The command interpreter accesses the target system location pointed at by `GD4`, adds six to the value stored there, and displays the final results.

In the second example, the Emulator displays the value stored in the sixth location above the address pointed to by `GD4`.

Memory mode always executes memory reads. This may be unacceptable for certain hardware configurations. To store values without entering memory mode, use:

```
>@ <address> = <data>
```

This causes the system to load data into the specified address.

In physical mode, more than one @ operator in an expression displays a quantity pointed to by another quantity located in the target system memory. The Emulator evaluates the expression following the @ operators, considers it an address, and looks at the value stored at this address. The value at this address is also considered to be an address. This address is accessed and displayed.

Registers

Registers are grouped into three types: integer only, don't care, and range. Any register can be assigned an integer value. Don't care registers can be loaded with don't care values or integers but not ranges. Range registers can be loaded with integers or ranges but not don't care values. See page 5-75 for a list of all registers and their data types.

Event Monitor System Control Statement

Event Monitor System statements describe combinations of target program conditions and the corresponding actions to be taken if the conditions are met; they do not describe mathematical or logical computations. Be aware that normal expression operators are illegal when specifying Event Monitor System statements. These statements are discussed in detail in Section 7, Event Monitor System.

Group

The Event Monitor System (EMS) is arranged in four independent groups. These groups provide a state-machine capability for debugging difficult problems. An EMS control statement can only be associated with one of the four groups. If no group numbers are mentioned in the EMS control statement, the statement is assigned to group 1. There are two ways to override this default selection of group 1. You can begin the EMS control statement with a group number, or you can append a group number to any one of the vent comparator names. For example: `3 WHEN AC1 THEN BRK` is functionally the same as `WHEN AC1.3 THEN BRK`; both use group 3. You cannot mix group numbers within a single EMS control statement.

Event

You can define an event to be some combination of address, data, status, count and logic state probe conditions. Numerous Event Monitor System control statements can be entered and will be in effect simultaneously. Conflicting statements may cause unpredictable action processing. Parentheses are not allowed in event specifications.

Disjunctive

The NOT operator is used to reverse the sense of the comparator output. NOT has higher precedence than either of the conjunctives, AND and OR.

```
WHEN AC1 AND NOT DC1 THEN BRK
```

This statement means break whenever any data pattern other than that in DC1 is read from or written to the address in AC1.

Conjunctive

AND and OR can be used where needed to form more restrictive event definitions. AND terms have higher precedence than OR terms.

```
AC1 AND DC1 OR DC2
```

This event is equivalent to `AC1 AND DC1` in one statement and `DC2` in another. If you are looking for two different data values at an address, use:

```
AC1 AND DC1 OR AC1 AND DC2
```

The OR operator is evaluated left to right and is useful for simple comparator combinations. For complex event specifications, OR combinations can be replaced with separate EMS control statements for clarity.

```
AC1 AND S1 OR AC2 AND S2
```

This event is the same as `AC1 AND S1` and `AC2 AND S2` in separate statements.

Unary Operator

All internal computations use 32-bit math. Values entered with a leading `-` are converted to signed numbers; e.g., `-1` is stored internally as `$FFFFFFFF`. Internal math, however, is signed only for the `+`, `-`, `*`, `/` operations; `-5+3` is `$FFFFFFFE`, while `-1 >> 1` is reduced to `$7FFFFFFF`.

ABS converts a signed number to its absolute value.

[!] is a logical NOT operator and compliments all 32 bits of a number. If the number is a range, the range type (internal or external) is inverted.

Unary operators have the highest precedence. [-2+3] is 1.

Operator

The operators are listed below in descending order of precedence. Operators of the same type are evaluated left to right.

```
Mul.op:
      *         Multiply
      /         Divide
      MOD       Modulo

Add.op:
      +         Add
      -         Subtract

Shft.op:
      >>        Right shift
      <<        Left shift

&               Logical AND

^               Logical OR
```

Modulo

The MOD operator. The result of this operation is the remainder after the value on the left has been divided by the value on the right.

```
>29 MOD 4
result = 1
>38 MOD 6
result = 2
```

Results of Single-Argument Operators

Operator	Argument	Result
!	Integer	Valid
	DC	Don't care bits are not affected.
	IRA	Complement (IRA becomes XRA)
ABS	Integer	Valid
	DC	Don't care bits are not affected.
	IRA	Invalid
	XRA	Invalid
	Integer	Valid
	DC	Don't care bits are not affected.
	IRA	Invalid
	XRA	Invalid
@	Integer	Valid
	DC	Invalid
	IRA	Invalid
	XRA	Invalid

Results of Dyadic Operator Combinations

Left Hand Expression	Right Hand Expression	Operator	Result
Integer	Integer	* / MOD	Valid
		& ^	Valid
		<< >>	Valid
		+ -	Valid
Integer	Don't care	MOD	Illegal
		* /	Don't care bits are passed to the left hand argument.
		& ^	Don't care bits are passed to the left hand argument.
		<< >>	Don't care bits are passed to the left hand argument.
Integer	IRA XRA	* / MOD	Invalid
		& ^	Invalid
		<< >>	Invalid
		+ -	The endpoints of the range will be altered by the value of the integer expression.
Don't care	Don't care	* / MOD	Invalid
		& ^	Invalid
		<< >>	Invalid
		+ -	Don't care bits are ANDed.
Don't care	Integer	* / MOD	Don't care bits are kept.
		& ^	Valid
		<< >>	Don't care bit positions are shifted.
		+ -	Don't care bits are kept.
IRA, XRA	Integer	* / MOD	Invalid
		& ^	Invalid
		<< >>	Invalid
		+ -	The end points of the range will be altered by the value of the integer expressed.

Help

There are two pages of help information available. Enter a ? as the first character of a command line to display the first help page, or type HELP. This page gives examples of the most commonly used commands and their meanings. The second page describes the Event Monitor System registers and commands. Enter a RETURN at the end of the first page to move to the second page. The menus are shown on the next two pages.

Information on switch settings, configuration settings, and special functions is available without using the help menus. Other help menus are described below.

Software Switches	Enter either ON or OFF to display the current settings and definitions of all software switches, (see page 5-9).
Communications Set-up	Enter SET to display the current configuration settings and possible values (see page 5-3).
Special Diagnostic Functions	Enter SF to display a list of the available special functions (RAM/ROM tests, Scope loops, etc.) (see page 6-67).

First Page of Help Menu

```
>?
RUN/EMULATION:                      RUN/RNV - RUN/RUN WITH NEW VECTORS
 STP - SINGLE STEP/STOP             RBK/RBV - RUN TO BREAKPOINT/WITH VECTORS
 RST - RESET TARGET PROCESSOR       WAIT - WAIT UNTIL EMULATION BREAK

TRACE HISTORY:                      DTB/DTF-DISASSEMBLE PAGE BACK/FORWARD
 DT - DISASSEMBLE MOST RECENT LINE  DRT (X)-DISPLAY PAGE RAW TRACE (FROM X)

MEMORY - REGISTER COMMANDS:         DR/DRL-DISPLAY VIR/PHY MODE CPU REGS
 DB X TO Y - DISPLAY BLOCK          FILL X TO Y,Z - FILL BLOCK WITH Z
 BMO X TO Y,Z - BLOCK MOVE TO Z     LOV/VFO X TO Y - LOAD/VERIFY OVERLAY
 DRL - DISPLAY REGISTERS IN LONG WORD (VIRTUAL MODE)
 X - EXIT MEMORY MODE               M X - VIEW/CHANGE MEMORY AT X

MEMORY MAPPING:
 MAP X TO Y :RO  :RW  :TGT  :ILG    DM/CLM - DISPLAY/CLEAR MEMORY MAP

COMMUNICATIONS:                     TRA - TRANSPARENT MODE TERMINAL-HOST
 DNL - DOWNLOAD HEX FILE FROM HOST  CCT - TRANSFER CONTROL TO COMPUTER PORT
 UPL X TO Y - UPLOAD HEX TO HOST    TCT - TRANSFER CONTROL TO TERMINAL PORT

SYSTEM:                             SET - VIEW/ALTER SYSTEM PARAMETERS
 ON/OFF - VIEW/ALTER SWITCHES       SF - VIEW/EXECUTE SPECIAL FUNCTIONS
 ASM (X) - IN LINE ASSEMBLER        DIS(X) DISASSEMBLE FROM MEMORY
 LD/SAV (X) - LOAD/SAVE 0=SETUP,1-REGS,2-EVENTS,3=MAP,4=SWITCHES,5=MACROS
```

Second Page of Help Menu

```
EVENT MONITOR SYSTEM
DES       -       DISPLAY ALL EVENT SPECIFICATIONS
CES       -       CLEAR ALL EVENT SPECIFICATIONS
DES X     -       DISPLAY ALL EVENT SPECIFICATIONS FOR GROUP X
CES X     -       CLEAR ALL EVENT SPECIFICATIONS FOR GROUP X

EVENT ACTIONS:
BRK - BREAK          CNT - COUNT EVENT     TGR     - TTL TRIGGER STROBE
TRC - TRACE EVENT    RCT - RESET COUNTER   FSI     - FORCE SPECIAL INTERRUPT
TOT - TOGGLE TRACE   TOC - TOGGLE COUNT    GROUP X - SWITCH TO GROUP X

EVENT DETECTORS - GROUPS 1,2,3,4:
AC1,AC2 OR AC1.X,AC2.X  - 24 BIT DISCREET ADDRESS OR INTERNAL EXTERNAL RANGE
DC1,DC2 OR DC1.X,DC2.X  - 16 BIT DATA, MAY INCLUDE DON'T CARE BITS
S1,S2 OR S1.X,S2.X      - STATUS AND CONTROL - BYT/WRD + RD/WR + TAR/OVL
                        - + MEM/IO/IAK/RIO/HSD/IF/RM/WM + QF/NMI/LOK/PEA
                        - + ALT/COD/DAT/STA
LSA       -       16 LOGIC STATE LINES, MAY INCLUDE DON'T CARE BITS
CTL       -       COUNT LIMIT, ANY NUMBER 1 TO 65,535

STEP 1 - ASSIGN EVENT DETECTORS           STEP 2 - CREATE EVENT SPECIFICATIONS
AC1 = $1234;S1 = BYT + RM                 WHEN AC1 AND S1 THEN GROUP 2
AC1.2 = $4576+14*6;DC2.2 = $5600 DC $FF   2 WHEN AC1 AND NOT DC2 THEN CNT
CTL.2 - 24;AC2.2 = $F000 LEN $400         WHEN CTL.2 OR AC2.2 THEN BRK
```

Power-Up Banner

On power-up, the following banner is displayed:

```
COPYRIGHT 1986
APPLIED MICROSYSTEMS CORPORATION
SATELLITE EMULATOR 80286  V1.0
USER = 0, SW = 0
#64K AVAILABLE OVERLAY
```

Prompts

Different prompts are displayed depending on the current operating mode of the ES1800.

>	The standard, or pause mode, prompt from ESL consists of a space character followed by a right arrow.
R>	During emulation, the run mode prompt is displayed. Most ESL commands are still valid.
$12345678 $00 > *$12345678 $00 R>* *$12345678 $0000 >* *$12345678 $0000 R>*	In memory mode, the prompt includes the memory address and the data contained there. Depending on whether byte mode or word mode (**BYM**, **WDM**) has been chosen, the data will be a byte or a word. The run prompt may also be present during memory mode.
***** 80286 LINE ASSEMBLER VX.XLA ***** *CSEG = 0000* *0100 >*	The line assembler displays a 16-bit address prompt. This prompt contains an [R] if you are assembling during emulation.
IO:$1200 > *IO:$1200 $00 >* *IO:$1200 $0000 >* *IO:$1200 R>* *IO:$1200 $00 R>* *IO:$1200 $0000 R>*	In I/O mode, the prompt includes the I/O address. The data is included when a [RETURN] is entered as the only character on the line. The data field is affected by byte and word mode. When emulating, the run prompt will also be present.

Special Modes

There are a few special modes you can enter, some of which must be exited before using regular ESL commands. These modes can be identified by the prompt displayed, or lack thereof.

Line Assembler

The line assembler has a single 16-bit address prompt. Exit by entering an [X] or the **END** directive.

Memory Disassembler

If initiated without a range argument, the memory disassembler (**DIS**) displays a full page of data, leaving the cursor at the lower right corner of the screen. A [RETURN] displays the next page of disassembled memory. A [SPACE] causes only the next instruction to be disassembled. Any other character terminates memory disassembly.

Memory Mode

Memory mode has an address and data prompt. Exit by entering an [X].

I/O Mode

I/O mode has an address prompt. Exit by entering an [X].

Transparent Mode

No characters are generated by the ES1800. Exit by entering the two- character escape sequence (default is [ESC] [ESC]), or system reset (default [CTRL] [Z]).

Display/Interpretation Modes

Byte Mode/ Word Mode

The **BYM** and **WDM** commands select byte or word mode operation. The mode selected determines whether 8 or 16 bit data is used or displayed. If byte mode is set, most data commands use byte values, and the indirection operator reads a byte from the address given. The same is true of word mode.

Physical Mode/ Virtual Mode

Two commands are used to globally designate the mode (**PHY** for physical mode, **VIR** for virtual mode) in which addresses are entered and displayed. (See Virtual Operations, page 6-2, for more information.)

Display Mode

The **DMD** command displays the data mode, emulator mode and processor (CPU) mode. See the examples below and on the following pages for different combinations. The state transition diagram (page 6-7) and mode translation table (page 6-5) provide information on the possible states.

```
>DMD
DATA MODE:       BYTE
EMULATOR MODE:   VIRTUAL
PROCESSOR MODE:  REAL
```

```
>WDM
>DMD
DATA MODE:      WORD
EMULATOR MODE:  VIRTUAL
PROCESSOR MODE: REAL
```

```
>PHY
>DMD
DATA MODE:      WORD
EMULATOR MODE:  PHYSICAL
PROCESSOR MODE: REAL
```

```
>PHY
>DMD
DATA MODE:      BYTE
EMULATOR MODE:  PHYSICAL
PROCESSOR MODE: REAL
```

```
>MSW=1
>DMD
DATA MODE:      BYTE
EMULATOR MODE:  PHYSICAL
PROCESSOR MODE: PROTECTED
```

```
>VIR
>DMD
DATA MODE:      BYTE
EMULATOR MODE:  VIRTUAL
PROCESSOR MODE: PROTECTED
```

```
>RUN
R>
R>DMD
DATA MODE:       BYTE
EMULATOR MODE:   VIRTUAL
PROCESSOR MODE:  NOT AVAILABLE DURING
                 EMULATION
```

```
R>STP
 CS:IP = $F000:FFF0
GROUP 1
>DMD
DATA MODE:       BYTE
EMULATOR MODE:   VIRTUAL
PROCESSOR MODE:  PROTECTED
```

Special Characters

DELETE *BACKSPACE*	Either character deletes a character just entered on a command line.
CTRL X	Deletes an entire command line.
CTRL R	Redisplays the current command line.
CTRL Z	The default system reset character. Resets the Emulator, stops emulation and/or clears an error condition. It does not clear or update Emulator registers. Used to terminated certain diagnostic functions. CTRL Z terminates an indefinitely repeating command. You can change the reset character (see page 5-3).
ESC ESC	The default transparent mode escape sequence, which terminates transparent mode. You can change the transparent mode escape sequence (see page 5-3).
CTRL S	The default XOFF character. When issued from the keyboard, the screen display stops scrolling, allowing you to see the information. You can change the XOFF character (see page 5-3).
CTRL Q	The default XON character. Restarts the screen display scrolling after an XOFF is issued. You can change the XON character (see page 5-3).

Errors

The ES1800 software generates two basic types of error messages. ES Language Syntax and operational errors in a command line are indicated by a beep (BEL code). The next line displayed contains a single [?] underneath, and usually just after, the place in your command line that caused the error. At the point the error is detected, the remainder of the command line is discarded. For example, the **DRT** command is invalid during emulation:

```
 >WHE AC1 THE BRK; RBK; DRT; DR
 <BEL>                      ?
R>
```

The **RBK** command was executed, but the **DR** command was not. Whenever you see an error message of this type, you can enter a single [?]. The ES1800 responds with a text message explaining the error. For the above example:

```
R>?
ERROR #56
TRACE DATA IS INVALID DURING EMULATION
R>
```

These error messages are described in this section. The second type of error message is caused by target hardware problems. There are various conditions that can occur in the target that prevent the pod processor from operating. If these error messages are displayed, the problem must be remedied before the ES1800 can be used. The error messages are quite explicit: e.g., [NO TARGET CLOCK] or [RESET ASSERTED]. Target hardware error messages are explained in Appendix B.

ES Language Error Messages

1,2,3 EXPRESSION HAS NO MEANINGFUL RELATION TO REST OF COMMAND. Often caused by entering symbols out of context. [DR] and [BRK] are both legal, but when entered together as [DR BRK] would cause this error message.

5 UNDEFINED SYMBOL OR INVALID CHARACTER DETECTED. Usually caused by improper spelling.

6 CHECKSUM ERROR IN DOWNLOAD DATA. The last record received was in error. Make sure that the format selected in the system set-up is the same as the format of the received data. Refer to download command (**DNL**) for error handling.

7 BAD STATUS = ...RETURNED FROM EMULATOR CARD. Contact Customer Service for ES Products.

8 ARGUMENT IS NOT A SIMPLE INTEGER OR INTERNAL RANGE. Don't cares are not allowed in this context.

9 NO MORE OVERLAY MEMORY AVAILABLE. You have not cleared the map or you are trying to map in more memory than is allowed. Contact Applied Microsystems for optional overlay memory expansion.

10 MULTIPLE-DEFINED EVENT GROUP. Only one group may be referenced in any event clause. Error is caused by trying to mix event register groups in an event clause

	(e.g., `2 WHEN AC1.3 THEN BRK` would cause this error).
11	ILLEGAL ARGUMENT TYPE FOR EVENT SPECIFICATION. Only the 8 event comparators may be used in the event portion of a WHEN/THEN statement.
12,13	ARGUMENTS MUST BE A SIMPLE INTEGER. Don't care masks and ranges not allowed.
14,15,16	OPERATION INVALID FOR THESE ARGUMENT TYPES. Usually caused by attempting arithmetic operations on incompatible variables (e.g., `(4 DC 9) + (IRA 500 to 700))`. (Same as error 23.)
17	SHIFT ARGUMENT CANNOT BE NEGATIVE. To shift a value in the reverse direction, use the opposite shift operator, `>>` or `<<`, not a negative shift value.
18	TOO MANY ARGUMENTS IN LIST . . . (9 MAX). When entering data in memory or I/O mode, a list of only 9 values can be entered on a single command line.
19	INVALID GROUP NUMBER . . . (NOT IN 1-4). There are only four event groups (1-4).
20,21,22,23	OPERATION INVALID FOR THESE ARGUMENT TYPES. Often caused by attempting arithmetic operations on incompatible variables.

24 BASE ARGUMENT MUST BE A SIMPLE INTEGER. Argument should be #0 to #16.

26 RANGE TYPE ARGUMENT NOT ALLOWED AS DATA. Data can only be expressed as masked values or integers.

27 ADDRESS ARGUMENT MUST BE A SIMPLE INTEGER. Cannot use ranges or masked values.

29 ILLEGAL DESTINATION - SOURCE TYPE MIX. Caused by trying to store don't care data into a range variable or other similar operations.

30,31 RANGE START AND END ARGUMENTS MUST BE SIMPLE INTEGERS. Cannot use masked values or ranges.

32 RANGE END MUST BE GREATER THAN RANGE START. `10 to 5` is an example of an invalid range.

33 RANGE START AND END ARGUMENTS MUST BE SIMPLE INTEGERS. Cannot use masked values or ranges.

34 READ AFTER WRITE-VERIFY ERROR. Data supposedly written to memory during a download operation was read back as a different value. The error message contains the locations and results of the comparison.

35 WARNING - DATA WILL BE LOST WHEN EMULATION IS BROKEN. Caused by assigning values to CPU registers during emulation. CPU registers are copied into internal RAM only when emulation is broken. The RAM contents are copied into the processor only when emulation is begun. The Emulator cannot access CPU registers during emulation. Thus, once emulation has been started the **DR** command shows the contents of the CPU registers as they were before emulation was begun. Changes can be made to these values, but the data will be rewritten when emulation is broken.

36,37,38 NO ROOM . . . BREAKPOINT CLAUSES TOO NUMEROUS OR COMPLEX. Too many WHEN/THEN clauses were entered.

39 INVALID GROUP NUMBER . . . (NOT IN 1-4). There are only four groups in the Event Monitor System.

40 ILLEGAL SELECT VALUE. Variable cannot be assigned value specified. Check manual.

41 INCORRECT NUMBER OF ARGUMENTS IN LIST. Check command argument list.

42 ILLEGAL SETUP SET VALUE. Consult **SET** menu for legal values (page 5-3).

43 "WHEN" CLAUSE REDUCED TO NULL FUNCTION. Caused by constructs such as `AC1 AND NOT AC1`.

44 INTERNAL ERROR . . . NULL SHIFTER FILE. Contact Customer Service for ES Products.

45	MAP CANNOT BE ACCESSED DURING EMULATION. The map hardware is constantly used by the emulating processor during emulation and cannot be accessed.
46	ARGUMENT MUST BE AN INTERNAL RANGE. External ranges and masked values not allowed.
47	16-BIT RANGE END LESS THAN START. Invalid range.
48	ILLEGAL MODE SELECT VALUE.
49,50	INVALID GROUP NUMBER . . . (NOT IN 1-4). Must be 1 through 4.
51	SAVE/LOAD INVALID ARGUMENT VALUE. Valid arguments include 0 through 5.
53	EEPROM WRITE VERIFY ERROR. Data in the EEPROM is verified during the SAV operation. (The store operation is retried many times before this error is generated.) EEPROMs have a finite write cycle life. The EEPROM in your Emulator is warranted for one year. Contact Applied Microsystems for service.
54	ATTEMPT TO SAVE/LOAD DURING EMULATION. These commands may only be used while in the pause mode.
55	EEPROM DATA INVALID DUE TO INTERRUPTED SAVE. Previous SAV command was interrupted by a reset or power off.

56	TRACE DATA IS INVALID DURING EMULATION. Viewing of the trace is only allowed during pause mode.
57	(INVALID GROUP NUMBER (NOT 1-4). Must use 1 - 4.
58	IMPROPER NUMBER OF ARGUMENTS. Check command argument list.
59	ARGUMENT MUST BE AN INTERNAL RANGE. External ranges and masked values not allowed.
60	ARGUMENT MUST BE A SIMPLE INTEGER. Ranges and don't care masks not allowed.
61	IMPROPER NUMBER OF ARGUMENTS. See error 58.
62	CANNOT STORE THIS VARIABLE DURING EMULATION. Must be in pause mode.
63	ILLEGAL ARGUMENT TYPE.
64	ARGUMENT TOO LARGE. Caused by entering **DRT** argument that includes numbers greater than #2045.
65	ILLEGAL RANGE.
66	STATUS CONSTANTS CANNOT BE ALTERED. System constants (e.g., **BYT**, **OVL**) cannot be assigned values.

67 TOO MANY "WHEN" CLAUSES. You have tried to enter more WHEN/THEN clauses than the Event Monitor System can handle.

68 INVALID DATA FORMAT FOR SYMBOLS. Must use Extended Tektronix Hex.

70 CANNOT INITIALIZE VECTORS DURING EMULATION. [LDV], [RNV], and [RBV] can only be entered in pause mode.

71 UNKNOWN EMULATOR ERROR. Call Applied Microsystems.

72 INCOMPATIBLE EEPROM DATA. Previous data saved to EEPROM was not from an 80286 Emulator system.

74 COMMAND INVALID DURING EMULATION. Must be in pause mode.

75 INVALID RECORD TYPE. Download routine received invalid record type code.

76 NO SYMBOLIC DEBUG. The symbolic debug option is not installed in your system. Cannot assign symbol and section values.

78,79,80 TOO MANY SYMBOLS. Symbols exceeded available RAM. Purge symbols before downloading again.

81 SYMBOL OR SECTION PREVIOUSLY DEFINED. You must delete a section before assigning it a new value.

82	SYMBOL NAME IN USE. Symbol name cannot be used more than once.
83	TYPE CONFLICT WITH DEFINED SYMBOL. Please refer to Extended Tek specification.
87	SECTION TABLE FULL. Too many symbolic section names have been defined.
88	INVALID ARGUMENT SIZE. Operand doesn't fit into destination register.
89	INVALID ADDRESSING MODE.
90	ARGUMENT OUT OF RANGE. Usually caused by reference to a "FAR" location without declaring "FAR."
91	INVALID TRAP VECTOR NUMBER.
92	INVALID OP CODE FOR 80286.
93	INVALID CONTROL REGISTER.
94	ARGUMENT NOT SYMBOLIC. Requires a symbolic argument.
97	ARGUMENTS MUST BE A VIRTUAL ADDRESS..
98	ARGUMENTS MUST CONTAIN IDENTICAL SELECTORS.
99	ARGUMENTS NEED 16 BIT LAYOUT.
100	VIRTUAL ADDRESS OFFSET OVERFLOW.

101	PROTECTION VIOLATION.
102	VIRTUAL ADDRESS ARGUMENT ILLEGAL.
103	LENGTH OF VIRTUAL ADDRESS RANGE MAY NOT EXCEED 64K.
104	NULL SELECTOR ILLEGAL.
105	COMMAND INVALID IN REAL MODE.
106	COMMAND INVALID IN PHYSICAL MODE.
107	NOT A VALID REGISTER.
108	CANNOT BE MODIFIED DURING EMULATION.
255	UNKNOWN ERROR.

SECTION 5

Table of Contents

System Commands

Page

Page

Page

SYSTEM COMMANDS

Setup Commands

The **SET** and **ON/OFF** commands allow you to configure the ES1800 according to hardware and debugging needs. There are two menus containing variables that are software selectable for quick and easy changes.

The **SET** menu contains all of the external communication variables such as baud rates, parity, and upload/download data format. Some set parameters require a reset before becoming effective. You can also set the serial communication parameters and save them to EEPROM without affecting the parameters currently in use.

The **ON/OFF** menu contains switches that control emulation and the serial port copy switch. For example, you can run the Emulator without a target system by using the Emulator-supplied clock signal, an Emulator-generated ready signal and overlay memory. The copy switch copies data to both serial ports for obtaining hard copy of your emulation session.

The **SET** menu and the **ON/OFF** menu can be saved to EEPROM after you have set them. These values may then be automatically loaded into the Emulator on power-up by setting the thumbwheel switch to the appropriate value, or manually by typing a load command (**LD**) to the Emulator after power-up.

The EEPROM is divided into two groups of six sections. Each section within a group may be loaded and saved individually. The two groups designate two users, referred to as user 0 or user 1 in the **SET** menu. This allows two users to save complete information about their emulation session, and reload it later. The six sections of information are:

Group #	Description
0	**SET** menu
1	Registers
2	Event Monitor WHEN/THEN clauses
3	Overlay map
4	**ON/OFF** menu
5	Macros

Command	Result
SET	Displays the **SET** menu. The parameters in this menu specify the external communication details.

```
>SET
ES SETUP: SEE MANUAL FOR DETAILS...

SET #X,#Y - SET ITEM X TO VALUE CORRESPONDING TO Y
LD 0;SAV 0  LOAD/SAVE SETUP FOR CURRENTLY SELECTED USER

SYSTEM:    #1 USER = 0; [0,1]
           #2 RESET CHAR = $1A
           #3 XON, XOFF = $11,$13

TERMINAL:  #10 BAUD RATE = #14; [2=110,5=300,10=2400,14=9600]
           #11 STOP BITS = 1 [1,2]
           #12 PARITY = 0; [0=NONE,1=EVEN,2=ODD]
           #13 CRT LENGTH = #24
           #14 TRANSPARENT MODE ESCAPE SEQUENCE = $1B,$1B

COMPUTER:  #20 BAUD RATE = #14; [7=1200,12=4800,15=19200]
           #21 STOP BITS = 1
           #22 PARITY = 0
           #23 TRANSPARENT MODE ESCAPE SEQUENCE = $1B,$1B
           #24 COMMAND TERMINATOR SEQUENCE = $0D,$00,$00
           #25 UPLOAD RECORD LENGTH = #32; [1 to 127]
           #26 DATA FORMAT = 0; [0=INT,1=MOS,2=MOT,3=SIG,4=TEK,5=XTEK]
           #27 ACKNOWLEDGE CHAR = $06
```

(continued)

SET *<parameter>*, *<exp>*	The value of the specified parameter is changed to *<exp>*. If you assign an illegal value to a variable, an error message is displayed, and the value is not changed.

Comments

The table below shows the valid values for each **SET** variable. All arguments preceeded with a [$] indicate that the value entered must be a 7-bit ASCII character.

The [#] preceding the **SET** command arguments below is typed in and designates the value entered as decimal. The [#] is optional for decimal numbers 0-9.

Parameters	Description	Reset Required
SET #1,#0	User 0	No
SET #1,#1	User 1	No
	Two users may save and load values to the EEPROM. This parameter indicates which user is active when executing the **SAV** and **LD** commands.	
SET #2,$n	Reset character	No
	The reset character resets the Emulator and the pod CPU. The system default is CTRL Z ($1A).	
SET #3,$n,$m	XON/XOFF characters	No
	XON and XOFF control the screen scrolling. An XOFF stops a scrolling display. XON resumes scrolling the display. The system defaults are CTRL Q, CTRL S ($13, $11).	

Parameters	Description	Reset Required
SET #10,#1	75 baud	Yes
#2	110 baud	
#3	134.5 baud	
#4	150 baud	
#5	300 baud	
#6	600 baud	
#7	1200 baud	
#8	1800 baud	
#9	2000 baud	
#10	2400 baud	
#11	3600 baud	
#12	4800 baud	
#13	7200 baud	
#14	9600 baud (default)	
#15	19200 baud	
	The terminal port baud rate	
SET #11,#1	1 stop bit (default)	Yes
#2	2 stop bits	
	The number of stop bits for the terminal port	
SET #12,#0	No Parity (default)	Yes
#1	Even Parity	
#2	Odd Parity	
	The parity for the terminal port	
SET #13,#n	CRT length (default: 24 lines)	No
	The maximum number of lines displayed for commands that use paging	

(continued)

Parameters	Description	Reset Required
SET #14,$n,$m	Transparent mode escape sequence When entered from either port, transparent mode is terminated. The default sequence is ESC, ESC ($1B,$1B).	No
SET #20,#1	75 baud	Yes
#2	110 baud	
#3	134.5 baud	
#4	150 baud	
#5	300 baud	
#6	600 baud	
#7	1200 baud	
#8	1800 baud	
#9	2000 baud	
#10	2400 baud	
#11	3600 baud	
#12	4800 baud	
#13	7200 baud	
#14	9600 baud (default)	
#15	19200 baud	
	The computer port baud rate	
SET #21,#1	1 stop bit (default)	Yes
#2	2 stop bits	
	The number of stop bits for the computer port	
SET #22,#0	No parity (default)	Yes
#1	Even parity	
#2	Odd parity	
	Parity for the computer port	

Parameters	Description	Reset Required
SET #23,$n,$m	Transparent mode escape sequence	No
	When entered from the computer port, transparent mode is exited. The default sequence is ESC, ESC ($1B,$1B).	
SET #24,$n,$m,$o	Command terminator sequence	No
	The default sequence is [RETURN], null, null ($0D, $00, $00).	
SET #25,#n	Upload record length	No
	The maximum length for an upload record. (The default length is 32 bytes of data.)	
SET #26,#0	Intel (default)	No
#1	MOS	
#2	Motorola	
#3	Signetics	
#4	Tektronix	
#5	Extended Tekhex	
	Upload/download serial data format	
SET #27,$n	Acknowledge character	No
	The acknowlege character is sent when a valid record is received when downloading in computer control. The default is $06.	

(continued)

Comments

Some **SET** parameters require the system to be reset, and prompt for a reset character. If you change a parameter that requires a reset, but do not enter one, subsequent displays of the **SET** menu show the new value you have assigned the variable, even though it is not currently in effect.

If you change the **SET** parameters and wish to use the new values at a later date, you can save them in EEPROM by entering a **SAV** or **SAV 0** command.

Saved parameters can be loaded automatically at power-up or manually after the system is up and running. To load automatically, set the thumbwheel switch (see page 3-5) before turning on the Emulator. To load manually, enter **LD** (to load all variables and settings) or enter the **LD 0** command (to load just the **SET** parameters).

See Serial Communication (page 5-38) for information on communicating with a host computer.

Command	Result
ON **OFF**	Displays the **ON/OFF** menu.

```
                    ES SWITCH SETTING MENU

NAME  STATE    DESCRIPTION

BKX    OFF     BREAK ON INSTRUCTION EXECUTION (NOT PREFETCH)
BTE    OFF     BUS(RDY) TIMEOUT ENABLE
CEE    OFF     COPROCESSOR ERROR ENABLE
CK     ON      SELECT INTERNAL CLOCK
CPY    ON      COPY DATA TO TERMINAL & COMPUTER PORTS
DBE    OFF     DESCRIPTOR TABLE BREAK ENABLE
ENI    ON      ENABLE INTERRUPTS
FSX    ON      FSI ON INSTRUCTION EXECUTION (NOT PREFETCH)
IEE    OFF     INTERRUPT ERROR ENABLE
IHE    OFF     IGNORE HALT ERRORS
IM     OFF     INTROSPECTIVE MODE
PEN    ON      PAUSE MODE PROTECTION ENABLE
PPT    OFF     TRACE PEEK AND POKE CYCLES
RDY    ON      SELECT INTERNAL READY WHEN ACCESSING OVERLAY
RMS    OFF     REAL MODE SELECT
SLO    OFF     SLOW INTERRUPT ENABLE
STI    OFF     ENABLE STEP THROUGH INTERRUPTS
 >
```

Comments

(continued)

Comments

Some switches cannot be set during run mode.

The arguments to the **ON** and **OFF** commands are the names of the switches themselves. These are:

Switch	Description	Default	Changeable During Emulation
BKX	Break on instruction execution	OFF	yes
BTE	Bus (RDY) timeout enable	OFF	yes
CEE	Coprocessor error enable	OFF	yes
CK	Internal/external clock	OFF	no
CPY	Copy data to both serial ports	OFF	yes
DBE	Descriptor table break enable	OFF	no
ENI	Enable interrupts	ON	yes
FSX	Force special interrupt on instruction execution	ON	yes
IEE	Interrupt error enable	OFF	yes
IHE	Ignore halt errors	OFF	yes
IM	Introspective mode	OFF	no
PEN	Pause mode protection enable	ON	no
PPT	Trace peek and poke cycles	OFF	yes
RDY	Internal/external ready signal	OFF	no
RMS	Real mode select	OFF	no
SLO	Slow interrupt enable	OFF	yes
STI	Step through interrupts	OFF	yes

You may turn on or off multiple switches by listing them with a + between their names.

All switches can be turned off with the command, `>OFF -1`.

You can save all of the current switch settings in EEPROM for later use by executing a **SAV** (to save all variables and settings) or **SAV 4** (to save just switch settings) command (see page 5-36).

The saved switches can be loaded automatically at power-up or manually after the system is up and running. To load automatically, set the thumbwheel switch (see page 3-5) before turning on the Emulator. To load manually, enter a **LD** (to load all variables and settings) or **LD 4** (to load just the switch settings) command (see page 5-34).

Examples

If you want a hard copy of an emulation session, attach a printer to the computer port on the back chassis of the Emulator. Turn on the copy switch so that all data is copied to both serial ports.

```
>ON CPY
>
```

Assume that you are debugging a program on a new piece of hardware. The program has already been debugged using the Emulator's overlay memory and appears to be functioning properly. When you try to run the program in the hardware it does not work correctly. In this case you may want to switch back and forth between running from overlay memory and the target. When running out of overlay you want to use an internal clock and ready signal. You do this with these two commands:

```
>ON RDY+CK
>OFF RDY+CK
```

Here are two alternative methods for doing the same thing using fewer keystrokes.

The first is to use a general purpose register for the command parameter. Assign the register the switch names. Then use the register as the parameter for the commands.

(continued)

```
>GRO = RDY+CK

>ON GRO
>OFF GRO
```

The next way is to use two macros for the commands. Assign macros 1 and 2 to the **ON** and **OFF** commands. Execute these macros by typing a [.] and [,] as the first character on each line (see page 5-107).

```
>_1=ON RDY+CK
>_2=OFF RDY+CK

>.
>,
```

BREAK ON INSTRUCTION EXECUTION

Command	Result
ON BKX	The Event Monitor System breaks on the execution of the instruction rather than the instruction pre-fetch.
OFF BKX	The Event Monitor System breaks whenever an address is seen on the bus. (Break on prefetch of instruction.) Default: OFF

Comments

The 80286 prefetches instructions. Because of this, an address can be detected on the address bus before the instruction is actually executed. If you set a breakpoint on an address that immediately follows a branch, the Emulator may break, even before the instruction is executed (it was prefetched). Set this switch to force the break to occur only on address execution.

When "breaking on instruction execution" the address entered into the address comparator register must be the address of the beginning of the instruction. The addresses are prequalified by the prefetch BRK hardware. Thus the address that causes the "break on instruction execution" must be seen first as a prefetched address.

BUS TIMEOUT ENABLE

Command	Result
ON BTE	ON BTE allows hardware to return a RDY to the CPU if there has been no activity with ALE for longer than 1.3 seconds. When the CPU waits longer than .5 seconds for a RDY signal, the message PROCESSOR WAITING is displayed. When bus timeout occurs, a RDY is asserted to the CPU. A BRK is also asserted to the CPU, causing emulation to break. This saves the contents of the CPU registers and trace at the point where the CPU stopped executing. The error message BUS TIMEOUT ERROR is displayed after emulation breaks. During peek/poke cycles the Emulator displays the message PROCESSOR WAITING if the CPU has been waiting for a RDY more than .5 seconds. After 1.3 seconds, the Emulator returns a RDY to the CPU and displays the data that was read.

OFF BTE

The processor waits for a ready signal. After .5 seconds, an error message displays.

Default: OFF

Comments

Bus timeout is disabled when a HOLDA or HALT is asserted by the CPU regardless of the state of the BTE switch.

COPROCESSOR ERROR ENABLE

Command	Result
ON CEE	When PEREQ, BUSY, or ERROR are asserted for longer than .5 seconds, PEREQ ASSERTED, INTR ASSERTED, or ERROR SIGNAL ASSERTED, respectively, are displayed. This may or may not indicate an error, depending on the target system design.
OFF CEE	When PEREQ, BUSY, or ERROR are asserted for longer than .5 seconds, PEREQ ASSERTED, INTR ASSERTED, or ERROR SIGNAL ASSERTED are not displayed. Default: OFF

INTERNAL/EXTERNAL CLOCK

Command	Result
ON CK	The CPU uses an internally generated clock. This is a nonadjustable clock set at 12.5 MHz (CPU speed).
OFF CK	The CPU uses the target system clock. Default: OFF

Comments

This command is valid only in pause mode.

Use an internal clock when debugging code before target hardware is available. Download the program to overlay memory. Turn on the internally generated ready signal and clock (**ON RDY** and **ON CK**) and begin debugging.

See also the Download command, pages 5-46, the overlay memory section, page 5-62, and the RDY switch, page 5-29.

COPY DATA TO BOTH PORTS

Command	Result
ON CPY	Sends all data to both the terminal and computer ports. Data sent to the controlling port is echoed to the other port (noncontrolling port).
OFF CPY	Only sends data from the Emulator to the controlling port. Default: OFF

Comments

This provides a way to make a hard copy of emulation data. It is also useful for monitoring computer control commands.

See Serial Communications, page x, for more information on the terminal and computer ports.

The controlling port is determined by the thumbwheel switch on power-up and by the **SET** menu after power-up.

DESCRIPTOR TABLE BREAK

Command	Result
ON DBE	**ON DBE** causes the Emulator to break emulation when any descriptors currently being used by the Event Monitor System are modified.
OFF DBE	Ignores writes to the descriptor table. Default: OFF

Comments

The descriptor breakpoint hardware is separate from the Event Monitor System and thus does not take away any power from the Event Monitor System.

Examples

```
AC1 = 8:0
```

(continued)

This address comparator is set to an address that uses descriptor table entry 1 in the global descriptor table. If DBE is ON and a write to descriptor table entry 1 in the global descriptor table is detected, the Emulator breaks and the message DESCRIPTOR TABLE MODIFIED is displayed.

ENABLE INTERRUPTS

Command	Result
ON ENI	Enables interrupts upon entering emulation. May be used in conjunction with the SLO switch.
OFF ENI	Disables interrupts.
	Default: ON

Comments

This switch has no effect on the STI (step through interrupts) switch.

FSI ON INSTRUCTION EXECUTION

Command	Result
ON FSX	An Event Monitor System forced special interrupt (FSI) occurs when an instruction is executed. Refer to page 7-25 for the **FSI** command.
OFF FSX	Forced special interrupt (FSI) occurs when an address is seen on the bus (on prefetch of instruction). Default: ON

Comments

The 80286 prefetches instructions. Because of this, an address can be detected on the address bus before the instruction is actually executed. If you set an FSI on an address that immediately follows a branch, the Emulator may execute the FSI before the instruction is executed (it was prefetched). Set this switch to force an FSI to occur only on address execution.

When entering addresses that will cause an **FSI** on instruction execution, the addresses must indicate the beginning of the instruction. All addresses to be qualified on execution are also prequalified by the Event Monitor System as prefetch addresses.

INTERRUPT ERROR ENABLE

Command	Result
ON IEE	When INTR is asserted for longer than .5 seconds, the error message INTR ASSERTED .5 SECS is displayed. This may or may not indicate an error, depending on the target system design.
OFF IEE	When INTR is asserted for longer than .5 seconds, the error message INTR ASSERTED is not displayed. Default: OFF

IGNORE HALT ERRORS

Command	Result
ON IHE	**ON IHE** inhibits the HALT ASSERTED error message from appearing on the screen. The user may assert this switch when the halt mode is entered multiple times and causes the message to scroll across the screen.
OFF IHE	Halt mode is monitored, and if active for more than .5 seconds, an error message is displayed. Default: OFF

Command	Result
ON IM	Uses the Emulator's internal memory space as the target system. Peeks and pokes are directed toward the Emulator's internal memory space. You cannot run or step in this internal environment.
OFF IM	Hardware connected to the Emulator is regarded as the target system. All peeks and pokes are directed to the target environment. Default: OFF

Comments

This command is valid only in pause mode.

PAUSE MODE PROTECTION

Command	Result
ON PEN	Violations of the Intel protection mechanisms cause "PROTECTION VIOLATION:" type errors or "ERROR #104 NULL SELECTOR ILLEGAL" to display.
OFF PEN	Violations of the protection mechanisms are ignored. If you set a selector value (CS, DS, SS, ES), the access rights and limits are overridden. You will see the error message "OFF PEN: OVERRIDING ACCESS RIGHTS AND LIMITS." Default: **ON**

Command	Result
ON PPT	Traces peek and poke cycles as specified by the Event Monitor System. If the Event Monitor System is cleared and PPT is ON, all peek and poke cycles are traced.
OFF PPT	The Emulator does not trace peek or poke cycles regardless of what is specified by the Event Monitor System. The trace buffer only records during execution of the **RUN** command (as denoted by the [R>] prompt) or a **STP**. Default: OFF

Comments

This command is valid only in pause mode.

If peek or poke cycles are traced, they may corrupt trace disassembly.

(continued)

Examples

```
ON PPT
AC1 = 5550
CES; WHEN AC1 THEN TRC
ITR
SF 1, 5000 TO 5FFF
```

All peeks, pokes, and memory cycles at the specified memory location are traced by trace memory. To display the transactions that occurred during the overlay memory diagnostic, use the **DRT** command. In this particular case, all accesses to 5550 are traced. Peek or poke cycles occur during memory display commands (e.g., **DB, MM**) and memory modify commands (e.g., **MM, DNL, SF, BMO**).

NOTE: The Event Monitor System must be initialized via the **ITR** command (see page 7-17) *after* setting up the WHEN/THEN statements and Event Monitor registers and *before* initiating the desired peek/poke cycle.

INTERNAL/EXTERNAL READY SIGNAL

Command	Result
ON RDY	Selects an internally generated ready signal to complete memory accesses. This allows use of overlay memory when no target system is being used.
OFF RDY	Selects the target system's ready signal to complete memory accesses. Default: OFF (See note on the next page.)

Comments

A ready signal denotes the end of a memory cycle. See the *Intel iAPX286 Hardware Reference Manual* and the *Intel iAPX286 Programmer's Reference Manual* for details.

If overlay memory is mapped in an area where target memory is nonexistent, the target decode logic may not provide a ready signal. An **ON RDY** provides this signal, allowing overlay memory to be used in those areas.

When the ready switch is on and the target system is also providing a ready signal, the first ready signal back to the Emulator will be the one used.

If internal ready is selected and there is a target, there is no synchronization between the ready signal and the target hardware. This can cause problems if a ready is returned by the Emulator before the target hardware is ready.

(continued)

Do not use **ON RDY** when overlaying dynamic RAM.

NOTE: Default is ON if there is no target clock on power-up and if internal clock has been selected.

REAL MODE SELECT

Command	Result
ON RMS	The Emulator register ADF is forced on the address line 20-23 when going into run mode or peeking and poking. The register ADF may be set to \$0-\$F.
OFF RMS	Address lines A20-A23 are defined as high at power-up and low after the first CS (code segment register) change until virtual protected mode is entered. Default: OFF

Comments

Before entering virtual mode, RMS must be OFF or the incorrect address is seen on A20-A23.

The RMS switch cannot be modified during emulation.

SLOW INTERRUPT ENABLE

Command	Result
ON SLO	Interrupts are enabled after 14 instruction fetches.
OFF SLO	Interrupts are enabled immediately upon entering run mode. Default: OFF

STEP THROUGH INTERRUPTS

Command	Result
ON STI	The Emulator recognizes an interrupt and steps through the interrupt service routine.
OFF STI	The Emulator ignores interrupts while stepping through a program. Default: OFF

Comments

Stepping through code is a common way to locate software bugs. This switch allows you to ignore interrupts while debugging higher level routines, or to step through and debug the interrupt routine itself.

See also the Step command (**STP**) on page 6-23.

The ENI and SLO switches do not affect the STI switch.

LOAD SYSTEM VARIABLES FROM EEPROM

Command	Result
LD	Copies all system variables stored in EEPROM into Emulator memory.
LD <category>	Copies the variables from one of the six categories in the EEPROM to the Emulator RAM.

Comments

This command is valid only in pause mode.

Executing a **LD** command reads system variables from the EEPROM and copies them to into internal ESL RAM. The EEPROM retains those original variables until replaced by a **SAV** command.

There is room in the EEPROM to load the system variables for two different users. The user is determined by a parameter in the **SET** menu.

You may load the following variable categories from EEPROM:

```
0 - SET menu
1 - Contents of Emulator registers
2 - Event Monitor System WHEN/THEN statements
3 - Overlay map
4 - Software switch settings
5 - Macros
```

Examples

```
>LD 3
```

The overlay memory map in the EEPROM is copied into internal RAM. This causes overlay to be mapped. Use the **DM** command to verify the new map. (See page 5-63.)

SAVE SYSTEM VARIABLES IN EEPROM

Command	Result
SAV	Copies all system variables from Emulator memory into EEPROM.
SAV *<category>*	Saves one of the six categories of variables from Emulator RAM to EEPROM.

Comments

This command is valid only in pause mode.

A SAV operation may take up to two minutes.

DO NOT INTERRUPT THE PROCESS!

Values saved to EEPROM continue to be valid within the Emulator.

There is room in EEPROM to save the system variables for two different users. The user is determined by a parameter in the **SET** menu. When you execute a **SAV**, the variables are saved to the user partition currently defined in the **SET** menu.

This chart shows the categories of information that can be saved in EEPROM and the corresponding page numbers to find more information.

```
0 - SET menu                               page x
1 - Contents of Emulator registers         page x
2 - Event Monitor System                   page x
    WHEN/THEN statements
3 - Overlay map                            page x
4 - Software switch settings               page x
5 - Macros                                 page x
```

Variables are loaded from EEPROM back to the Emulator using the **LD** command.

When you first use the Emulator, you should execute a **SAV** command with no parameter. This initializes EEPROM, so that subsequent **LD** commands will work properly with the 80286 Emulator board and pod.

Examples

```
>SAV 1
```

Saves the current values of all the Emulator registers in EEPROM.

(continued)

Serial Communications

The ES1800 can communicate through both DB-25 connectors on the chassis rear panel using standard RS232C serial protocol. The ports can be independently configured for baud rate, data length, and number of stop bits.

USING A HOST COMPUTER

The most common development configuration is with a terminal connected to the terminal port of the ES1800 and a host development system connected to the computer port. The ES1800 provides a transparent mode that essentially connects your terminal to the computer. The ES1800 also has a special download command to load modules from the host system.

In configurations where the ES1800 is connected directly to a host computer, there are a few details that need to be considered.

DATA BUFFERING AND BAUD RATE

When downloading from a computer, the ES1800 buffers all the data bytes until the end of record. If the checksum is correct, the data are then loaded into target memory. During this load time, the host computer may start sending the next data record. The serial data buffer in the ES1800 is 64 bytes deep. When the sixth character is placed in the buffer, an XOFF character is sent to the host computer. This means that the host computer must transmit no more than 58 characters after the XOFF. Some multi-tasking development systems may not be capable of quickly stopping character transmission. For these systems, it may be advisable to lower the computer port's and host computer's baud rates.

XON and XOFF characters can be used to control either output port on the ES1800. These characters are user definable. The problem described in the above paragraph can happen in the reverse direction. If the ES1800 is uploading data to the host, it may be able to overrun the host's ability to receive characters. While lowering baud rates may help, there are probably commands available on the host to solve the problem. You should also make sure that the host does not echo characters sent to it while uploading data. If the characters are echoed, the ES1800 will quickly send an XOFF to the host while continuing to send normal upload characters. The host system will then probably send an XOFF to the ES1800 because the host's buffers are full. The result of this situation is that both systems will lock up waiting for the other to send an XON. See your system administrator or call Applied Microsystems Corporation customer service at 1-800-426-3925 for help.

COMMUNICATION WITH THE HOST COMPUTER

While in transparent mode, the ES1800 passes characters between the computer and terminal ports. There is a user definable two-character escape sequence to exit transparent mode. If the first character of the escape sequence arrives at either port, the ES1800 holds it until it receives another character from the same port. If the second character matches the second character of the escape sequence, transparent mode is terminated. If the second character is not part of the escape sequence, then both the character being held and this second character are sent to the proper port. See page 5-3 for setting the escape character sequence.

While in transparent mode, the only characters that are meaningful to the ES1800 are XON, XOFF, the first character of the escape sequence, and the reset character. The reset character may be sent from the host as part of a command sequence to the terminal. This is common during edit sessions and depends on the command set of your terminal. You should define the reset character to be a character that will not normally be used by the host system.

PORT DEPENDENT COMMANDS

Most commands are symmetric with respect to the controlling port and appear to respond in the same manner if entered from either the computer port or the terminal port. The "controlling" port is determined at power-up by the setting of the thumbwheel switch on the controller board (see page 3-5). After power-up, the commands CCT and TCT switch control from one port to the other. TCT entered to the terminal port acts like a null command as does a CCT entered at the computer port.

Entering transparent mode from either port causes both ports to be "connected" to each other. If transparent mode is terminated from either port, control returns to the port that initiated the transparent mode (**TRA**) command.

(continued)

DOWNLOAD FROM TERMINAL PORT

When the ES1800 receives a download command (**DNL**), it always expects data records to arrive at the computer port. If the download command is entered from the terminal port, the ES1800 automatically enters transparent mode to allow you to send commands to the host system. You normally enter a command that causes the host system to copy the formatted object file to your terminal (see page 5-3 for setting serial data format). The proper procedure is to enter the command to the host system but not terminate it (i.e., do not press the RETURN key). Instead, enter the two-character transparent mode escape sequence. When transparent mode terminates, control returns to the download process. The download routine then sends the user definable command terminator sequence to the host system (see page 5-3 for setting the command terminator sequence). The host system responds by sending the data records from the formatted object file. Any characters sent by the computer are echoed to the terminal port. All valid data records are copied into internal buffers and the data written into target memory. When the End of File (EOF) record is received, the download process terminates and a normal ESL prompt is displayed.

If an error occurs (checksum or read-after-write) during the download, the process terminates with an error and a new prompt is displayed. No special characters are sent to the host, however, so it is likely that the next time you enter transparent mode, the host will send the remainder of the download data records.

DOWNLOAD FROM COMPUTER PORT

If the download command is entered from the computer port, the process is different. In this case, the ES1800 does not enter transparent mode. The **DNL** command can be immediately followed by data records. Each data record is acknowledged with an ACK (6) character if its checksum is correct and correctly written into target memory (verified with read-after-write cycles). The EOF record is also acknowledged if valid. If an error occurs during a download, the first character sent back to the host will be the BEL (7) code. Programs written on the host system can use these two characters to handshake the data records in an automatic download routine.

Command	Result
TRA	The system enters transparent mode.
ESC ESC	Port control is returned to the previous settings. Note that this escape sequence can be changed using the **SET** command (page 5-3).

Comments

Transparent mode can be entered while in terminal (**TCT**) or computer control (**CCT**) modes.

In transparent mode the Emulator acts only as an interface between the two serial ports. The Emulator can buffer up to 64 characters for each port and can operate each port at independent baud rates.

There must be devices connected both to the terminal port (such as a terminal) and the computer port (host system, line printer) for this command to have any meaning.

Transparent mode is used to communicate with a host computer or any other peripheral you want to attach to a serial port.

Refer also to Serial Communications (page 5-38).

(continued)

Examples

```
>TRA
```

Data entered at either port is transmitted directly to the other port.

TERMINAL PORT CONTROL

Command	Result
TCT	The terminal port becomes the controlling port.

Comments

This command, along with the **CCT** command, allows control to be switched between to two serial ports without powering down the ES1800 Emulator.

Any output generated by a command is directed to the controlling port. The copy switch directs output to both serial ports.

This command is essentially a null command when entered from the terminal port. (See page 5-39 for a discussion of port dependent commands.)

Port selection on power-up is controlled by the thumbwheel switch setting. (See page 3-5.)

COMPUTER PORT CONTROL

Command	Result
CCT	The computer port becomes the controlling port.

Comments

This command, along with the **TCT** command, allows control to be switched between the two serial ports without powering down the ES1800 Emulator.

Any output generated by a command is directed to the controlling port. The copy switch directs output to both serial ports.

This command is essentially a null command when entered from the computer port.

If there is a host attached to the computer port and you type a **CCT** from a terminal connected to the terminal port, the host system takes control of the Emulator. The host system must be able to handle incoming data at high rates. Both hardware and software handshakes are supported (see page 3-12).

The upload and download operations always send/receive data from the computer port regardless of which port is the designated controller.

If you execute **CCT** in error with no terminal or host system connected to the computer port:

- Move the terminal cable to the computer port, enter the **TCT** command and return the cable to the terminal port.

  This process will work in most cases to return control to terminal. If not:

- Turn the Emulator off and then on.

This command can be executed from the computer port (see page 5-39 for a discussion of port dependent commands). For port selection on power-up refer to page 3-5.

DOWNLOAD OPERATIONS

Command	Result
DNL	**DNL** readies the Emulator to receive data. If in terminal control mode, the Emulator enters a transparent mode automatically, allowing direct communication with the host system. Other host system commands may be executed prior to the download operation.

Comments

You can choose the destination of the downloaded file:

- Target memory
- Emulator overlay memory

If the downloaded data is going to overlay memory, verify that the overlay is mapped in the appropriate address range. Make sure that the start address of the file is the address to which you expect to download.

Verify also that the data format of the host system file matches that being used by the Emulator. Refer to **SET** menu set parameter #26 for verification of Emulator format. Use transparent mode (**TRA**) to verify host system format and the address in the file. (See page 5-41.)

You can download files with either the computer port or the terminal port in control. That is, the downloading of files can be initiated and controlled either by the user or by a host system. There are some differences in procedure depending on which port is in control of the downloading process.

DOWNLOADING UNDER TERMINAL PORT CONTROL

After typing **DNL**, the system automatically enters transparent mode, allowing communication with the host system. When you are ready to download the file, enter a command that causes the host system to display a file to the terminal, but *in place of a* [RETURN], enter the transparent mode escape sequence (see page 5-41).

The Emulator is now ready to read the data records the host system will be sending. Data records are displayed as they are received by the Emulator. Checksums are verified and if a checksum error occurs, the download is aborted with an error message. The data in the erroneous record will not have been written to memory.

Each data byte is verified with a "read after write" cycle. If an error is detected, the download is aborted.

RETURN CONTROL TO EMULATOR

Once the download command (**DNL**) is entered, control is returned to the Emulator in one of three ways:

1. An end of file record is received. If an end of file record is not recognized by the Emulator, control will *not* be returned to the Emulator terminal port. This can be caused by:

   - Using a [RETURN] instead of the proper escape sequence to terminate the command line to the host computer

   - Selecting the incorrect data format.

(continued)

2. An Emulator reset is executed (default is CTRL Z).

3. An error is detected.

DOWNLOADING UNDER COMPUTER PORT CONTROL

To download while in computer control with a host computer attached, the host computer should send:

```
>DNL
```

After the host sends the download command, the Emulator waits for data at the computer port. The host computer should then send the downloadable records followed by an end of file record. After the end of file record, the system prompt (>) is sent to the computer port.

An acknowledge character (factory default is ASCII ACK $06) will be sent to the computer port after storing a data record, when in computer control. No acknowledgments are sent when in terminal control.

There are some differences between computer port control and terminal port control during the downloading process. Under computer port control:

- All good records are adknowledged with an ACK $6.
- All error messages from bad records are received on the computer port; therefore the host file that is controlling the Emulator will need to be able to interpret error messages.
- Records are not echoed.

SYMBOLIC DOWNLOAD

The download command accepts symbolic definition records as well as data records when the symbolic debug option is used and the Emulator download format variable is set to 5 (Extended Tekhex). (See **SET** parameter #26, page 5-3.)

Serial data can be verified with memory constants using the VFY command.

Errors

CHECKSUM ERROR IN THE DATA RECORD

The download process is aborted because the checksum sent with a record file is not the same as the checksum calculated by the Emulator.

READ AFTER WRITE VERIFY ERROR

Every byte in a data record is verified after it is stored. This error indicates that the data in memory does not match the data that was stored.

Problem	What to Check
Emulator does not return a prompt after file has been sent.	1. Serial data format - **SET** menu. 2. No end of file (EOF) record. 3. You entered a [RETURN] instead of the transparent mode escape sequence after entering the host copy command
Read-after-write verify error.	1. Target hardware problem.

(continued)

Problem	What to Check
	2. Overlay memory not mapped in download range. Address is indicated by misverify message.
Checksum error.	1. Improperly formatted record sent by host. 2. Noisy serial data lines. 3. Host computer is not responding to XON/XOFF protocol.
Display of data does not commence after entering transparent mode escape sequence.	1. Host not responding to user defined command terminator sequence - **SET** menu (page 5-3).

If the Emulator does not return a prompt, you will need to reset the system (default is CTRL Z) in order to enter any other Emulator commands.

If the host computer does not respond to the XON/XOFF protocol fast enough, you may need to lower the baud rate on the computer port and the host computer.

VERIFY SERIAL DATA

Command	Result
VFY	Verifies serial data with data in memory. If the data in memory does not match the incoming serial data, this message is displayed: `ADDRESS = XX NOT YY` *Address* is the address where the data mismatch occurred. *XX* denotes the actual data present at that location. *YY* is the serial data just sent.

Comments

This command is similar to the download command but no data is written to memory, and the serial data is not displayed on the screen. The serial data is compared to the data in target or overlay memory. Mismatches are displayed.

Use this command if you suspect a file you downloaded was corrupted. If downloaded data is being corrupted by your program, you can detect it by mapping overlay as RO (read only) (see page 5-67).

This command is also useful for determining differences between object files. Follow instructions for downloading a file on page 5-40.

UPLOAD SERIAL DATA

Command	Result
UPL *<range>*	The Emulator formats and sends data to the computer port.

Comments

Data is transferred from the Emulator to a host system or other peripheral interfaced to the Emulator computer port.

When uploading to a file on a host system, enter transparent mode first and open a file to store the uploaded data records. (Review the Serial Communications discussion, pages 5-38 - 5-40.)

Examples

For UNIX:

```
Cat ><filename>
```

For VMS:

```
TYPE ><filename>
```

(Create or EDT are also acceptable.)

For CPM:

```
PIP A:<filename> = RDR:
```

Next, type the transparent mode escape sequence and the upload command.

After all data has been uploaded and the Emulator prompt returns, enter transparent mode and close the file by entering the appropriate control character.

Remember to close the file *before* trying to view it.

If the host system does not respond to XON/XOFF protocol, it may be necessary to lower the communicating port's baud rates so that the host's input buffer is not overrun.

Upload performs no data verification.

A file may be uploaded to a printer, PROM programmer, or other peripheral instead of to a host. In this case, there is no need to enter transparent mode before uploading. Just be sure the peripheral is ready to receive data.

Refer also to Serial Communications, page 5-38.

UPLOAD SYMBOLS

Command	Result
UPS	All currently defined symbols and sections are sent to the computer port in extended tekhex format.

Comments

Extended tekhex restricts the number and range of characters that can be used for symbol names. When formatting symbols for upload, the Emulator truncates symbol names to 16 characters and substitutes % for characters not allowed by tekhex.

Extended tekhex serial data format should be set before uploading symbols (**SET** parameter #26,5).

When uploading to a file on a host system, enter transparent mode first and open a file to store the uploaded data records. (Review the Serial Communications discussion, pages 5-38 - 5-40.)

Examples

For UNIX:

```
Cat ><filename>
```

For VMS:

```
TYPE ><filename>
```

(Create or EDT are also acceptable.)

For CPM:

```
PIP A:<filename> = RDR:
```

Next, type the transparent escape sequence and begin uploading.

After all data has been uploaded and the Emulator prompt returns, enter transparent mode and close the file by entering the appropriate control character.

Remember to close the file *before* trying to view it.

Refer also to Symbols, page 5-114.

COMMUNICATION WITH TARGET PROGRAMS

Command	Result
COM *<address>*	Establishes communication with the target program through a two-byte psuedo-port at the specified address. Exit **COM** mode by entering the two-character transparent mode escape sequence (see **SET**, page 5-3).

Comments

Only useful during run mode.

Affects real-time operation.

Requires special target code. **COM** mode uses two bytes at the specified address. The byte at *<address>* is used for characters sent from the target to the controlling port. The byte at *<address>* + 1 is used for characters being sent to the target program. This command makes use of 7-bit ASCII characters, with the eighth bit of each byte used for handshaking.

To transmit a character to the ES1800, the target program first checks the most significant bit (MSB) of the byte at *<address>*. If this bit is set (1), the Emulator has not yet collected the previous character. If the bit is cleared, the target program sets the MSB of the character to be transmitted and places the result in the byte at *<address>*.

To receive a character from the Emulator, the target examines the byte at *<address>* + 1. If the MSB of this byte is cleared, the Emulator has not yet transmitted a new character. If the MSB is set, the character is "new." If the controlling port of the ES1800 is a terminal, the target program should echo the character by immediately copying it into the byte at *<address>* with the MSB still set. The target then program masks the MSB off and stores the result back at *<address>* + 1. This prevents the target program from re-reading the same character.

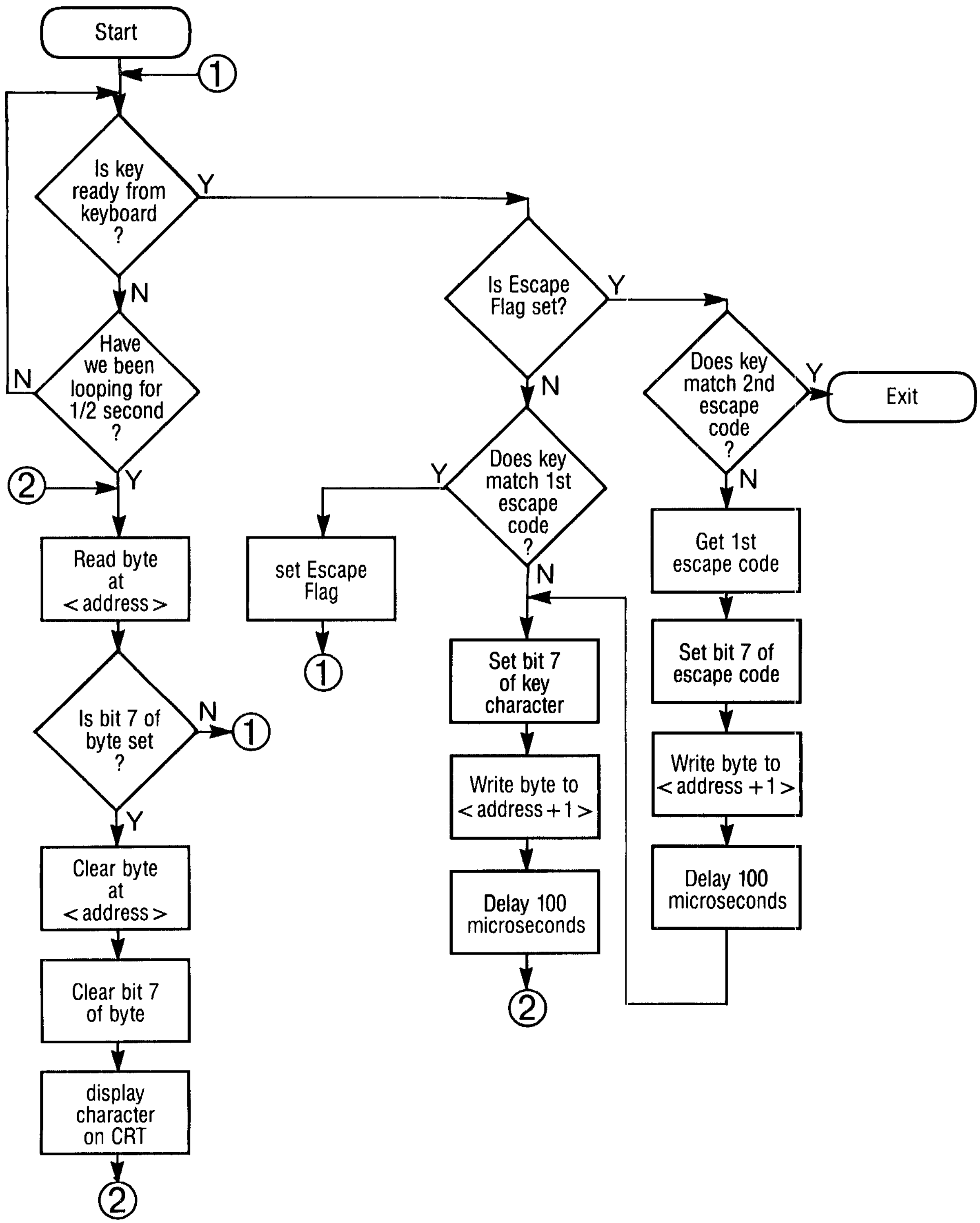
The **COM** routine does not check the byte at *<address>* + 1 to see if the target program has received it. Generally, the target program will be substantially faster than the **COM** routine and will always receive one character before the **COM** routine can transmit the next.

In effect, the **COM** mode establishes a "transparent mode" between the running target program and the controlling port of the ES1800. Whenever the ES1800 reads target memory during run mode, it actually stops emulation for about 100 microseconds. To avoid significant impact on real time operation, the **COM** routine examines the byte at *<address>* only once every 0.5 seconds. When the **COM** routine discovers a new byte from the target program, it reads the byte and clears the location. The byte is then sent to the controlling port of the ES1800. The **COM** routine then immediately returns to examine the byte at *<address>*. A target output routine has approximately 100 microseconds to place another character in the output location. If this 100 microsecond window is missed, the display of the subsequent character is delayed for 0.5 second.

The flow diagram on the next page summarizes the **COM** process.

(continued)

Figure 6. Flow Chart



Examples

One good use of the **COM** command is to simulate a serial I/O port when debugging code before target hardware is available. The **RUN** command downloads the target program into overlay memory and enters run mode. The address supplied to the **COM** command is that of a simulated RS232 data port. Data entered at the terminal is passed to the target program, and data output by the program appears on the screen.

```
 >MAP 0 TO -1                         /* Map all available overlay memory*/
 >DNL
%cat serial.driver                    /* Download program to overlay */
(enter transparent mode escape sequence)
 >RNV                                 /* Run program */
R>COM 'serial_port                    /* Use serial data port as COM addr */
```

A note of caution: if a breakpoint or an error is encountered while running the **COM** command, the system will appear to hang up. This is because emulation has been broken, and the target program that receives and transmits characters is no longer running. Entering the transparent mode escape sequence will terminate **COM** mode and cause the break or error message to be displayed.

DISPLAY CHARACTER STRING

Command	Result
DIA *<address>*	Reads and displays characters from target memory starting at the specified address. The **DIA** routine terminates when it reads [$00] from target memory. Affects real time operation when entered in run mode. See pages 6-18, 6-19, 6-31.

Comments

DIA is commonly used for test purposes in target systems that have no human-readable I/O channels.

When a test routine detects a problem, it can load a register with the address of a null terminated error message. The routine then jumps to an address that causes the Emulator to break emulation. The **DIA** command can then be used to display the error message.

DIA can also be used to check the contents of any null terminated string in memory.

Examples

```
>BYM                                   Make sure we're in byte mode.
>M 120                                 Enter Memory mode at address 120
$000120 $00  >48,65,6C,6C,6F,0
$000126 $00  >X                        Enter a null terminated string and exit
>DIA 120                               Display string starting at 120
Hello
>
```

This example sets a breakpoint in the target error routine. When the breakpoint occurs, a message pointed to by the ES:BX register pair is displayed. If the DX register is zero, the process stops. Otherwise, the ES1800 immediately begins emulation and waits for another breakpoint and message.

```
>AC1 = 'Error_stop
>WHE AC1 THE BRK
>* RBK;WAI;DIA ES:BX;TST = DX
```

(continued)

Overlay Memory

Overlay memory can be used to debug target hardware and software. It can be used to create and verify programs before hardware is available, determine whether the program is making illegal accesses, and patch target PROM code quickly and easily.

Overlay memory is available in memory ranges from 32K to 512K. See your Applied Microsystems Corporation sales representative for incremental options.

Overlay can be mapped in segments as small as 2K bytes. Each segment can be assigned one of four attributes; target, read/write, read only, or illegal. If memory is mapped, it means that you have assigned at least one segment of overlay as read/write, read only, or illegal memory. Unmapped memory is assigned the target attribute. Memory mapped as target or illegal does not use up overlay memory.

When a segment of memory is mapped, program accesses in that memory range are directed to the overlay instead of the target. The overlay can be further qualified by the overlay enable register (OVE). This register indicates whether code, data, or all accesses in a mapped memory range should be directed to the overlay memory.

Overlay memory accesses occur in real-time. No wait states are added by the Emulator at 10 MHz or below.

Command	Result
DM	Displays the memory map currently in effect.

Comments

This command is valid only in pause mode.

Examples

```
>DM
MEMORY MAP:
$000000 TO $FFFFFF:TGT
```

Default map at power-up.

DM displays regardless of mode (physical or virtual). See page 6-2 for more information.

(continued)

DISPLAY VIRTUAL MEMORY MAPPING LOG

Command	Result
DML	Displays the last 16 virtual memory mapping operations. If overlay memory is mapped using virtual addresses, the **DML** command shows the physical addresses that were mapped according to the descriptors that were valid at the time the memory mapping operation occured. The **DML** command allows the user to display the last 16 virtual memory mapping operations and includes the virtual address entered and the address of the physical memory that was mapped. If a protection error occurs while mapping, the mapping operation is not logged. If more than 16 operations occur, the oldest entry in the log is overwritten. This is a history of the virtual memory operations and does not display the *current* memory map if physical mapping operations have occured. Please use the **DM** command (display map) for this information.

If a modification is made to a descriptor table entry that was previously used in a mapping operation, the memory map is not altered to reflect this change.

If a range is used with virtual addresses, each virtual address is converted to a physical address. The first is used as the start point and the second as the end point. Adjustments are made for 2K boundaries. If a single virtual address is used, it is converted to a physical address and when mapped, is adjusted for a 2K boundary.

Examples

The following is an example of a virtual mapping log:

```
>DML
VIRTUAL MAPPING LOG:
ENTERED - MAP 0005:0000 TO 0005:2000    MAPPED - 000000 TO 0007FF : RO
ENTERED - MAP 0004:0000 TO 0004:1000    MAPPED - 000800 TO 001FFF : RW
ENTERED - MAP 000A:1000 TO 000C:0000    MAPPED - 010000 TO 012FFF : TGT
ENTERED - MAP 0108:0400 TO 0560:0040    MAPPED - 004000 TO 0043FF : ILG
```

SET MEMORY MAP

Command	Result
MAP *<range>*	Maps the specified range and assigns it the default attribute type, RW.
MAP *<value>*	Maps a 2K-byte block surrounding the specified value. Assigns the block the default attribute type, RW.
MAP *<range><attribute>*	Maps the specified range and assigns it the specified attribute type.
MAP *<value><attribute>*	Maps a 2K-byte block surrounding the specified value. Assigns the block the specified attribute.

Attributes

RW — Memory mapped as read-write (RW) responds like normal overlay memory. The overlay memory is high speed and may actually run faster than target system memory if that memory normally asserts wait states.

RW is the most common attribute and is therefore the default. MAP commands that do not specify an attribute default to RW partitions.

RO — Memory mapped as RO acts like read-only memory to the target program. If the program attempts to write to this memory, the ES1800 aborts run mode and displays the error message, MEMORY WRITE VIOLATION. The contents of RO overlay cannot be altered by a running target program.

The same comments about speed given in the paragraph on RW apply to memory mapped as RO. You can always modify memory mapped as RO (in pause mode) even though the target program (run mode) cannot.

ILG — Memory mapped as illegal can be used to mark address ranges that should not be accessed by the target program. Any access to an address range mapped as ILG causes the ES1800 to abort run mode and display the error message, MEMORY ACCESS VIOLATION. Memory mapped as ILG does not use up available overlay memory.

TGT — The ES1800 ignores accesses in address ranges mapped with this attribute. Memory that is not explicitly mapped is defaulted to TGT.

Comments

Overlay memory is mapped in segments of 2K bytes. When you specify an address or a range to be mapped as RW or RO, the mapping outline allocates the minimum number of 2K segments that will completely enclose the address(es) of interest.

There is a distinction between the overlay map and overlay memory. If your system has any overlay memory installed (it is an option), you have a complete overlay map and some limited amount of overlay memory. The overlay map covers the entire address space (24 bits). The overlay map is used to logically place segments of overlay memory anywhere throughout the address space.

(continued)

You can save and restore the contents of the overlay map by using the EEPROM **LD/SAV** commands (see pages 5-34 and 5-36). You cannot save the contents of overlay memory in EEPROM.

Examples

The following command sequence might reflect a common mapping:

```
Command                          Comments
---------------------------      -----------------------------------
 >CLM                            Clear map to all TGT
 >MAP 0 TO -1:ILG                Default entire address space to Illegal
 >LDV                            Sets CS:IP to 0FFFF0 (reset vector)
 >MAP CS:IP:RO                   Map ROM for reset vectors
 >MAP 'RAM_start LEN 20000       Map some overlay memory to work with
 >MAP 'I/O_start:TGT             Have I/O already in target space
 >MAP 0 LEN 800                  Allocate RAM for interrupt vectors
 >DM                             Display what we've done
MEMORY MAP:
MAP $000000 TO $0007FF:RW        Interrupt Vectors
MAP $000800 TO $00FFFF:ILG
MAP $010000 TO $02FFFF:RW        Working RAM
MAP $030000 TO $03FFFF:ILG
MAP $040000 TO $0407FF:TGT       I/O space
MAP $040800 TO $0FF7FF:ILG
MAP $0FF800 TO $0FFFFF:RO        Reset vectors
MAP $100000 TO $FFFFFF:ILG
 >
```

Since the contents of overlay memory are not affected by changing the overlay map, you can compare the operation of a program in target memory with one in overlay memory.

Command	Comments
>CLM	Clear any previous mapping:
>MAP 1000 to 7FFF:RO	Map ROM over existing target program
>LOV 1000 to 7FFF	Copy target program into overlay memory
>ASM 2000	Use line assembler to make a patch
(Assembler commands)	
>RNV	Run patched version
>STP;MAP 1000 TO 7FFF:RO;RVN	Stop, Remove map, Run normal version
>STP;MAP 1000 to 7FFF:RO;RNV	Stop, Restore map, Run patched version

If you do not have target memory but you still want to compare two programs, you can use a trick of overlay memory allocation. This example assumes you have 128K or more of overlay memory.

Command	Comments
>CLM;MAP OFFFF0:RO	Need Reset Vector mapped as ROM
>GRO = 1000 LEN 8000	Will save some typing
>MAP GRO	Map 32K bytes for code space
>DNL	Download first program into overlay
(Download commands and records)	
>MAP GRO:TGT	Unmap code space (The data is still in overlay memory)
>MAP GRO + 10000	Remap but at higher address range. The first program now "exists" again but in a higher address range.
>MAP GRO	Now map more overlay at the normal range
>DNL	Download second program.
(Download commands and records)	
	Now you have a copy of both programs.
>MAP GRO:TGT;MAP GRO + 20000	Relocates second program out of the way
>MAP GRO +10000:TGT;MAP GRO	Relocates first program back to normal address range.

CLEAR MEMORY MAP

Command	Result
CLM	The entire address range is assigned the TGT attribute.

Comments

This command clears all addresses from the overlay map.

This command is valid only in pause mode.

LOAD OVERLAY MEMORY

Command	Result
LOV *<range>*	Moves data from the target system memory to the Emulator overlay memory in the specified address range.

Comments

This command is valid only in pause mode.

In order to load overlay memory from the target memory, you must have a target system interfaced with the ES1800 Emulator and have overlay memory installed and mapped.

In order to load a target memory range into the overlay memory at a different address, use the **LOV** command, then do a block move (**BMO**) of the data.

VERIFY OVERLAY MEMORY

Command	Result
VFO *<range>*	Compare the specified range in the target memory to the same range in the overlay memory. If there are no differences between the data in the overlay and target, the Emulator prompts you for the next command. If there are any differences, the address of each difference displays: `<ADDRESS> = XX NOT YY` *XX* denotes the data present in overlay memory. *YY* is the data at that location in the target system memory.

Comments

This command is valid only in pause mode.

SET OVERLAY SPEED

Command	Result
OVS *<value>*	The **OVS** command allows the user to set the number of wait states to be inserted in each bus cycle when accessing overlay memory. The RDY switch must be ON. **OVS** accepts values of $0 to $0E.

(continued)

Registers

The following is a complete list of all the registers in the Emulator. These registers can be logically divided into three groups:

- microprocessor registers
- general Emulator registers
- Event Monitor System registers

Each register accepts one or two of three value types:

- integer values
- range values
- don't care values

Registers that accept range and don't care types can also be assigned integer values.

Microprocessor Registers

Name	Description	Type	Length (bits)
AX, AL, AH	accumulator (low and high)	Int	16,8,8
BP	base pointer	Int	16
BX, BL, BH	base register (low and high)	Int	16,8,8
CS	code segment	Int	16
CSA	code segment access rights	Int	8
CSB	code segment base	Int	24
CSL	code segment limit	Int	16
CX, CL, CH	count register (low and high)	Int	16,8,8
DI	destination index	Int	16
DS	data segment	Int	16
DSA	data segment access rights	Int	8
DSB	data segment base	Int	24
DSL	data segment limit	Int	16
DX, DL, DH	data (low and high)	Int	16,8,8
ES	extra segment	Int	16
ESA	extra segment access	Int	16
ESB	extra segment base	Int	24
ESL	extra segment limit	Int	16
FLX	flags register	Int	16,8,8
GDB	global descriptor table base	Int	24
GDL	global descriptor table limit	Int	16
GDTR	= GDB + GDL	Int	16
IDA	interrupt descriptor table access rights	Int	8
IDB	interrupt descriptor table base	Int	24
IDL	interrupt descriptor table limit	Int	16
IP	instruction pointer	Int	16
LDA	local descriptor table access rights	Int	8
LDB	local descriptor table base	Int	24
LDL	local descriptor table limit	Int	16
LDT	local descriptor table register	Int	16
LDTR	local descriptor table register = LDA + LDB + LDL	Int	16

(continued)

Microprocessor Registers (cont.)

Name	Description	Type	Length (bits)
MSW	machine status word	Int	16
SI	source index	Int	16
SP	stack pointer	Int	16
SS	stack segment	Int	16
SSA	stack segment access rights	Int	8
SSB	stack segment base	Int	24
SSL	stack segment limit	Int	16
TA	task access rights	Int	8
TB	task base	Int	24
TL	task limit	Int	16
TR	task register	Int	16

General Emulator Registers

Name	Description	Type	Length (bits)
ADF	force address register	Int	4
DFB	default base	Int	8
GD0-GD7	general purpose data	DC	32
GR0-GR7	general purpose range	Range	32
IOP	I/O mode pointer	Int	16
MMP	memory mode pointer	Int	32

Event Monitor System Registers

Name	Description	Type	Length (bits)
AC1.1-AC1.4	address comparator	Range	24
AC2.1-AC2.4	address comparator	Range	24
CTL.1-CTL.4	count limit comparator	Int	16
DC1.1-DC1.4	data comparator	DC	16
DC2.1-DC2.4	data comparator	DC	16
LSA.1-LSA.4	logic state comparator	DC	16
S1.1-S1.4	status comparator	DC	16
S2.1-S2.4	status comparator	DC	16
SIA	special interrupt address (real mode)	Int	32
SIA	special interrupt address selector: offset (virtual mode)	Int	32

Each register has a separate display base. The display base is viewed and changed with the **BAS** command (see page 5-83). Display bases are often changed for registers such as the Event Monitor LSA comparators, which you might like to see in binary, and the CTL register, which you might want to see in decimal.

The CPU registers and the Event Monitor registers can be displayed as a group by using the **DR** and **DES n** commands.

See Event Monitor System (Section 7) for Event Monitor System Register descriptions.

The complete register set can be loaded from or saved to EEPROM. Executing a **SAV** or **LD** copies all system variables. A **SAV 1** or **LD 1** copies only the register group.

DISPLAY/LOAD MICROPROCESSOR REGISTERS

Command	Result
DR	Displays values of all microprocessor registers.

```
 >DR
CS:IP      FLX          AX   BX   CX   DX   DS   SI   ES   DI   BP   SS   SP
F000:FFF0  .........    0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
```

<register name>	Displays the value of the specified microprocessor register in its display base.
<register name>=<exp>	Assigns the specified register the value *<exp>*.
CLR	Clears the four CPU data registers; AX, BX CX, and DX.
LDV	Loads the reset vectors into the CS, IP and FLX registers. The reset vectors can also be loaded by the **RNV** and **RBV** commands. These load the vectors and enter run mode (page 6-21).

Comments

On power-up an **LDV** command is automatically executed. This command sets the registers to Intel-defined default values. Register values may be saved to and loaded from EEPROM.

The CPU registers are automatically copied from Emulator overlay memory to the microprocessor when run mode is entered. When emulation is broken, they are copied from the processor to Emulator overlay memory.

If a CPU register is loaded with a value during run mode, a warning message is to be displayed. This warning informs you that the value you are entering will not be sent to the pod CPU during emulation. The value is stored in the Emulator's internal memory, but when emulation is broken, the new value of the CPU register overwrites the value just entered.

The display of the FLX register is different from that of the other CPU registers. The flags are more conveniently decoded by using an alpha character to indicate whether the flag was set or cleared by a particular instruction cycle. If the flag is clear, you see a □ as a place holder. If set, the following characters describe the flag.

```
N - Nested task
O - Overflow
D - Direction
I - Interrupt
T - Trap
S - Sign
Z - Zero
A - Auxillary carry
P - Parity
C - Carry
```

16 8 0
x x x x | O D I T | S Z x A | x P x C |

If FLX were assigned the value $FFFF, the **DR** command would display the FLX register as:

```
>DR
CS:IP      FLX         AX   BX   CX   DX   DS   SI   ES   DI   BP   SS   SP
0000:0000  NODITSZAPC  0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
```

(continued)

Examples

Load the data segment and verify that it contains the correct value.

```
 >DS=$A700;DS
$A700
 >
```

Command	Result
DRL	Displays values of all memory management registers as well as all microprocessor registers.

Examples

```
>DRL
 IP    AX    BX    CX    DX    SI    DI    BP    SP
____  ____  ____  ____  ____  ____  ____  ____  ____
FFF0  0000  0000  0000  0000  0000  0000  0000  0000

FLX = ..........       IOP = 0      MSW = ../../../..

REG    INDEX   TI   RPL    BASE   LIMIT    ACCESS       DPL
____________________________________________________________

CS      1E00   G     0    FF0000  FFFF    P C D U W .    0
DS      0000   G     0    000000  FFFF    P C D U W .    0
ES      0000   G     0    000000  FFFF    P C D U W .    0
SS      0000   G     0    000000  FFFF    P C D U W .    0
GDTR    ....   .     .    000000  0000    . . . . . .    .
LDTR    0000   G     0    000000  0000    . C D U . .    0
IDTR    ....   .     .    000000  FFFF    . . . . . .    .
TR      0000   G     0    000000  0000    . C D U . .    0
```

(continued)

Comments

The memory management unit translates logical addresses to physical addresses, based upon information found in the descriptor table. The CS, DS, ES, SS, GDTR, LDTR, IDTR, and TR show the contents of the most recently used descriptor table entry for each appropriate group.

SET/DISPLAY REGISTER DEFAULT BASE

Command	Result
BAS *<register>*	Displays the decimal base of the specified register.

```
#0     - default
#2     - binary
#8     - octal
#10    - decimal
#16    - hexadecimal
```

If the register has not been assigned a separate display base, the current default base is displayed.

BAS *<register>= <base value>*	Sets the display base of the register to the base value. If the base value for a register is set to 0, the current default base is used for display.

Comments

Base values may be stored in EEPROM and automatically loaded on power-up or manually retrieved using the **LD** or **LD 1** command.

Be careful when setting private display bases to unusual bases such as 4, 7, or 11. The Emulator operates correctly, but the results may be confusing. If you set the base value to a value other than hexadecimal, decimal, octal,

(continued)

or binary, the Emulator displays a question mark ([?]) preceding the base value when asked to display the base in effect.

Refer to the default base command, **DFB** (page 5-88), to display the system global default base.

Examples

```
>BAS FLX
>#16
```

The value of GD3 is displayed in binary until you change its display base or power down the Emulator.

```
>GD3
$0000AA55
>BAS GD3 = 2
>BAS GD3
#2
>GD3
%00000000000000001010101001010101
```

MEMORY MODE POINTER

Command	Result
MMP	Displays the current value of the memory mode pointer.
MMP = *<exp>*	Assigns the value *<exp>* to the memory mode pointer.

Comments

The MMP is the last address examined while in memory mode. If you enter memory mode without specifying an address, the MMP value is used as the entry point.

The default power-up value of the MMP register is zero. This register may be saved to and loaded from EEPROM.

The memory mode pointer is automatically modified when you scroll to a new address after entering memory mode. When you exit memory mode, the MMP reflects the last address examined. For more information on memory mode, see page 6-52.

(continued)

Examples

Sets an address comparator to the last address examined in memory mode.

```
 >M 6000

   (examine memory until you find a location of interest)

$006013 5A >X
 >AC1=MMP
```

I/O MODE POINTER

Command	Result
IOP	Displays the current value of the I/O mode pointer.
IOP = *<exp>*	Assigns the value *<exp>* to the I/O mode pointer.

Comments

The IOP is the last value examined while in I/O mode. If you enter I/O mode without specifying an address, the IOP value is used as the entry point.

The default power-up value of the IOP register is zero. This register may be saved to and loaded from EEPROM.

The I/O mode pointer is automatically modified when you scroll to a new address after entering I/O mode. When you exit I/O mode, the IOP reflects the last address examined. For more information on I/O mode, see page 6-52.

DEFAULT BASE

Command	Result
DFB	Displays the global default base. On power-up the default base is hexadecimal unless another default base was loaded by the EEPROM on power-up.
DFB = #2	Sets the default base to binary.
DFB = #8	Sets the default base to octal.
DFB = #10	Sets the default base to decimal.
DFB = #16	Sets the default base to hexadecimal.

Comments

Specific operators determine the base of the input value:

Operator	Description	Example
<%>	Binary	%10011100001111
<\>	Octal	\23417
<#>	Decimal	#9999
<$>	Hexadecimal	$270F

Base prefixes can be used any time to enter a value in a base different from the default base. Values not preceded by one of these prefixes are presumed by the Emulator to be in the default base.

For example, if you set the global default base to binary, and you then want to assign a value to a register in a base other than binary, use a base prefix.

The Emulator works correctly with any base between 2 and 16. However, if you set an uncommon base, such as 5 or 9, the results of assignments and commands may be confusing.

If the base is outside the allowable range, an error message is displayed and the Emulator defaults to the hexadecimal base.

GENERAL PURPOSE DATA REGISTERS

Command	Result
GD*<0-7>*	Displays the value of the specified register.
GD*<0-7>* = *<value>*	Assigns a value to one of the eight general purpose data registers.

Comments

Use the general purpose registers as arguments to commands to save keystrokes when using values repeatedly. They can also be used to save space in macro definitions.

These general purpose registers may be used in place of integer or don't care values in command statements.

The general purpose data registers can be loaded with any integer or don't care value. They will not accept a range value.

Examples

General purpose data register 4 is loaded with 5000. GD4 can now be used anywhere you would use this integer value.

```
>GD4 = 5000
```

If you are looking for a specific pattern on the LSA pod lines in more than one event group, assign a general purpose data register the value you are looking for. All subsequent LSA assignments can use this register.

```
>GD2 = %01100101100 DC % 10011
>LSA = GD2; LSA.2 = GD2
>GD3 = 'datpat1 DC %FF00              Looking for one byte
>DC1 = GD3                            of a specified word?
```

If you have a hard time remembering the memory mode status mnemonics, use a general purpose register instead.

```
>GD6 = ALT
>MMS = GD6
>GD1 = OVL+RD+IOA                     To set-up a breakpoint on an overlay
>S1 = GD1                             read from I/O space.
```

GENERAL PURPOSE ADDRESS REGISTERS

Command	Result
GR*<0-7>*	Displays the value of the specified register.
GR*<0-7>* = *<value>*	Assigns a value to one of the eight general purpose address registers.

Comments

Use the general purpose registers as arguments to commands to save keystrokes when using values repeatedly. They can also be used to save space in macro definitions.

These general purpose registers may be used in place of integer or range values in command statements.

The general purpose data registers can be loaded with any integer or range value.

Examples

General purpose address register 4 is loaded with 5000. GR4 can now be used wherever you would use this integer value.

```
>GR4 = 5000
```

Assign a register a range you will be using often. Then use it as a parameter for other commands.

```
>GR0 = 'start_code LEN 20
>DIS GR0
>DB GR0
```

If you do not know the absolute address in your target hardware, but have downloaded a symbol table containing them, then use the symbol names instead of looking up the hardware specifications.

```
>GR2 = 'RAM_START LEN 8000      Initialize GR2
>SF 0,GR2                       Run a test on your overlay memory
>AC1 = GR2                      Set a breakpoint on any overlay
>WHE AC1 THE BRK                   memory access in the range of GR2
```

TEST REGISTER

Command	Result
TST	Stops repeating commands. The text register is set to an expression in a command line. When it becomes zero, the repeat halts.

Comments

See Repeat Operators (page 5-110) for more detailed information.

Trace Memory

During emulation, the activity of the executing program is recorded and stored in trace memory. All address lines, data lines, processor status lines, and 16 bits of external logic-state are traced. This record becomes a history of the program. If something unexpected happens during program execution, trace memory can be reviewed to determine what exactly took place. When used in conjunction with the trace disassembler, hardware and software problems may be found.

Although you cannot access trace memory during emulation, you can stop program execution at any point, either manually, or by using the Event Monitor System. The address, data, and control signals of the most recently traced cycles may be reviewed.

Trace memory commands deal with the display and disassembly of trace memory data. Refer to the Event Monitor System (Section 7) for sophisticated uses of trace memory.

Trace memory is 71 bits wide and 2046 bus cycles deep. Some bus cycles may be used for marks to identify start and stop points within the trace buffer. An unqualified trace contains all bus activity for the last 2046 bus cycles.

NOTE: The sequence numbers in **DT**, **DTB**, and **DTF** (instructions) correlate with the line numbers displayed in the **DRT** (bus cycles). However, one or more bus cycles in the **DRT** display may make up one instruction on the **DT**, **DTB** or **DTF** displays. These displays may have missing sequence numbers indicating that a multiple bus cycle instruction has been executed. Also, the sequence number (SEQ #) may be repeated when two-byte wide instructions were executed from contiguous addresses.

DISPLAY RAW TRACE BUS CYCLES

Command	Result
DRT	Displays the last page of bus cycles recorded in trace memory.
DRT *<line number>*	Displays a page of the trace buffer starting with *<line number>*.
DRT *<range>*	Displays the range of line numbers. XON and XOFF may be used to start and stop scrolling if the range is larger than the console display. *Note that the range is a range of bus cycles, not the address recorded in the trace memory.*

Comments

SET parameter #13 sets the page length. Refer to **SET** (page 5-3).

This command is valid only in pause mode.

Examples

```
 >RUN
R>STP
 CS:IP = $000334
GROUP 1
 >DRT
LINE  ADDRESS   DATA  R/W  MEM  M/IO   BCYC  EA QFL  LSA  -  8  7  -  0
-------------------------------------------------------------------------
 #20  00033A  > 23FF   R   TAR   M      IF    8       %11111111 %11111111
 #19  00033C  > 7400   R   TAR   M      IF    8       %11111111 %11111111
 #18  000333  > 4B     R   TAR   M      IF    9  QF   %11111111 %11111111
 #17  000334  > FD75   R   TAR   M      IF    0  QF   %11111111 %11111111
 #16  000336  > F9E2   R   TAR   M      IF    4       %11111111 %11111111
 #15  000338  > 3E80   R   TAR   M      IF    4       %11111111 %11111111
 #14  00033A  > 23FF   R   TAR   M      IF    8       %11111111 %11111111
 #13  00033C  > 7400   R   TAR   M      IF    8       %11111111 %11111111
 #12  000333  > 4B     R   TAR   M      IF    9  QF   %11111111 %11111111
 #11  000334  > FD75   R   TAR   M      IF    0  QF   %11111111 %11111111
 #10  000336  > F9E2   R   TAR   M      IF    4       %11111111 %11111111
 # 9  000338  > 3E80   R   TAR   M      IF    4       %11111111 %11111111
 # 8  00033A  > 23DF   R   TAR   M      IF    8       %11111111 %11111111
 # 7  00033C  > 7400   R   TAR   M      IF    8       %11111111 %11111111
 # 6  000333  > 4B     R   TAR   M      IF    9  QF   %11111111 %11111111
 # 5  000334  > FD75   R   TAR   M      IF    0  QF   %11111111 %11111111
 # 4  000336  > F9E2   R   TAR   M      IF    4       %11111111 %11111111
 # 3  000338  > 3E80   R   TAR   M      IF    4       %11111111 %11111111
 # 2  00033A  > 23FF   R   TAR   M      IF    4       %11111111 %11111111
 # 1  00033C  > 7400   R   TAR   M      IF    4       %11111111 %11111111
 # 0  BREAK
```

LINE Line number 0 in the trace buffer indicates the last bus cycle prefetched or executed before the Emulator went into pause mode. The larger the line number, the further back in the history of the program you are viewing. You can get a good idea of the relationship of bus cycles to instructions by matching the bus cycle line numbers in the DRT to the SEQ# in the disassembled trace.

(continued)

ADDRESS The memory address or location where the instruction was fetched. The addresses in the raw trace represent physical addresses.

DATA The address displayed is where the bus cycle took place, along with the data written to or read from that address. The addresses in the raw trace (**DRT**) represent physical addresses.

[>] and [<] are data direction indicators. They indicate whether data was read from an address ([>]) or written to an address ([<]). These same indicators are used in the trace disassembly.

MEM MEM indicates whether the access was in the target memory area or in the Emulator's overlay memory (see **DM** command to determine what addresses are mapped).

M/IO M/IO indicates whether the bus cycle access was a memory access (M) or an I/O access (IO). This is determined by the program.

BCYC BCYC indicates what type of bus cycle was run. This is determined by your program. The possibilities are:

HSD	halt shutdown
IAK	interrupt acknowledge
IF	instruction fetch
LK	locked bus cycle
NBC	no bus cycles
PE	processor extension cycle
RIO	read from I/O
RM	read memory
WIO	write to I/O
WM	write memory

EA Indicates the four least significant bits of those instructions that were executed. Not all addresses of instructions executed are traced, because of the asynchronous nature of executed address vs. prefetched information. Information is stored in trace at the end of each bus cycle. Several instructions can be executed during each bus cycle. Only the last instruction executed before the end of the bus cycle is executed.

R/W R/W indicates whether the bus cycle access was a read or write cycle. This is determined by the program.

QUE QUE indicates how many bytes (up to 6) are in the processor queue or how many were "flushed" (usually caused by a branch). A flush is indicated by a [Q] preceding the queue depth value.

LSA-8 7-0 LSA-8 7-0 columns display the state of each pin of the LSA pod during that bus cycle.

NOTE: The same information that is recorded in the trace buffer can be used by the Event Monitor System to cause event actions. Therefore, everything in the trace buffer such as QUE flushes or WIO or any combination of these traced items can cause event actions such as selective tracing, counting, or breaking emulation (refer to the Event Monitor System, Section 7).

DISASSEMBLE TRACE MEMORY

Command	Result
DT	Disassembles and displays the last instruction in trace memory. A sequence number is not included. Overwrites current display line.
DT *<range>*	Disassembles a range of bus cycles, starting at the specified value and proceeding back in time.
DT *<value>*	Disassembles a page of trace starting at *<value>*.

Comments

This command is valid only in pause mode.

A page is defined by the CRT length parameter in the **SET** menu.

The sequence #0 is always the most recently recorded bus cycle in trace memory. If an argument is specified to the **DT** command, the values refer to the raw trace sequence numbers.

The sequence number shown is a decimal value. For numbers larger than 9, precede with a decimal ([#]) base sign.

When using the disassemble trace (**DT**) and the display register (**DR**) on the same line, make sure you enter **DT** before **DR**, because **DT** will overwrite the current line. It does this so that the **STP;DT** command used repeatedly will give a listing similar to a program listing without the **STP;DT** line between each command.

Examples

These two commands used in conjunction will produce output similar to a program listing.

```
>STP;DT
```

(continued)

```
>DTB
PARTIAL TRACE MEMORY MAP

SEQ#  ADDR    LOG ADDR   DATA        MNEMONIC  OPERANDS  BUS CYCLE DATA

0084  00086B  0002:0098  C745020000  MOV   WORD PTR [DI+2],0000
                                           0003:0082<0000
0080  000870  0002:00A0  C745040292  MOV   WORD PTR [DI+4],9202
                                           0003:0084<0292
0071  000875  0002:00A5  C745060000  MOV   WORD PTR [DI+6],0000
                                           0003:0086<0000
0067  00087A  0002:00AA  0F01E0      SMSW  AX
0065  00087D  0002:00AD  0D0100      OR    AX,0001
0057  000880  0002:00B0  0F01F0      LMSW  AX
0056  000883  0002:00B3  EB01        JMP   SHORT 002182
0052  000886  0002:00B6  90          NOP
0052  000887  0002:00B7  EA8C080800  JMP   FAR PTR 0008:088C  0003:006C>009E
                                           0003:0068>0010  0003:006A>0000
                                           0003:006C<009F
0037  00088C  0002:00BC  B81800      MOV   AX,0018
0036  00088F  0002:00BF  8ED0        MOV   SS,AX  0003:007C>0092
                                           0003:0078>0080  0003:007A>0000
                                           0003:007C<0093
0035  000891  0002:00C1  B81000      MOV   AX,0010
0033  000894  0002:00C4  8ED8        MOV   DS,AX 0003:0074>0092
                                           0003:0070>0010  0003:0072>0020
                                           0003:0074<0093
0032  000896  0002:00C6  90          NOP
0032  000897  0002:00C7  90          NOP
0021  000898  0002:00C8  90          NOP
0021  000899  0002:00C9  90          NOP
0020  00089A  0002:00CA  90          NOP
0020  00089B  0002:00CB  90          NOP
0019  00089C  0002:00CC  90          NOP
0019  00089D  0002:00CD  90          NOP
>
```

SEQ# Correlates the disassembled instruction to the raw trace bus cycle.

This is a decimal number and must be preceded by a [#] sign when referenced for selective disassembling of the trace. This corresponds to the line number in the **DRT** command display.

ADDR The memory address or location where the instruction was fetched.

OPCODE The machine-language (hex number) equivalent of the following assembly-language instruction.

MNEMONIC The command used to invoke the instruction.

OPERAND FIELD The assembly-language instruction.

BUS CYCLE DATA The bus cycle transaction, if any, that occurred as a result of the instruction. This includes any information written to, or read from, memory or I/O locations.

DISASSEMBLE TRACE PAGE

Command	Result
DTB	Disassembles the previous page of trace memory (from current trace memory pointer).
DTF	Disassembles the following page of trace memory (from the current trace memory pointer).

Comments

This command is valid only in pause mode.

A page is defined by the **CRT** length parameter in the **SET** menu. Three lines are subtracted for header and prompt lines.

Refer also to the **DT** command, page 5-100, the **DRT** command, page 5-96, and the slash command, page 5-113.

Macros

A macro defines a list of commands or expressions that are executed with one command key word. This allows you to execute repetitive operations quickly and easily.

You can define up to ten macros. They are referred to by the decimal numbers #0-9. The ten macros are linked in one buffer with #1 first, #2...#9, and #0 last.

If the lengths of all ten macros exceeds the buffer length of 125 characters, the highest numbered macro is truncated. Spaces are also considered characters, so use them only when required, to save macro buffer space.

Examples

If macros #1 to #8 are defined and in this process use up all of the space in the buffer, then an attempt to define macro #9 and #0 results in those macros remaining null. Also, if the length of any macro from #1 to #7 is increased after filling the buffer, then macro #8 will be truncated. If the increase is more than the size of macro #8, macro #8 becomes null and macro #7 is truncated.

WARNING!

There are no warnings when truncation or nullification takes place.

When you define a number of long macros, execute the **MAC** command to determine if the macros of the highest numbers are still intact. Using the general purpose registers in macros helps minimize the number of characters you need to use.

Macros can be saved in the Emulator EEPROM. Refer to the **LD** and SAV (pages 5-34 and 5-36) commands for information on saving and reloading macros.

DISPLAY DEFINED MACROS

Command	Result
MAC	Displays all defined macros in order #1-9,0 identified by three character sequences.

Examples

```
  >_1=DR;DIS CS:IP LEN 4; RUN
  >_2=DB; SS:SP LEN 10;@'Data_ptr
  >MAC
_1=DR;DIS CS:IP LEN 4; RUN
_2=DB; SS:SP LEN 10;@'Data_ptr
  >
```

Command	Result
_*<0-9>= <com, exp, op>*	Defines the specified macro.
_*<0-9>*	Executes the specified macro.

Comments

A space between the underscore, digit, or equals sign causes an error.

There are shorthand notations for two macros: a comma as the first character on a line executes macro #1 and a period as the first character on a line executes macro #2.

Examples

Three macros are defined. Macros #1 and #2 can be executed independently. Macro #3 contains two nested macros (#1 and #2).

Macros are not expanded when the macro is defined, so the definition of macro #3 may change, depending upon the content of macros #1 and #2.

In this example, macro #2 uses a general purpose register as a counter.

```
>_1=STP;DT
>_2=GD1=GD1+1
>_3=_1;_2
```

(continued)

Step and disassemble one instruction at a time.

```
>_1= DB SS:SP LEN 20;RET;DIS CS:IP LEN 12
```

Display the first 20H bytes on the stack, skip a line for readability and disassemble the next instructions that will be executed.

There is no display on the screen and no syntax checking when a macro is defined. Errors are detected only when the macro is executed.

Macro #3 is executed.

```
>_3
```

CLEAR MACROS

Command	Result
CMC	Clears all defined macros.
_*<0-9>=*	Clears the specified macro.

Examples

Clear macro #1.

```
>_1=
```

(continued)

The Repeat Operators

The command repeat feature provides a way to repeat a command line a specified number of times or indefinitely. A repeat is indicated by an asterisk ([*]) at the beginning of a command line. The asterisk is followed by an optional decimal argument to specify the number of times to repeat the buffer contents. If the argument is zero, the buffer content is not executed.

Examples

```
>*5STP;DT
>*5 STP:DT
>* 5 STP;DT
```

In these three equivalent examples, the **STP;DT** command is repeated five times. If the slash key is typed after the above example is input, the entire line is repeated, causing five more **STP;DT** commands to be executed.

The repeat argument must be specified in decimal, not in hex, or as a variable, and there must be a space following the repeat count if the next character is a decimal digit.

When the repeat argument is not specified it is assumed to be 4,294,967,295 ($2^{32}-1$). A repeat can always be terminated by executing a system reset. However, this will also abort emulation, if it is in progress, without saving the state of the CPU.

The TST register terminates a repeats by setting it to zero with an expression in the command line. It is tested just before the command line is executed and if it has become zero, the command buffer is not executed and the repeat halts.

To single step and disassemble until a specified address is reached:

```
>*STP;DT; TST=CS:IP-$C324
```

If you are waiting for an overlay memory location to be cleared:

```
>*STP;DT;TST=@87020
```

You can use the system reset character to stop the repeat if the specified test conditions are never reached.

The TST register is set to all 1s at the start of a repeat. This is necessary so that the register is in a known state at the start of a repeat loop.

Repeats can also be terminated by the states of the limit (LIM) and index (IDX) registers. Just before execution begins, the values of LIM and IDX are compared. If IDX is greater than or equal to LIM, the repeat is terminated. The LIM register is initialized to the number of times the loop will execute, which is the decimal loop count you specified in the command line.

IDX is a counter. It starts at zero and is incremented every time the repeat loop is executed. You may assign new values to these registers within repeat command lines if you wish.

For example, if you need a decimal counter:

```
 >BAS IDX=#10
 >*#3 IDX
#0
#1
#3
```

(continued)

Initialize a block of memory to a decrementing count ending in zero, then display it.

```
 >BYM; M $1000
$001000 $34  >*4 LIM-IDX-1
$001001 $C0
$001002 $BF
$001003 $00
$001004 $21  >M MMP-4
$001000 $03  >*4
$001001 $02
$001002 $01
$001003 $00
$001004 $21  >
```

REPEAT COMMAND LINE

Command	Result
/	Re-executes the previous command line. No [RETURN] is necessary.

Comments

In order to be recognized as the repeat character, the slash must be the first character on a line.

Examples

This causes the system to single step and disassemble the instruction just executed.

```
>STP;DT
>/
>/
>/
>/
```

This causes the system to single step and disassemble memory starting at the instruction pointer (IP) location.

```
>STP;DIS CS:IP LEN 10
>/
```

(continued)

Symbols

Symbol definitions allow you to refer to addresses and data values using names rather than numbers. Symbols are 32-bit integer values and sections are 32-bit ranges. Symbols and sections are sometimes collectively referred to as symbols.

64K bytes of overlay memory are allocated for symbol definitions. To determine approximately how many symbols you can define, take the average symbol name length, add six and divide into 64K (64 x 1024).

Symbols are not typed within the Emulator, so all symbols are global. This implies that a symbol and a section may not be defined using the same name. A symbol name may only be defined once. Section range values may not overlap.

Symbols may be redefined by assigning a new value to the symbol name. If you want to reassign a symbol name to a section value, or if you want to change the range value of a section, you need to delete the symbol or section name before assigning the new value.

Most compilers and assemblers create symbol tables from the symbols defined in the program. These symbols can be easily downloaded if you have a linker and converter that can create Extended Tekhex serial data records. See the **SET** command (page 5-3) for the serial data format variable. If you are going to download sections that have already been defined (perhaps from a previous download of the same file), purge all symbols or delete the section definitions from memory before downloading. If you do not, an error occurs when you attempt to redefine the value of a section, and the download aborts.

Symbols may be used as parameters to any ESL commands. The only limitation on symbols is that they cannot be used meaningfully with the colon operator ([:]). The single line assembler accepts symbols as address references and data values.

Memory and trace disassembly display symbol names in place of absolute values for address fields. The following examples illustrate the difference when the same program is disassembled with and without symbol definitions.

Examples

First, the symbols are defined.

```
 >SYM
$00000480 csr
$00000486 sh_csr
$00001000 CMND
$00001022 Tauc
$00000004 busy
$00000002 got_it
$00000080 action
$00004020 es10
 >SEC
$000010000 TO $0000104F monitor
```

The following example shows memory disassembly with symbol definitions.

```
 >GRO=1000 LEN 2A
 >DIS GRO
CMND
1000  F70680048000 TEST     WORD PTR csr,0080
1006  74F8         JE       SHORT CMND
1008  C606800402   MOV      BYTE PTR csr,02
100D  C606860402   MOV      BYTE PTR sh_csr,02
1012  A02040       MOV      AL,BYTE PTR es10
1015  800E860404   OR       BYTE PTR sh_csr,04
101A  8A268604     MOV      AH,BYTE PTR sh_csr
101E  88268004     MOV      BYTE PTR csr,AH
Tauc
1022  F70680048000 TEST     WORD PTR csr,0080
1028  75F8         JNE      SHORT Tauc
```

(continued)

The following example shows trace disassembly with symbol definitions.

```
 >DTB
 >PARTIAL T.M. MAP:  PASS 1  PASS 2
FULL T.M. MAP:  PASS 1  PASS 2
SEQ#  ADDR   OPCODE MNEMONIC  OPERAND FIELDS   BUS CYCLE DATA
------------------------------------------------------------------
SEC:monitor
0038+CMND
0038+0000  F7068004800   TEST     WORD PTR csr,0080
0034+0006  74F8          JE       SHORT CMND
0033+0008  C606800402    MOV      BYTE PTR csr,02
0031+000D  C606860402    MOV      BYTE PTR sh_csr,02
0027+0012  A02040        MOV      AL,BYTE PTR es10
0026+0015  800E860404    OR       BYTE PTR sh_csr,04
0021+001A  8A268604      MOV      AH,BYTE PTR sh_csr
0018+001E  88268004      MOV      BYTE PTR csr,AH
0014+Tauc
0014+0022  F70680048000 TEST      WORD PTR csr,0080
0010+0028  75F8          JNE      SHORT Tauc
0008+002A  EBD4          JMP      SHORT CMND
0005+CMND
0005+0000  F706          TEST     WORD PTR 0000,06F7
```

The following example shows trace disassembly without section definitions.

```
 >DEL 'monitor
FULL T.M. MAP:  PASS 1  PASS 2
SEQ#  ADDR   OPCODE MNEMONIC  OPERAND FIELDS   BUS CYCLE DATA
---------------------------------------------------------------
0038 CMND
0038 1000  F7068004800  TEST     WORD PTR csr,0080
0034 1006  74F8         JE       SHORT CMND
0033 1008  C606800402   MOV      BYTE PTR csr,02
0031 100D  C606860402   MOV      BYTE PTR sh_csr,02
0027 1012  A02040       MOV      AL,BYTE PTR es10
0026 1015  800E860404   OR       BYTE PTR sh_csr,04
0021 101A  8A268604     MOV      AH,BYTE PTR sh_csr
0018 101E  88268004     MOV      BYTE PTR csr,AH
0014 Tauc
0014 1022  F70680048000 TEST     WORD PTR csr,0080
0010 1028  75F8         JNE      SHORT Tauc
0008 102A  EBD4         JMP      SHORT CMND
0005 CMND
0005 1000  F706         TEST     WORD PTR 0000,06F7
```

(continued)

The following example shows a memory disassembly with both sections and symbols purged, followed by a trace disassembly with no section or symbol definitions.

```
>PUR
>SYM;SEC
>
>DIS GRO
1000  F70680048000 TEST     WORD PTR 0480,0080
1006  74F8         JE       SHORT 1000
1008  C606800402   MOV      BYTE PTR 0480,02
100D  C606860402   MOV      BYTE PTR 0486,02
1012  A02040       MOV      AL,BYTE PTR 4020
1015  800E860404   OR       BYTE PTR 0486,04
101A  8A268604     MOV      AH,BYTE PTR 0486
101E  88268004     MOV      BYTE PTR 0480,AH
1022  F70680048000 TEST     WORD PTR 0480,0080
1028  75F8         JNE      SHORT 1022
>
>DTB
FULL T.M. MAP:  PASS 1  PASS 2
SEQ# ADDR   OPCODE MNEMONIC  OPERAND FIELDS   BUS CYCLE DATA
-------------------------------------------------------------------
0038 1000  F7068004800  TEST     WORD PTR 0480,0080
0034 1006  74F8         JE       SHORT CMND
0033 1008  C606800402   MOV      BYTE PTR 0480,02
0031 100D  C606860402   MOV      BYTE PTR 0486,02
0027 1012  A02040       MOV      AL,BYTE PTR 4020
0026 1015  800E860404   OR       BYTE PTR 0486,04
0021 101A  8A268604     MOV      AH,BYTE PTR 0486
0018 101E  88268004     MOV      BYTE PTR 0480,AH
0014 1022  F70680048000 TEST     WORD PTR 0480,0080
0010 1028  75F8         JNE      SHORT 1022
0008 102A  EBD4         JMP      SHORT 1000
0005 1000  F706         TEST     WORD PTR 0000,06F7
```

DISPLAY SYMBOLS

Command	Result
SYM	Displays all defined symbols.
SYM *<value>*	Displays all symbols assigned the specified value.
'*<symbol>*	Displays the value of the specified symbol.

Examples

```
 >'sym = 1000
 >'start = 8000
 >'end = 'start +37E
 >SYM
$00001000 sym
$00008000 start
$0000837E end
```

DISPLAY SECTION

Command	Result
SEC	Displays all currently defined sections and their values.
SEC *<value>*	Displays the section assigned the specified value.
'*<section>*	Displays the value of the specified section.

Examples

```
 >'sec = 1000 LEN IF
 >'init_mod = 'start TO 'end
 >'RAM =$0000 TO $FFFF
 >SEC
$00001000 TO $0000101F sec
$00008000 TO $0000837E init_mod
$00000000 TO $0000FFFF RAM
```

Command	Result
'*<symbol>* = *<value>*	Assigns the *<value>* to the specified symbol.

Comments

A space indicates the end of the symbol name. Symbol names can be up to 64 characters long, but only 16 character names can be uploaded and downloaded.

<symbol>	Any combination of ASCII characters with decimal values in the range 33-126. This range includes all of the printable ASCII characters.
<value>	A 32-bit integer value or a range.

Be sure to end a symbol name with a space when assigning a value. If a space is not entered as the last character of a symbol name, the characters that follow are recognized as a continuation of the symbol.

Once you type the single quote, the Emulator displays what you type in lower case letters unless you explicitly type upper case letters (using the shift key). After a space is typed (ending the symbol name), display reverts to all upper case letters.

If a symbol name is assigned a value that is a range, it is assumed that you are defining a section. Section range values cannot overlap.

(continued)

Examples

`'testing` is recognized as the symbol.

```
>'testing =GRO
```

`'testing=GRO` is recognized as the symbol name. The name will probably not be found and you will get an error message.

```
>'testing=GRO
```

```
>'section_X =10000 TO 1FFF
>'main_loop ='prog_start TO 'RAM_START-1
```

DELETE A SYMBOL OR SECTION

Command	Result
DEL '*<symbol>*	Deletes the specified symbol.
DEL '*<section>*	Deletes the specified section.

Examples

```
 >SYM
$00001000 Sym
$00008000 start
 >DEL 'Sym; SYM
$00008000 start
 >
```

DELETE ALL SYMBOLS AND SECTIONS

Command	Result
PUR	Purges all symbols and section references.

Comments

Be sure to purge before downloading symbols that may already be defined. If you do not, an error occurs and the download is aborted.

```
 >SYM
$00001000 sym
$00008000 start
$0000837E end
 >SEC
$00001000 TO $0000101F sec
$00008000 TO $0000837E init_mod
$00000000 TO $0000FFFF RAM
 >PUR;SYM;SEC
 >
```

Miscellaneous Commands

DISPLAY THE SOFTWARE REVISION DATES

Command	Result
REV	Displays the software revision dates for ESL, firmware, and disassembler. See page 3- 15.

Comments

This command is valid only in pause mode.

When you call AMC customer service, they will ask you what software revisions are in your machine. This command gives you the necessary information.

Examples

```
 >REV
WED AUG 6   08:50:26 PDT 1986 - ESL 1.0
WED AUG 6   16:50:26 PDT 1986 - FW  1.0
TUES DEC 1  12:20:32 PDT 1986 - DIS 1.0
 >
```

DISPLAY A BLANK LINE

Command	Result
RET	Executes a [RETURN], linefeed.

Comments

This command improves readability when displaying multiple lines of data.

Examples

Display two blocks of data, separating them with a blank line.

```
>DB SS:SP LEN 20;RET;DB DS:DX LEN 20
07FF76                     02 06 - 20 46 40 62 00 00 12 20        .. F@b...
07FF80  07 90 90 00 70 20 03 07 - 47 41 63 01 01 21 21 71 ....p ..GAc..!!q
07FF90  01 90 06 21 12 13                                 ...!..

088060  01 02 03 04 05 06 07 08 - 00 20 21 22 23 24 25 26 ........ !"#$%&
088070  30 31 32 33 34 35 36 37 - 55 56 50 49 48 47 30 30 01234567UVPIH600
```

SECTION 6

Table of Contents

Target Commands

Page

TARGET COMMANDS

Introduction

The term "run mode" indicates that emulation has begun, that the microprocessor in the pod is running a program in the target. Pause mode is the opposite of run mode. The term "pause mode" indicates that emulation is not taking place. Generally, the target is the hardware and software that you are debugging. If there is no target hardware available, the target may be just a program, downloaded into the overlay memory. The microprocessor in the Emulator's pod replaces the microprocessor in the target. This gives the Emulator control of the processor, which in turn gives you use of the powerful Event Monitor System and the ability to see what is happening within the target system.

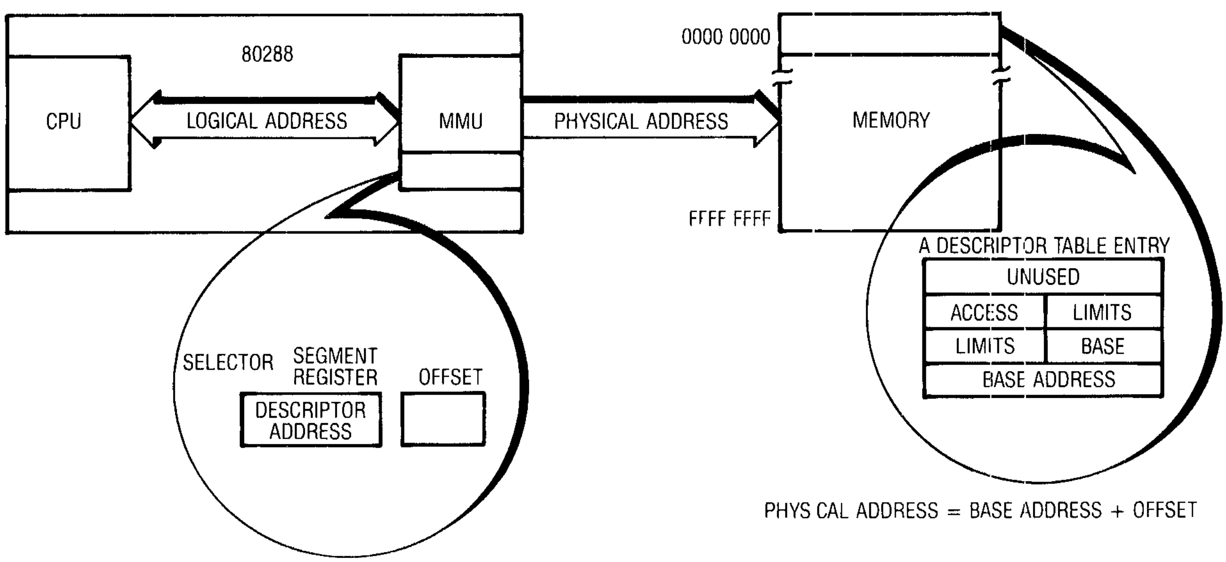
The processor in the pod runs the target in real-time. All processor functions are available and valid during emulation.

Virtual Operations

In real mode, the 80286 behaves like an 8086, except that fewer clock cycles are required for most instructions. Only 20 of the 24 address lines are used, and the 20 bit addresses are generated by shifting the 16 bit segment register left four places and adding the 16 bit offset.

In virtual protected mode, the segment register becomes a pointer to an entry in a descriptor table in memory. A descriptor table is a place in memory that stores the present physical location of each block of virtual memory, as well as protection information for that memory block.

Figure 7. Virtual Operations



The 80286 descriptor table contains a 24 bit physical segment address, which the on-chip memory management unit adds to the offset to generate the physical address. Since all 24 lines are used, 16 megabytes are accessible, as opposed to 1 megabyte when emulating the 8086.

Another feature of virtual operations is the existence of privilege levels, crucial for multiuser systems. Four privilege levels are possible, and the descriptor table entry notes which level may access each region of memory. This prevents users on a time-shared system from interfering with the executive program or other users' programs or data. It also means that the logical (virtual) addresses that the program thinks it is using are internal to the CPU chip and do not appear on the address bus.

These features make hardware debugging of the 80286 more complex than it is for other chips. The ES1800 Emulator supports the following features which aid in making debugging the 80286 possible:

- Showing logical and physical addresses in disassembly display
- Accessing all protected-mode registers
- Examining and modifying the descriptor table
- Assembling patches in protected mode
- Setting breakpoints on either logical or physical addresses
- Setting breakpoints on descriptor table changes (optional)
- Mapping on logical addresses

ADDRESS MODES

When programming a chip like the 80286, you use logical rather than physical addresses, but logical addresses are not available outside the chip. The translation from logical to physical takes many steps without an Emulator. It becomes still harder if the memory map changes after loading.

These physical-to-logical mappings are not necessarily one to one, so their translation must provide a table of all descriptors that would lead to the physical address. The ES1800 displays an error message if protection violation occurs in going from the logical to the physical. The ES1800 allows breakpoints to be set in one or a range of logical addresses and allows you to set a flag to make addresses of the form, `nnnnH:nnnnH` be taken systemwide as either logical or physical.

Two commands are used to globally designate the mode (physical or virtual) in which addresses are entered and displayed.

PHY Physical address mode. When the Emulator is in physical mode, a pair of 16 bit values separated by a [:] is interpreted and viewed as a segmented address with an offset. All addresses displayed by the Emulator are absolute values (no [:] operator), except the CS and IP in the register display. The segment value is rotated to the left four places and then added to the offset value prior to being displayed.

```
 >PHY
 >0F000:0001
$000F0001
 >
 >VIR
 >0F000:0001
$F000:0001
 >
```

VIR Virtual address mode. When the Emulator is in virtual mode, a pair of 16 bit values separated by a [:] is interpreted and viewed as a selector value with an offset.

```
 >PHY
 >0F000:0001
$000F0001
 >
 >VIR
 >0F000:0001
$F000:0001
 >
```

Processor and Emulator Mode Translation

The Emulator's physical and virtual modes interact with the processor's real and protected modes. Real and protected modes are determined by the PE bit in the CPU's MSW register (refer to the *Intel iAPX286 Hardware Reference Manual* and the *Intel iAPX286 Programmer's Reference Manual* for information on the PE and MSW registers). The following chart will help determine which registers are available and how addresses are displayed, depending on the combinations of modes used.

Processor and Emulator Mode Translation

PROCESSOR → / EMULATOR ↑	REAL	PROTECTED
PHYSICAL	Absolute addresses used. 8086 attribute used. No No "DRL" emulator command available. No modification or viewing of hidden MMU registers available.	Absolute addresses used. "DRL" available. Hidden MMU registers available for viewing. Protections of processor not enforced.
VIRTUAL	Virtual addresses used. All other 8086 attributes used. No "DRL" available. No modification or viewing of hidden MMU registers except LDTR and GDTR.	Virtual addresses used. "DRL" available. Modification and viewing of hidden MMU registers available.

USING THE DESCRIPTOR TABLES

The overlay memory function also works with descriptor tables, which are in memory. The ES1800 includes an assembler and disassembler for the descriptor table, allowing you to create a new table or look at an old one, thus speeding up the debug process. You may break emulation on access to any descriptor table entry and use descriptor table tools to write new values to the table. Then, by switching back to run mode, you can continue emulation with the new table contents.

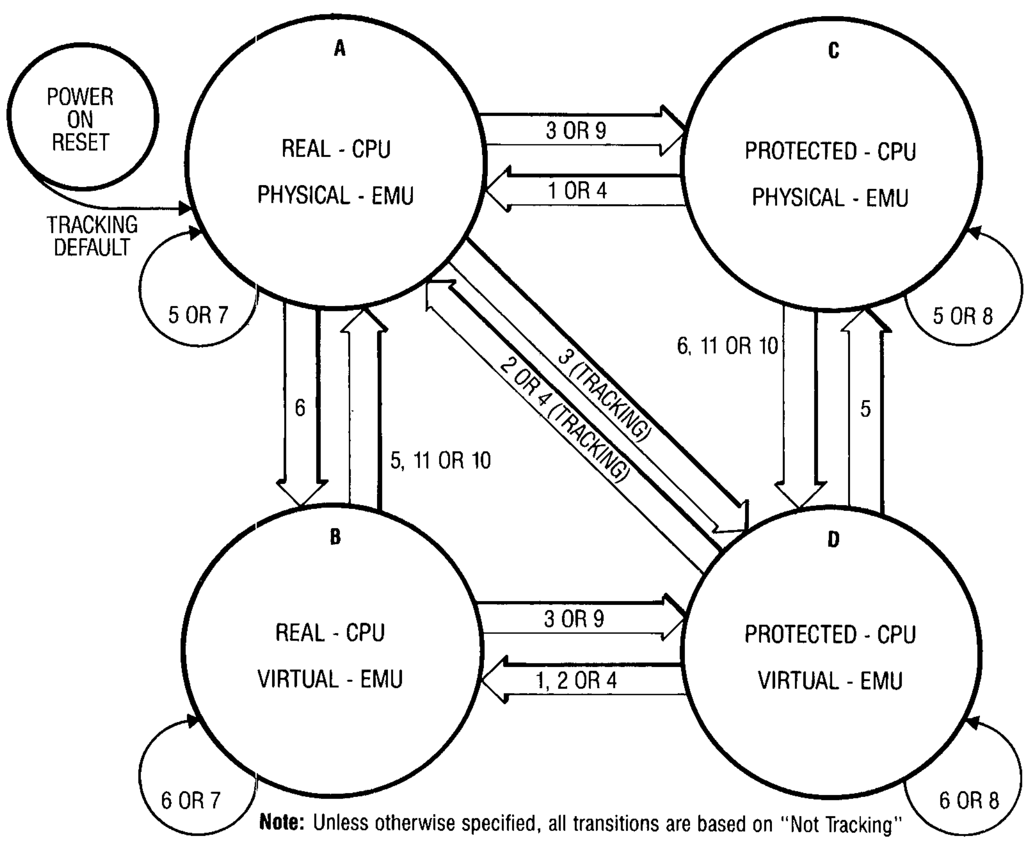
CPU AND EMULATOR STATE TRANSITION

The diagram describes the methods for moving from one state to another. A state is a combination of modes used by the Emulator and processor and is displayed via the **DMD** command. The Emulator controls the physical (**PHY**) and the virtual (**VIR**) modes. The processor is in control of the protected and real modes. The numbers refer to the action the Emulator or CPU must perform to change to a different state.

On power-on reset, the Emulator state tracks the state of the processor on a run-to-pause transition. This is referred to as "tracking."

When the virtual (**VIR**) or physical (**PHY**) command is issued and a run-to-pause transition occurs, the Emulator maintains the user- selected state (**VIR** or **PHY**) regardless of the processor state, indicating the Emulator is no longer tracking. Tracking can then be re-enabled by issuing the **TRK** command.

Figure 8. CPU and Emulator State Transitions



The numbers below explain the functions on the chart.

1. Cause a reset from the target hardware or issue an **RST** command from the Emulator keyboard while paused (not running).
2. Cause a reset from the target hardware or issue a **RST** command from the Emulator keyboard while running.
3. Set PE bit in MSW.
4. Clear the PE bit in the MSW.
5. Issue a **PHY** command on the Emulator (this clears tracking).
6. Issue a **VIR** command on the Emulator (this clears tracking).
7. Enter emulation (run or single step), then stop emulation while processor is in real mode.
8. Enter emulation (run or single step), then stop emulation while processor is in protected mode.
9. Execute code in the target that will put the processor into protected mode.
10. Run-to-pause transition (break on a **STP** command) while tracking is ON.
11. Issue a **TRK** command while in pause mode.

USING REGISTERS IN VIRTUAL MODE

The selector points to an area in memory that determines the actual address. That area in memory is called the descriptor table. The descriptor table entry has base, limit, and access information. The descriptor table assembler (ADT) and descriptor table disassembler (DDT) allow you to handwrite a descriptor table.

```
 >DRL
 IT    AX    BX    CX    DX    SI    DI    BP    SP
----  ----  ----  ----  ----  ----  ----  ----  ----
FFF0  0000  0000  0000  0000  0000  0000  0000  0000

FLX = ..........       IOP = 0      MSW = ../../../..

REG    INDEX   TI   RPL    BASE   LIMIT     ACCESS      DPL
-----------------------------------------------------------

CS      1E00   G     0    FF0000  FFFF    P C D U W .    0
DS      0000   G     0    000000  FFFF    P C D U W .    0
ES      0000   G     0    000000  FFFF    P C D U W .    0
SS      0000   G     0    000000  FFFF    P C D U W .    0
GDTR    ....   .     .    000000  0000    . . . . . .    .
LDTR    0000   G     0    000000  0000    . C D U . .    0
IDTR    ....   .     .    000000  FFFF    . . . . . .    .
TR      0000   G     0    000000  0000    . C D U . .    0
```

VIRTUAL OPERATIONS COMMANDS

The following six commands convert a physical address to a virtual address and vice versa, switch to physical or virtual mode, and assemble and disassemble the descriptor table.

PHYSICAL TO VIRTUAL

Command	Result
PTV *<expression>*	Converts a physical address to a virtual address. The *<expression>* is any address that evaluates to an unsigned binary value between 0 and $2^{24}-1$. The Emulator scans the descriptor tables in the target system to find the logical address record(s) that corresponds to the physical expression. The logical address is displayed in the format: *<selector>* :*<offset>*. If multiple mappings are possible, they are all listed. Valid descriptor tables must exist in the target memory for this command to produce accurate results.

VIRTUAL TO PHYSICAL

Command	Result
VTP *<selector>:<offset>*	Converts a virtual address to a physical address. The selector entry is found in the target system's descriptor tables; the physical address is calculated and displayed as an unsigned 24 bit value. If no entry is found, the error message, DESCRIPTOR TABLE INVALID, is displayed.

PHYSICAL ADDRESS MODE

Command	Result
PHY	When the Emulator is in physical mode, two 16 bit values separated by a [:] is interpreted and viewed as a segmented address with an offset. All addresses displayed by the Emulator are absolute values (no [:] operator), except the CS and IP in the register display. The segment value is rotated to the left four places and then added to the offset value prior to being displayed.

```
 >PHY
 >0F000:0001
$000F0001
 >
 >VIR
 >0F000:0001
$F000:0001
 >
```

ASSEMBLE DESCRIPTOR TABLE

Command	Result
ADT *<argument>*	Assembles a descriptor table entry and prompts the user for allowable inputs. As it is assembled, the table is expanded and displayed in **DDT** format.
.	Quits current line and assembles previous line.
,	Quits current line and assembles next line.
[RETURN]	Enters current line.
X *or* **Q**	Exits the **ADT** command at any time.

Comments

The **ADT** command may take one or two arguments or be used alone. In real mode with no arguments, it assembles a descriptor table starting at 0 or at the last entry to be assembled from a previous exit from **ADT**. **ADT** interprets a single argument as the table base, and assembly begins at this point in physical memory. With two arguments, **ADT** interprets the first as the table base and the second as the selector number (lower three bits are masked off). In real mode, **ADT** always assigns a limit of $FFFF.

(continued)

In protected mode with no arguments, **ADT** obtains the base and limit from GDTR the first time it is entered. On subsequent usage, **ADT** continues with the same base, limit, and selector that were in effect on previous exit. If the base of the table is modified or if the limit is set to a value less than the previous exit value, **ADT** interprets the first as a selector for the global descriptor table and the second as a selector for a local table. Assembly begins in the local table at the specified selector. An LDT descriptor must be present in the global table at the selector specified by the first argument.

When using **ADT** in real mode after using it in protected mode, if no arguments are specified it uses the base, limit, and selector in effect when exiting **ADT** in protected mode.

Examples

The following is an example of an **ADT** operation in protected mode.

```
 >ADT 500
GLOBAL DESCRIPTOR TABLE: BASE = $002000  LIMIT = $0FFF
$0500 - ...> P                             P=PRESENT, <RET>=NOT PRESENT
$0500 - PRS/ ...> S                        S=SEGMENT, C=CONTROL
$0500 - PRS/SEG/ ...> C                    D=DATA, C=CODE
$0500 - PRS/SEG/COD ...> R                 X=EXEC ONLY, R=EXEC/READ
$0500 - PRS/SEG/COD/XR/ ...> C             C=CONFORMING, <RET>=NOT CONF
$0500 - PRS/SEG/COD/XR/CNF ...> A          A=ACCESSED, <RET>=NOT ACCESSED
$0500 - DPL > 3                            DES PRIVILEGE LEVEL (0-3)
$0500 - DPL=3  BASE > 500000               BASE (0 TO $FFFFFF) <RET>
$0500 - DPL=3  BASE = $500000  LIMIT > 0FFFF LIMIT (0 TO $FFFF) <RET>
$0500 - DPL=3  BASE = $500000  LIMIT = $0FFFF<RET> = ENTER DESCRIPTOR
$0500 - PRS/SEG/COD/XR/CNF/ACC    DPL = 3   BASE = 50000   LIMIT = FFFF
$0508 - ...>X                              P=PRESENT, <RET>=NOT PRESENT
>
```

DISASSEMBLE DESCRIPTOR TABLE

Command	Result
DDT *<argument>*	Disassembles a descriptor table. Also displays a [DDT:] prompt accepting one of the following responses:
H	Displays on-line help.
.	Goes up one entry.
, or [RETURN]	Goes down one entry.
Space or D	Displays down one page in table.
U	Displays up one page in table. Mainly useful after Space or D.
X or Q	Exits the **DDT** command at any time.
<n>	Goes to specified selector.

DDT may take one or two arguments or be used alone. In real mode with no arguments, it disassembles a descriptor table starting at memory address 0, or at the last entry to be disassembled from a previous exit from **DDT**. **DDT** interprets a single argument as the table base. Assembly begins at this point in physical memory. With two arguments, **DDT** interprets the first as the table base, and the second as the selector number (lower three bits are masked off). In real mode, **DDT** always assigns a limit of $FFFF.

(continued)

In protected mode with no arguments, **DDT** obtains the base from the GDTR the first time it is entered. On subsequent usage, **DDT** continues with the same base, limit, and selector that were in effect on previous exit. If the base of the table is modified or if the limit is set to a value less than the previous exit value, **DDT** uses the new base and limit, and starts up at selector 0. It interprets a single argument as a selector. With two arguments, **DDT** interprets the first as a selector for the global descriptor table and the second as a selector for a local table. Disassembly begins in the local table at the specified selector. An LDT descriptor must be present in the global table at the selector specified in the first argument. When using **DDT** in real mode after using it in protected mode, if no arguments are specified, it uses the base, limit, and selector in effect when exiting **DDT** in protected mode.

The following is an example of a **DDT** operation.

```
 >DDT
GLOBAL DESCRIPTOR TABLE: BASE = $002000  LIMIT = $008F

$0000 - PRS/SEG/DAT/RW/EDU/ACC  DPL = 3  BASE = $500000  LIMIT = $FFFF
$0008 - PRS/SEG/DAT/RO/EDN/...  DPL = 3  BASE = $500000  LIMIT = $FFFF
$0010 - PRS/SEG/DAT/RO/EUP/...  DPL = 3  BASE = $500000  LIMIT = $FFFF
$0018 - PRS/SEG/COD/XO/CNF/ACC  DPL = 3  BASE = $500000  LIMIT = $FFFF
$0020 - PRS/SEG/COD/XR/.../...  DPL = 3  BASE = $500000  LIMIT = $FFFF
$0028 - PRS/SEG/COD/RW/CNF/...  DPL = 3  BASE = $500000  LIMIT = $FFFF
$0030 - PRS/SEG/COD/RD/.../ACC  DPL = 3  BASE = $500000  LIMIT = $FFFF
$0038 - PRS/SEG/COD/RD/CNF/ACC  DPL = 3  BASE = $500000  LIMIT = $FFFF
$0040 - PRS/CTL/TSS BUSY        DPL = 3  BASE = $500000  LIMIT = $FFFF
$0048 - PRS/CTL/TSS AVAILABLE   DPL = 3  BASE = $500000  LIMIT = $FFFF
$0050 - PRS/CTL/LDT DESCRIPTOR  DPL = 3  BASE = $500000  LIMIT = $FFFF
$0058 - PRS/CTL/RESERVED        DPL = 3  BASE = $500000  LIMIT = $FFFF
$0060 - PRS/CTL/INVALID         DPL = 3  BASE = $500000  LIMIT = $FFFF
$0068 - PRS/CTL/TASK GATE       DPL = 3  DST $5000:0000
$0070 - PRS/CTL/CALL GATE       DPL = 3  DST $5000:0000  WRDCT = $001F
$0078 - PRS/CTL/CALL GATE       DPL = 3  DST $5000:0000  WRDCT = $0000
$0080 - PRS/CTL/INTR GATE       DPL = 3  DST $5000:0000
$0088 - PRS/CTL/TRAP GATE       DPL = 3  DST $5000:0000
END OF DESCRIPTOR TABLE
DDT:
```

Command	Result
VIR	When the Emulator is in virtual mode, two 16 bit values separated by a [:] are interpreted and viewed as a selector value with an offset.

```
 >PHY
 >0F000:0001
$000F0001
 >
 >VIR
 >0F000:0001
$F000:0001
 >
```

(continued)

Emulation

STARTING EMULATION

Enter run mode by executing any of four run commands. Two of the run commands load the reset vectors before entering run mode, and two of them enable the breakpoints in the Event Monitor System. Event system breakpoints may be enabled or disabled during run mode. Even when breakpoints are disabled, all other Event Monitor System functions are active. The reset vectors are defined by Intel as:

```
CS  = F000
IP  = FFF0
FLX = 0002
```

The reset vectors cannot be loaded during run mode. **RUN** and **RBK** are typically used in run mode to disable and enable break points. The following table is a quick reference to the **RUN** commands.

Run Command	Load Reset Vectors	Break-points enabled	Valid in Run mode
RUN	NO	NO	YES
RNV	YES	NO	NO
RBK	NO	YES	YES
RBV	YES	YES	NO

Many Emulator commands are valid during run mode. If you are unsure whether a command may be entered during run mode, just enter it. An error message is displayed if it is not valid. Some commands need to communicate with the pod processor, and many of these commands cannot be entered during run mode, because emulation must stop in order to complete the command.

The following commands may be entered in run mode, but *do* halt emulation briefly in order to read or write data to the target system or overlay memory.

```
M      - Memory mode
MIO    - I/O mode
@      - Indirection operator
DB     - Display block of memory
ASM    - In-line assembler
DIS    - Memory disassembler
NXT    - Memory mode
LST    - Memory mode
```

If there are target hardware problems, it may not be possible to enter run mode. In these cases, error messages are displayed describing the problem. If the error conditions do not clear, a reset may be required to bring the system back into command entry mode.

HALTING EMULATION

Emulation can be halted in one of four ways:

1. Enter the stop emulation command, **STP**. When this command is entered during run mode, emulation is stopped and the values of the microprocessor registers are copied into Emulator memory. The current CS:IP and event monitor group number are displayed.

2. The Event Monitor System can stop emulation if you have set up breakpoints and the breakpoints are enabled. When a breakpoint condition occurs, emulation is halted, the microprocessor registers are copied into Emulator memory, and the CS:IP and event monitor group number are displayed.

(continued)

3. Issuing the reset character stops emulation. (See page 4-29 for information on the reset character.) After the reset character is issued, the Emulator registers have the same value they had before emulation began. The operator should check those values or load the reset vectors (**LDV**) before restarting emulation.

4. Emulation breaks automatically if the target program commits an access or write violation in overlay memory. (See page 5-67 for overlay memory access rights). An error message indicates the condition that caused the error.

USING REGISTERS IN RUN MODE

Setting and displaying the microprocessor registers during run mode can lead to unexpected results because the Emulator keeps a RAM image of the microprocessor registers. This image is copied to the processor whenever run mode is entered. The image is copied from the processor when emulation is stopped by the **STP** command or the Event Monitor System.

Because of this, modifying these registers during run mode simply alters the Emulator's image of the registers. The Emulator does not copy the new values of the registers to the microprocessor. When emulation is broken, the current values of the microprocessor registers are copied and the RAM image is overwritten. Thus, you cannot dynamically change the value of the microprocessor registers while emulating, and a display register command entered after emulation has begun will show you the register values upon entry to emulation, not the values the registers currently contain.

RUN TARGET PROGRAM

Command	Result
RBK	Begins executing the target program at the current CS:IP memory location with breakpoints enabled.
RBV	Loads the restart vectors and begins executing the target program at memory location FFFFF0H with breakpoints enabled.
RUN	Begins executing the target program at the current CS:IP memory location with breakpoints disabled.
RNV	Loads the restart vectors and begins executing the target program at memory location FFFFF0H with breakpoints disabled.

(continued)

Comments

Refer to Chapter 7, Event Monitor System, for breakpoint information.

RNV and **RBV** are valid only in pause mode.

All defined events are active while **RBK** and **RBV** are executing.

Run commands containing a **B** indicate that Event System breakpoints are enabled. Run commands containing a **V** indicate that the reset vectors are loaded prior to entering run mode.

Entering **RNV** is identical to entering **LDV;RUN** and entering **RBV** is the same as entering **LDV;RBK**.

STOP AND STEP TARGET SYSTEM

Command	Result
R>STP	From run mode the **STP** command stops emulation and returns the Emulator to pause mode. Displays the current CS:IP address and the Event Monitor System group number.
>STP	From pause mode, the **STP** command executes one instruction. To receive visual feedback, combine this command with a display command such as **STP;DT**.

Comments

[R>] indicates that the Emulator is in run mode. [>] indicates that the Emulator is in pause mode.

See the Switch section under STI, page 5-33 for more information about stepping.

(continued)

Examples

```
>STP;DR
>STP;DT
>STP;DIS IP LEN 4
```

LOAD RESET VECTORS

Command	Result
LDV	Loads the CPU reset vectors.

Comments

This command is valid in pause mode only.

RNV and **RBV** also load the reset vectors, then enter run mode. The **RST** command resets the processor if in run mode and always loads the reset vectors.

Intel defines the CPU reset vectors as:

```
CS  = F000
IP  = FFF0
FLX = 0002
```

To verify that the reset vectors are loaded, execute the **DR** command or individually display the CS, IP and FLX registers.

Refer also to Registers (page 5-74).

(continued)

Examples

Display the registers, then load the reset vectors, clear the data registers, and verify the changes by redisplaying the register set.

```
 >DR
CS:IP       FLX          AX   BX   CX   DX   DS   SI   ES   DI   BP   SS   SP
8000:1002  ......Z...   0100 FF00 1234 0040 C000 0000 D000 0000 0000 CC00 0024
 >LDV;CLR;DR
CS:IP       FLX          AX   BX   CX   DX   DS   SI   ES   DI   BP   SS   SP
F000:FFF0  ..........   0000 0000 0000 0000 C000 0000 D000 0000 0000 CC00 0024
 >
```

WAIT UNTIL EMULATION BREAK

Command	Result
WAI	Delays executing the specified command until emulation is broken.

Comments

Usually this command is used to delay executing a display command until an event system breakpoint is reached.

An event may never occur to bring the Emulator out of run mode. When this happens, use the system reset character to reset the system. (See pages 4- 29, 6-29 for more information on the reset character.)

After a reset, the delayed command is lost from the input buffer.

Examples

The Emulator disassembles a page of trace after a breakpoint is reached. Entering **RBK;DTB**, without the **WAI** command, results in a CANNOT EXECUTE COMMAND WHILE IN RUN MODE error.

```
RBK;WAI;DTB
```

(continued)

The Emulator runs until an access violation or a write violation is encountered, then displays a message pointed at by the BX register.

```
RUN;WAI;DIA BX
```

Command	Result
RST	Resets pod microprocessor and loads the reset vectors.

```
IP  = $FFF0
CS  = $FFF0
FLX = $0002
```

Comments

The **RST** command can be issued from either run or pause mode. When in pause mode, the **RST** command loads the reset vectors (**LDV**). While in run mode the microprocessor's reset pin is momentarily asserted. This causes the microprocessor to start fetching instructions from the reset vector. **RST** does not affect the target reset signal; therefore no target hardware is reset. This may cause problems when the target program tries to interact with unitialized hardware.

CTRL Z differs from **RST** in that CTRL Z stops emulation if in run mode. Further, CTRL Z does not initialize the emulator registers. CTRL Z resets the universal controller board, Emulator board, and pod, whereas **RST** only resets the pod and microprocessor.

(continued)

Examples

In the example below, the Emulator is in run mode. The microprocessor is reset in the target environment and emulation continues.

```
R> RST
R>
```

In the next example, the Emulator is in pause mode. The microprocessor is reset and the reset vectors are loaded into the Emulator registers.

```
>RST
>
```

Memory Commands

Memory commands allow you to modify and display memory in a number of different ways. "Memory" refers to memory in the target system or the Emulator's overlay memory. If the overlay memory is mapped (mapped memory will have the RW, RO or ILG attributes assigned to it), read and write accesses are directed to it. Mapped memory is modified by a memory command even if it is mapped as read only. If memory is unmapped, (memory with the TGT attribute assigned to it), memory command accesses are directed to the target system memory. Mapped and unmapped memory may be interleaved in any way you desire. See the Overlay Memory section (page 5-62) for details.

The default data length affects most memory commands. There are two data lengths to choose from: byte mode (BYM) and word mode (WDM). Commands that accept data parameters truncate the data entered to the current default data length. If you enter `FIN 0 LEN20,23F6` and the default data length is byte mode, the find command truncates the data field to `F6` and searchs the range for that byte. Commands that display data use the current data length.

Some memory commands may be executed during run mode. These commands halt emulation for a brief time in order to read from or write to memory.

(continued)

The following table shows the target-related commands that can be entered in run mode and the commands that are affected by the default data length.

Command	Legal in Run Mode?	Uses Default Data Length?
DB	YES	YES
FIN	NO	YES
FIL	NO	YES
BMO	NO	YES
VBL	NO	YES
LOV	NO	YES
VFO	NO	YES
ASM	YES	N/A
DIS	YES	N/A
M	YES	YES
MIO	YES	YES
@	YES	YES

Command	Result
DB *<address range>*	Reads and displays the specified address range.
DB	Reads and displays one page of memory, starting at the last address displayed by any previous **DB** command. On power-up, this command displays a page of memory from address zero.
DB *<address>*	Reads and displays one page of memory, starting at the specified address.

Comments

The page length is defined by the CRT length parameter in the **SET** menu (see page 5-3). When displaying a block of data in byte mode, the ASCII representation of each byte is also displayed.

The **DB** command provides an easy way to page through memory. Enter the **DB** *<address>* command to start reading memory at the desired address. Follow the display of this page of data with the **DB** command, and type a slash ([/]) (page 5-113). This repeats the **DB** command to increment the address and scroll through memory.

(continued)

If the display is longer than one page, the XON/XOFF characters can be used to start and stop scrolling (page 4-29).

DB affects real time operation when entered in run mode (see pages 6-18, 6-19 and 6-31).

Examples

Display 20 words pointed to by DS:DX.

```
>WDM; DB DS:DX LEN 20
```

Display a page of values pointed to by the value on top of the stack (see page 4-13 for information on @ operator).

```
>DB @SS:SP
```

Display block in byte mode and word mode.

```
>BYM
>DB 0 LEN 20
000000  80 48 45 4C 4C 4F 80 80 - 2F 0F F1 F9 5E 2F F6 F0  .HELLO../...^/..
000010  0F 03 F0 40 0F 0C F0 40 - 07 06 F0 90 0F 0C D8 00  ...@...@........

>WDM
>DB 0 LEN 2F
000000  4880  4C45  4F4C  8080 -  0F2F  F9F1  2F5E  F0F6
000010  030F  40F0  0C0F  40F0 -  0607  90F0  0C0F  00D8
000020  0FFF  F9FF  1FFF  7FFF -  3FFF  BDFF  1FFF  FFFF
```

Command	Result

FIN *<range>*, *<data>*	Searches *<range>* for the data pattern. All occurrences of the pattern are displayed:
	`$<address>=$<data>`
	`>`
	If the pattern is not found within the range:
	`NOT FOUND`
	`>`

Comments

This command is valid in pause mode only.

Data may be either an integer or don't care value. The find command uses the default data length, regardless of the length of the *<data>*. (See **SET** #26, page 5-7 for default data length in memory commands.)

Refer also to the "don't care" description (page 4-11).

(continued)

Examples

(Assume word mode.) To find a bit pattern using don't cares, use either of the following forms:

```
>FIN 1000 TO 2FFF, 60XX

   or

>FIN 1000 LEN 1000,6000 DC OFF
```

(Assume byte mode.) Find the initialization data in the start module section.

```
>FIN 'start_module,'init_uart
```

Find any NOPs in the range.

```
>FIN 100 TO 1000,90
```

Command	Result
FIL *<range>*,*<constant>*	Fills *<range>* with the *<constant>* data pattern.

Comments

This command is valid in pause mode only.

<constant> must be an integer.

The **FIL** command uses the default data length, regardless of the length of *<constant>*. (See page 5-3.)

The **FIL** command can be verified using the **VBL** (Verify BLock) command (page 6-38).

Examples

Fill RAM with zero to initialize data space.

```
>FIL 2000 LEN 50,0
```

Fill RAM section with initialization data.

```
>FIL 'ram, 'init_data
```

VERIFY BLOCK DATA

Command	Result
VBL *<address range>*, *<data>*	Verifies that *<address range>* contains the specified data.

Comments

This command is valid only in pause mode.

The **VBL** command uses the default data length, regardless of the length of *<data>*. (See page 5-3.)

Examples

Verify that a range contains $3F.

```
 >VBL 0 TO 2000,3F
$00000004 - $00, NOT $3F
$00000126 - $76, NOT $3F
 >
```

Command	Result
BMO*<range>*,*<address>*	Moves *<range>* to the new *<address>*.

Comments

This command is valid in pause mode only.

Examples

Move a range to a new location in data space.

```
>BMO 100 TO 500, 1000
```

Move 20 bytes from the stack to the value pointed to by the data register.

```
>BMO SS:SP LEN 20,DX
```

VERIFY BLOCK MOVE

Command	Result
VBM<*range*>,<*address*>	Verifies move of <*range*> to the new <*address*>.

Comments

This command is valid only in pause mode.

Verifies that a non-overlapping block move was successful.

LOAD OVERLAY MEMORY

Command	Result
LOV *<range>*	Moves data from the target system memory to the Emulator overlay memory in the specified address range.

Comments

The **LOV** command may not be entered during run mode.

Refer to the **VFO** command, page 6-42 to verify the load overlay command.

To load overlay memory from the target memory, a target system must be connected to the ES1800 Emulator and overlay memory installed and mapped.

To load a target memory range into the overlay memory at a different address, use the **LOV** command, then do a block move of the range.

Refer also to the Overlay Memory section, page 5-62.

Examples

```
>LOV 80000 LEN 7FFF
>LOV 'BOOT_RANGE
```

VERIFY OVERLAY MEMORY

Command	Result
VFO *<range>*	Compares *<range>* in target memory to the same range in the overlay memory. If there are any differences, the address and data difference is displayed: `<address> = XX NOT YY` XX is the data present in overlay memory. YY is the data at that location in the target system memory.

Comments

This command is valid only in pause mode.

Refer also to the Overlay Memory section, page 5-62.

Examples

To verify the two overlay loads in the **LOV** command section:

```
>VFO 80000 LEN 7FFF
>VFO 'BOOT_RANGE
```

LINE ASSEMBLER

Command	Result
ASM	Assembly begins at the last address displayed during a previous assembly session. At power-up the start address is zero.

```
 >ASM
**** 80286 LINE ASSEMBLER VX.XLA ****
0000 >X
 >
```

ASM <arg>	Assembly begins at the specified address.

```
 >ASM <address>
**** 80286 LINE ASSEMBLER  VX.XLA ****
0000 >END
 >
```

END X	Exits line assembly.

```
0000 >X
 >
```

Comments

Modification of the line assembler address is a two-step process.

1. To change the segment, use the CSEG directive after entering line assembly mode.
2. To change the offset, enter the assembler using a 16-bit address parameter, or use the ORG directive after entering the assembler.

All 80286 instructions can be entered from line assembly mode. The instructions are converted to machine code and loaded into memory at the address specified in the prompt.

The following pages describe the supported assembler directives.

ASSEMBLER DIRECTIVES

Command	Result

CSEG	Sets 64K byte code segment window:
	`1012 >CSEG D400H`
	`1012 >`

ORG	Sets 64K byte offset into the code segment window:
	`1012 >ORG 3ACH`
	`03AC >`

END or X	Exits line assembler to the command level:
	`58FD >X`
	`>`

DB

Defines constant byte data:

```
58FD >DB 1,2,3,4, "TEST", 0
58FD 01 02 03 04 54 45 53 54 00
5907 >
```

DW

Defines constant word data: (Note: odd length text strings are padded with nulls)

```
58FD>DW 1,2,3,4, "TEST", 0
58FD 0100 0200 0300 0400 4554
     5453 0000
590D >
```

PRE

Toggles to preview mode (causes next instruciton to be disassembled):

```
6590 >PRE
6590 C6470234 MOV BYTE PTR
     [BX+2H],34H
```

Toggles out of preview mode:

```
6590 C6470234 MOV BYTE PTR
     [BX+2H],34H
 >PRE
6590>
```

(continued)

EQU

Defines/redefines local symbol (L0-L9):

```
6590 >L3 EQU 7A44H
6590 >
```

or if symbolic debug hardware is installed:

```
6590 > 'Unit EQU 0FDE0H
6590 >
```

L0,L1...L9

Prints value of local symbol:

```
756A >L3
756A >L3 EQU 7A44h
756A >
```

<symbol>

Prints value of symbol. This is only valid if symbolic debug hardware is installed:

```
756A >'Unit
756A >'Unit EQU FDE0H
756A >
```

[RETURN]	Disassembles one instruction at the current address:

```
5D0A >
5D0A 3306AD78        XOR AX,WORD
     PTR 781DH
5DE >
```

$	Current assembler offset address.
NEAR	Within current line assembly segment.
FAR	Outside current line assembly segment.

MEMORY DISASSEMBLER

Command	Result
DIS *<range>*	Disassembles and displays the data in the specified range.
DIS *<address>*	Disassembles one page of memory beginning at a specified address.
DIS	Disassembles and displays a page of memory beginning at the last address display during previous **DIS** command. At power-up this value is zero.

Comments

You should be familiar with 80286 assembly language programming and have the *Intel iAPX286 Hardware Reference Manual* and the *Intel iAPX286 Programmer's Reference Manual.*

Page length is defined by the CRT length parameter in the **SET** menu (page 5-3).

The disassembly can be continued by typing a <space> or [RETURN]. Exit disassembly by typing any other character.

<space>	Continues disassembling one line at a time.
[RETURN]	Continues disassembling one page at a time.
any char except <space> or [RETURN]	Exits disassembly mode.

(continued)

Memory and I/O Modes

MEMORY MODE

Memory mode allows you to view and modify memory using a simple scrolling scheme. Enter memory mode by executing the **M** command. The current address and associated data are displayed. If the first character entered on a memory mode command line is a [RETURN], the next address and its data are displayed. If a value is entered before the [RETURN], that value is written to the current address before displaying the next address. A list of up to nine values separated by commas may be entered after a memory mode prompt. This data is stored to consecutive addresses.

The scroll direction is determined by two commands, **NXT** and **LST**. **NXT** (next) increments the address and **LST** (last) decrements the address. Entering either of these commands during run or pause mode sets the scroll direction. The scroll direction can also be changed after you have already entered memory mode by executing the appropriate command. The scroll direction can be manually overridden at any time by using the period [.] and comma [,] keys. A period increments the address; a comma decrements it.

The MMP register (memory mode pointer) is always set to the current address being accessed. If memory mode is entered without specifying an address, the value in this register specifies the starting address. On power-up, MMP is set to zero. (For further information on MMP see page 6-62.)

I/O MODE

I/O mode allows viewing and modification of the data in I/O address space. I/O mode is entered with the **MIO** command. Data is not automatically read from an I/O address on entry to I/O mode. Many I/O ports are "write only" ports, and trying to read from them may cause hardware problems. In order to read data from an I/O port, you must enter a [RETURN] as the only character on the line. The data is displayed, but the address is not automatically incremented. You must manually change the address while in I/O mode using the period and comma keys. A [.] increments the address and a [,] decrements the address. Up to nine values separated by commas can be entered in response to the I/O mode prompt. All of the values in the list are written to the same I/O address.

The IOP register (I/O pointer) is always set to the current I/O address being accessed. If I/O mode is entered without specifying an address, the value in this register will determine the starting address. On power-up, IOP is set to zero. (For further information on IOP, see page 6-61.)

ENTER MEMORY MODE

Command	Result
M <*address*>	Enters memory mode at <*address*>. The address and the data at that address are displayed preceding the prompt.
M	Enters memory mode at the last address examined in a previous memory mode session. The last address is stored in the MMP register, (Memory Mode Pointer). At power-up, this value is zero.
X	Exits memory mode.

Comments

The M command affects real time operation when entered in run mode (see pages 6-18, 6-19, and 6-31).

Data displayed in memory mode can be in either byte or word lengths. Set byte mode (**BYM**) or word mode (**WDM**) before entering memory mode. If you are in word mode and enter a byte of data, the byte is padded with zeroes and a word is written. If you are in byte mode and enter a word of data, the value is truncated, and only a byte is written. (See page 4-26.)

The MMP register is modified if you scroll to a new address while in memory mode. When you exit memory mode, MMP reflects the last address examined.

When a [RETURN] is entered as the first character on a line, the address is incremented or decremented and the new address and data are displayed. On power-up, the default scroll mode is toward increasing memory addresses. To change the scrolling direction use the **NXT** (forward) and **LST** (backward) commands. These can be entered in memory mode. If they are entered in pause mode, the scroll mode is set and memory mode is entered at MMP.

The scroll mode can be overridden by using the period and comma keys. A [.] increments the address and a [,] decrements the address.

To modify data at a memory location, enter the data and press [RETURN]. The data is written to the current address and the next address and data are displayed.

Data can be entered quickly using a list. A list can contain up to nine values separated by commas. See example below.

Examples

Set the MMP and use the **NXT** command to enter memory mode. Change a word of data and verify.

```
>WDM; MMP=$FF000; NXT
$0FF000 $1234  >1122
$0FF001 $00FF  >,
$0FF000 $1122  >X
>
```

(continued)

Assume that address 1000H is the start of a data table and you want to write a short program to utilize that data.

Initialize the data using a list. Then invoke the line assembler using MMP as the start address (page 6-44, 6-62).

```
 >M 1000
 $001000 $00  >0,1,2,3,4,5,6,7,8
 $001009 $00  >X
 >ASM MMP
****  80286 LINE ASSEMBLER  VX.XLA  ****

1009 >                              Enter your program here.
                                    Use 'X' or 'END' to exit
                                    the line assembler.
```

Command	Result
MIO *<address>*	Enters I/O mode at *<address>*. The port address is displayed, but no data is read until a [RETURN] is entered as the first character on the line.
MIO	Enters I/O mode at the last address examined in a previous I/O mode session. This address is stored in the IOP (I/O mode pointer) register. At power-up, this value is zero.
X	Exit I/O mode.

Comments

Affects real time operation when entered in run mode (see pages 6-18, 6-19, and 6-31).

The IOP is modified by scrolling to a new address while in I/O mode. When you exit I/O mode, the IOP reflects the last address examined. (See IOP, page 6-61.)

To read from an I/O port, enter I/O mode using one of the above commands, and enter a [RETURN] as the first character following the I/O mode prompt. The value of the current address is displayed.

(continued)

To write to the I/O port, enter the value and press [RETURN]. The value is written and the current address redisplayed.

Data can be entered quickly using a list. A list contains up to nine values separated by commas. All of the values in a list are written to the same address. The values are written 6 ms apart.

Addresses are not automatically incremented or decremented. Scrolling the address in I/O mode must be done manually, by using the period to increment the address, and the comma to decrement the address.

Examples

Enter I/O mode, write to a port and verify.

```
>MIO $2F00
IO:$2F00  >$7F
IO:$2F00  >
IO:$2F00 $7F >X
>
```

Set word mode and enter I/O mode at the last address, increment the address and read the data.

```
>WDM; MIO
IO:$2F00  >.
IO:$2F01  >
IO:$2F01 $05A6 >X
>
```

EXIT MEMORY AND I/O MODES

Command	Result
X	Exits memory or I/O mode.

SCROLLING IN MEMORY MODE

Command	Result
[RETURN]	Scrolls through memory addresses either one byte (8 bits) at a time, or one word (16 bits) at a time.
LST	The [RETURN] key now decrements addresses in memory mode.
NXT	The [RETURN] key now increments (default mode) addresses in memory mode.
.	Increments the address in memory mode.
,	Decrements the address in memory mode.

Comments

The **NXT** and **LST** commands may be entered from pause, run, or memory mode. If entered from run or pause mode, the [RETURN] key is set to increment or decrement.

When a comma or period is entered in memory mode, this temporarily overrides the scrolling direction.

I/O MODE POINTER

Command	Result
IOP	Displays the current value of the I/O mode pointer.
IOP = *<exp>*	Assigns the value *<exp>* to the I/O mode pointer.

Comments

IOP is the last value examined while in I/O mode. If you enter I/O mode without specifying an address, the IOP value is used as the entry point.

The default power-up value of the IOP register is zero. This register may be stored in EEPROM.

The I/O mode pointer is modified by moving to a new address after entering I/O mode. When you exit I/O mode, the IOP reflects the last address examined. As with any register, the IOP can be used as a parameter for other commands (see Memory and I/O Modes, page 6-52).

Examples

Set the IOP and verify.

```
 >IOP=$1100;IOP
$00001100
 >
```

MEMORY MODE POINTER

Command	Result
MMP	Displays the current value of the memory mode pointer.
MMP = *<exp>*	Assigns the value *<exp>* to the memory mode pointer.

Comments

MMP is the last value examined while in memory mode. If you enter memory mode without specifing an address, the MMP value is used as the entry point.

The default power-up value of the MMP register is zero. This register may be stored in EEPROM.

The memory mode pointer is modified if you change to a new address after entering memory mode. When you exit memory mode, the MMP reflects the last address examined.

As with any register, the MMP can be used as a parameter to another command. (See memory and I/O modes, page 6-52.)

Examples

Set the MMP and verify.

```
 >MMP=$12330;MMP
$00012330
 >
```

(continued)

Diagnostic Functions

The diagnostic functions (also called special functions or SFs) are a group of utility routines and special tests. They are valuable for locating address, data, status or control line problems. There are two categories:

- RAM tests
- Scope loops

For a complete list see the **SF** command (page 6-66).

RAM TESTS

The prewritten tests check that RAM is operating properly. They can be run on the target or overlay memory and may be executed in either byte or word mode. Byte or word mode must be specified prior to initiating the SF test.

SF 1 and 3 are modeled after a study by Abraham, Thatte, and Narir entitled *Efficient Algorithms for Testing Semiconductor Random-Access Memories* [IEEE Transaction on Computers, vol. c-27, no. 6 June 1978]. Refer to this publication for background information on these two diagnostics.

If you are going to test a large chunk of RAM, it may take a significant amount of time. If you attach a printer to the computer port and turn on the copy switch, you can let the test run while you do something else. The printer will record any errors that may occur in your absence.

SCOPE LOOPS

Scope loops are diagnostic routines built into the Emulator firmware for use when troubleshooting with an oscilloscope. The uses for these special functions range from locating stuck address data, status or control lines, to generating signatures using signature analysis equipment. Special Functions 4 through 12 are the memory scope loops and 24 through 32 are the I/O scope loops.

The firmware is optimized so that the loops execute at maximum speed. This short cycle time allows the hardware engineer to review the timing of pertinent signals in the target system without using a storage oscilloscope.

All of these routines must be terminated by resetting the Emulator with the reset character (see pages 4-29 and 6-29). The scope loops can be executed in either byte or word mode.

SPECIAL FUNCTIONS LIST

Command	Result
SF	Displays list of all available RAM tests, scope loops and miscellaneous tests.

Examples

```
>SF
SF 0,<RANGE><CR>                        SIMPLE RAM TEST, SINGLE PASS
SF 1,<RANGE><CR>                        COMPLETE RAM TEST, SINGLE PASS
SF 2,<RANGE><CR>                        SIMPLE RAM TEST, LOOPING
SF 3,<RANGE><CR>                        COMPLETE RAM TEST, LOOPING
SCOPE LOOPS:  {SELECT NUMBER FOR I/O LOOPS}
SF 4 {24},<ADDRESS>,<PATTERN><CR>       TOGGLE DATA AT ADDRESS
SF 5 {25},<ADDRESS><CR>                 READ FROM ADDRESS
SF 6 {26},<ADDRESS>,<DATA><CR>          WRITE DATA TO ADDRESS
SF 7 {27},<ADDRESS>,<PATTERN><CR>       WRITE PATTERN, THEN PATTERN COMPLEMENT
SF 8 {28},<ADDRESS>,<PATTERN><CR>       WRITE PATTERN, THEN ROTATE
SF 9 {29},<ADDRESS>,<DATA><CR>          WRITE DATA, THEN READ
SF 11 {31},<ADDRESS>,<DATA><CR>         WRITE INCREMENTING VALUE
SF 12 {32},<RANGE><CR>                  READ DATA OVER ENTIRE RANGE
MISCELLANEOUS:
SF 13<CR>                               CRC CHECK OF EMULATOR FIRMWARE
CLK <CR>                                DISPLAY TARGET CLOCK FREQUENCY
CRC <RANGE><CR>                         CALCULATE CRC OF SPECIFIED RANGE
CRE/CRO <RANGE><CR>                     CALCULATE CRC OF EVEN/ODD BYTES ONLY
 >
```

SIMPLE RAM TEST, SINGLE PASS

Command	Result
SF 0, *<range>*	Writes a test pattern to all locations within the specified range, then reads each location to verify the data. The following pattern sequence is used:

Sequence	Pattern BYM	Pattern WDM
1	0000 0000	0000 0000 0000 0000
2	0000 0001	0000 0000 0000 0001
3	0000 0011	0000 0000 0000 0011
4	0000 0111	0000 0000 0000 0111
5	0000 1111	0000 0000 0000 1111
6	0001 1111	0000 0000 0001 1111
7	0011 1111	0000 0000 0011 1111
8	0111 1111	0000 0000 0111 1111
9	1111 1111	0000 0000 1111 1111
10	1111 1110	0000 0001 1111 1111
11	1111 1100	0000 0011 1111 1111
12	1111 1000	0000 0111 1111 1111
13	1111 0000	0000 1111 1111 1111
14	1110 0000	0001 1111 1111 1111
15	1100 0000	0011 1111 1111 1111
16	1000 0000	0111 1111 1111 1111

(continued)

Comments

This command is valid in pause mode only.

If a location is read that does not match the test pattern, a failure is reported.

The address, correct data, and faulty data is displayed.

If no failure is detected, the following prompt is displayed:

```
TESTING RAM
COMPLETE
```

This is a single pass test.

COMPLETE RAM TEST, SINGLE PASS

Command	Result
SF 1, *<range>*	Writes, then reads, a test pattern to all locations in the specified range. Refer to *Efficient Algorithms for Test Semiconductor Random-Access Memories* mentioned in the introduction to Diagnostic Functions for the test pattern.

Comments

This command is valid in pause mode only.

If an error is detected, the associated address, correct data, faulty data, and test sequence number are displayed. The sequence number specifies which test in the complete list of tests caused the failure.

This is a single pass test.

Examples

```
TEST FAILED AT $20;GOOD DATA-$00, BAD DATA-$01 SEQ#-$02
```

An error is detected.

SIMPLE RAM TEST, LOOPING

Command	Result
SF 2, *<range>*	Writes a test pattern to all locations in *<range>*, then reads each location to verify the data. See SF 0 for test pattern. Each time the test is executed, the pass count is incremented and displayed on the screen.

Comments

This command is valid in pause mode only.

If no failure is detected, the pass line is the only line displayed. It is continually updated, showing the number of times the test has been executed.

```
SF 2, 0 TO 4
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
PASS COUNT = $XXXX
```

If a failure is detected, the problem address, correct data, and faulty data are displayed on the line after the pass number line, and the test continues.

```
 >SF 2,0 TO 4
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
TEST FAILED AT $02; GOOD DATA - $FE, BAD DATA - $FF
PASS COUNT = $0000
TEST FAILED AT $02: GOOD DATA - $FE, BAD DATA - $FF
PASS COUNT $0001
 .
 .
 .
until reset
```

You must issue the reset character to terminate this test (see pages 4-29 and 6-29).

COMPLETE RAM TEST, LOOPING

Command	Result
SF 3, *<range>*	Writes a test pattern to all locations within *<range>*, then reads each location to verify the data. See SF 1 for test reference information.

Comments

This command is valid in pause mode only.

During execution, a pass count is maintained and displayed on the screen.

If no failure is detected, the pass line is the only line. It is continually updated, showing the number of times the test has been executed.

```
>SF 3, 0 TO 2
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
PASS COUNT = $XXXX
```

If a failure is detected the associated address, the correct data, faulty data, and test sequence number are displayed.

```
>SF 3, 0 TO 2
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
TEST FAILED AT $02; GOOD DATA - $00, BAD DATA - $01  SEQ # - 02
PASS COUNT $0000
TEST FAILED AT $02; GOOD DATA - $00, BAD DATA - $01  SEQ # - 02
PASS COUNT $0001
 .
 .
 .
until reset
```

You must issue the reset character to terminate this test (see pages 4-29 and 6-29).

TOGGLE DATA AT ADDRESS

Command	Result
SF 4 *<address>*,*<data>*	*<data>* is written to the specified address in the memory space.
SF 24,*<address>*,*<data>*	*<data>* is written to the specified address in I/O space. The user defined data pattern is written to *<address>*, alternating with a data pattern of zeros.

SEQ	BYM	WDM
1	00	0000
2	XX	XXXX (user data)
3	00	0000
4	XX	XXXX (user data)
.	.	.
.	.	.
.	.	.

Comments

These commands are valid in pause mode only.

You must issue the reset character to terminate this test (see pages 4-29 and 6-29).

Examples

Assume you are in word mode (WDM).

```
 >SF 4, 2, $FFFF
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```

The data pattern written to address 2 is:

```
0000
FFFF
0000
FFFF
 .
 .
 .
```

PEEKS INTO THE TARGET SYSTEM

Command	Result
SF 5,*<address>*	Consecutively reads from the specified memory address.
SF 25,*<address>*	Consecutively reads from the specified I/O address.

Comments

These commands are valid in pause mode only.

You must issue the reset character to terminate this test (see pages 4-29 and 6-29).

Examples

```
>SF 5, 2
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```

POKES INTO THE TARGET SYSTEM

Command	Result
SF 6,<*address*>,<*data*>	Consecutively writes the user defined data pattern to the specified memory address.
SF 26,<*address*>,<*data*>	Consecutively writes the user defined data pattern to the specified I/O address.

Comments

These commands are valid in pause mode only.

You must issue the reset character to terminate this test (see pages 4-29 and 6-29).

Examples

```
>SF 6, 10,$FFFF
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```

(continued)

The data pattern written to address 10 is:

```
(BYM)  (WDM)
  FF    FFFF
  FF    FFFF
  FF    FFFF
```

WRITE ALTERNATE PATTERNS

Command	Result
SF 7,*<address>*,*<pattern>*	Consecutively writes the user defined data pattern to the specified memory address, followed by the complement of that data pattern to the same address.
SF 27,*<address>*,*<pattern>*	Consecutively writes the user defined data pattern to the specified I/O address followed by the complement of that data pattern to the same address.

Comments

These commands are valid in pause mode only.

You must issue the reset character to terminate this test (see pages 4-29 and 6-29).

Examples

```
>SF 7, 10, 55
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```

(continued)

The following data pattern is written to address 10:

```
BYM   WDM
55    0055
AA    FFAA
55    0055
AA    FFAA
.      .
.      .
.      .
```

WRITE PATTERN THEN ROTATE

Command	Result
SF 8, *<address>*, *<pattern>*	Consecutively writes the data pattern to the specified memory address, rotates the pattern 1 bit to the left, and writes to the same address.
SF 28, *<address>*, *<pattern>*	Consecutively writes the data pattern to the specified I/O address, rotates the pattern 1 bit to the left, and writes to the same address.

Comments

These commands are valid in pause mode only.

You must issue the reset character to terminate this test (see pages 4-29 and 6-29).

Examples

```
>SF 8,1000,05
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```

(continued)

WRITE PATTERN THEN ROTATE *(cont.)*

The following data pattern is written to address 10:

BYM	WDM
05	0005
0A	000A
14	0014
28	0028
50	0050
A0	00A0
41	0140
82	0280
	0500
	0A00
	1400
	2800
	5000
	A000
	4001
	8002

WRITE DATA THEN READ

Command	Result
SF 9, *<address>*, *<data>*	Consecutively writes the specified data pattern to the specified memory address, then reads from that same address.
SF 29, *<address>*, *<data>*	Consecutively writes the specified data pattern to the specified I/O address, then reads from that same address.

Comments

These commands are valid in pause mode only.

You must issue a reset character to terminate this test (see pages 4-29 and 6-29).

Examples

```
>SF 9, 100,$FFFF
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```

WRITE INCREMENTING VALUE

Command	Result
SF 11, *<address>*	Consecutively writes a constantly incrementing value to the specified memory address.
SF 31, *<address>*	Consecutively writes a constantly incrementing value to the specified I/O address.

Comments

These commands are valid in pause mode only.

You must issue the reset character to terminate this test (see pages 4-29 and 6-29).

Examples

```
>SF 11, 100
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```

READ DATA OVER AN ENTIRE RANGE

Command	Result
SF 12, *<range>*	Consecutively reads from the specified memory address range.
SF 32, *<range>*	Consecutively reads from the specified I/O address range.

Comments

These commands are valid in pause mode only.

The Emulator performs consecutive reads over the specified address range. The first read occurs at the starting address of the range. The address is then incremented for each additional read cycle. After the last address in the range has been read, the process starts again.

You must issue the reset character to terminate this test (see pages 4-29 and 6-29).

Examples

```
>SF 12, 10 TO 20
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```

CYCLIC REDUNDANCY CHECK

Command	Result
SF 13	A CRC is calculated on the ES 1800 internal PROM that contains the Emulator firmware.

Comments

This command is valid in pause mode only.

This is an Emulator self-test.

If a failure is detected, a CRC error is displayed.

This is a single pass routine.

When the text completes without an error, the command prompt ([>]) is displayed.

READ TARGET SYSTEM CLOCK

Command	Result
CLK	Reads the target system clock and displays the value in KHz. The value is accurate to plus or minus 2 KHz.

Examples

```
>CLK
CLOCK FREQUENCY = #2001 KHZ
>
```

TARGET CYCLIC REDUNDANCY CHECK

Command	Result
CRC *<address>*	The system calculates a cyclic redundancy check on all data bytes in address *<range>*.
CRE *<address range>*	Calculates a cyclic redundancy check on even addressed data bytes in *<range>*.
CRO *<address range>*	Calculates a cyclic redundancy check on odd addressed data bytes in *<range>*.

Comments

These commands are valid in pause mode only.

The **CRC** command generates a cyclic redundancy check value over a user defined address range. Only the byte mode is used for this test.

If code is split into two PROMs, with one even and the other one odd, the **CRE/CRO** commands allow you to do a cyclic redundancy check on each PROM.

CRC calculations can be used to determine if RAM based data is being corrupted. Do a **CRC** over the data base and save the value. Then run the program and do the **CRC** over the range again. If the values do not match, data is being corrupted. The Event Monitor System can be set up to catch writes to the data base. The **CRC** algorithm is based on the polynomial $X^{16}+X^{15}+X^2+1$.

DISPLAY STATUS OF STATUS LINES

Command	Result
BUS	Displays the state of the CPU's bus control signals.

Comments

The status of the following bus lines is displayed:

VCC	Power
RST	Reset
ALE	Address Latch Enable
RDY	Ready
HDA	Hold Acknowledge
INT	Interrupt
NMI	Non-maskable Interrupt
HLT	Halt
SDN	Shutdown
PRQ	Processor Extension Request
PEA	Processor Extension Acknowledge
BSY	Busy
ERR	Error

(continued)

(T)	Toggling
1	Asserted
0	Inactive

If a signal maintains a steady state for longer than 1 ms., a "1" or a "0" appears.

If a signal toggles within the 1 ms. sampling window, a "(T)" appears.

Examples

The following is an example of the screen display:

```
>BUS
  VCC  RST  ALE  RDY  HDA  INT  NMI  HLT  SDN  PRQ  PEA  BSY  ERR
   1   (T)  (T)  (T)  (T)  (T)  (T)  (T)  (T)  (T)   0    0    0
```

0 indicates an inactive condition. 1 indicates an active condition.

SECTION 7

Table of Contents

Event Monitor System

EVENT MONITOR SYSTEM

Overview

The ES1800's Event Monitor System provides extremely flexible system and breakpoint control, enabling you to isolate or break on any predefined series of events and then perform various actions. You control and monitor the target by entering commands that define events as logical combinations of address, data, status, count limit, and optional logic state probe inputs. When an event is detected, the ES1800 can break emulation, trace specific sequences, count events, execute user supplied target routines, and trigger TTL outputs.

WHEN/THEN control statements define events and the corresponding actions. There can be several actions for any event. The system only recognizes the first 3 letters of any word in a control statement (e.g., WHEN=WHE; THEN=THE). There can be many control statements in effect at any time. The Event Monitor System can also switch groups or states. There are four event groups available and the control statements and comparator values for any group are independent of those in other groups.

You can enter Event Monitor System control statements while in run mode. You can also modify the event comparator values during run mode. These new statements and values will not go into effect until you stop and restart run mode.

The ES1800 Event Monitor System monitors target information at the bus cycle level, including every read or write cycle that the microprocessor executes. The EMS "sees" every signal that can affect the target system. It can also monitor inputs from the logic state analyzer probe.

The EMS essentially takes a picture of the microprocessor's signals at the end of every TC02 state (refer to the Intel manual, *80286 Microprocessor Users Manual*). The information that is recorded into trace memory is the same information that the EMS is monitoring.

The basic Event Monitoring System control statement is of the form:

```
[Group] WHE[N] <event> THE[N] <action>
```

Notice that the ESL command processor needs only the first three letters of the symbol.

COMPARATOR REGISTERS

There are eight comparator registers for each of the four event groups. These event registers are listed in the table below. The address comparators are used to detect discrete addresses or addresses inside or outside a specified range. The data comparators can detect specific data patterns and can ignore specified bit positions. The status comparators monitor all of the status signals from the microprocessor as well as some generated by the ES1800. The status comparators can also ignore bit positions. The count limit register can be used to detect when an event has occurred more than a specified number of times. The logic state analyzer register can detect bit patterns in the inputs from the logic state probe.

The following table describes the available event comparator registers.

Register Description	Type	Size (bits)	Name by Group 1	2	3	4
Address 1	Range,Int	24	AC1 or AC1.1	AC1.2	AC1.3	AC1.4
Address 2	Range,Int	24	AC2 or AC2.1	AC2.2	AC2.3	AC2.4
Data 1	Don't Care,Int	16	DC1 or DC1.1	DC1.2	DC1.3	DC1.4
Data 2	Don't Care,Int	16	DC2 or DC2.1	DC2.2	DC2.3	DC2.4
Status 1	Don't Care,Int	16	S1 or S1.1	S1.2	S1.3	S1.4
Status 2	Don't Care,Int	16	S2 or S2.1	S2.2	S2.3	S2.4
LSA	Don't Care,Int	16	LSA or LSA.1	LSA.2	LSA.3	LSA.4
Count	Int	16	CTL or CTL.1	CTL.2	CTL.3	CTL.4

ADDRESS COMPARATORS

Address comparators may be assigned integer values or range values. Ranges may be either internal (IRA) or external (XRA). If a range is specified without IRA or XRA operators, the default range type will be IRA. The following are examples of valid address comparator assignments.

```
>AC1=2000
>AC2=1000 LEN 20
>AC2.2=XRA 1100 TO 1250
>AC1.4 = IRA $FF006 LEN $FF
>AC1.1 = @SS:SP
>AC2='Symbol
>AC1 =IP + 200
>AC1.2 = !AC1.4
```

DATA AND LSA COMPARATORS

The data comparators monitor the data bus for specified patterns. The LSA comparators monitor the input pulses from the logic state probe.

Data and LSA comparators may be assigned integer values or don't care values. Don't care values allow specified bits to be monitored while other specified bits are ignored.

For example:

```
0040 DC FFBF
```

In this case the Emulator monitors bit 6 for a high value and ignores the remaining bits.

Don't care values may be assigned in two ways.

- The first is to specify the value followed by the don't care mask
- The second is to specify the value using [X] in the don't care positions.

The following are examples of valid data and LSA comparator assignments.

```
>DC1=237F
>LSA=5300 DC $FF
>LSA.3 = 53XX
>LSA = %110101 DC $FF00
>DC2.2 = 42 DC %101
>DC2 = GDO + $F
>DC1.4 = @'data table + 56
```

STATUS COMPARATORS

Status information in the trace buffer represents miscellaneous bit information that is accessible on the bus every bus cycle. This information, as with all traced information, can be used to qualify the Event Monitor System. The following charts outline the various bit information associated with the status comparators (S1 and S2) of the Event Monitor System. These terms (WRD, BYT, IF, etc.) are to be used when setting S1 and S2 to a new value. Refer to the *Intel iAPX286 Hardware Reference Manual* and the *Intel iAPX286 Programmer's Reference Manual* published by the Intel Corporation.

STATUS MNEMONICS

MNEMONIC	VALUE	DESCRIPTION
WRD	$0000 MASK (DC) - $FFFE	Word access
BYT	$0001 MASK (DC) - $FFFE	Byte access
WR	$0000 MASK (DC) - $FFFD	Write cycle
RD	$0002 MASK (DC) - $FFFD	Read cycle
OVL	$0000 MASK (DC) - $FFFB	Overlay access
TAR	$0004 MASK (DC) - $FFFB	Target access
IOA	$0000 MASK (DC) - $FFF7	I/O access
MEM	$0008 MASK (DC) - $FFF7	Memory access
NMI	$0020 MASK (DC) - $FFDF	NMI cycle
IAK	$0000 MASK (DC) - $FF27	Interrupt acknowledge
RIO	$0090 MASK (DC) - $FF27	I/O read
WIO	$0050 MASK (DC) - $FF27	I/O write
HSD	$0008 MASK (DC) - $FF27	A0 high = halt A0 low = shutdown
RM	$0088 MASK (DC) - $FF27	Memory read
WM	$0048 MASK (DC) - $FF27	Memory write
IF	$0098 MASK (DC) - $FF27	Instruction fetch
LOK	$0000 MASK (DC) - $EFFF	Locked bus cycle
PEA	$0000 MASK (DC) - $DFFF	Processor extension ack
QF	$4000 MASK (DC) - $BFFF	Que flush

DC is don't care value

The status mnemonic table shows which status values can be assigned to the comparators. You may assign a status comparator a single mnemonic, or you may combine a mnemonic from each of the columns 1-4 and any or all from column 5. Mnemonics are combined using an addition operator ([+]) as a Boolean AND.

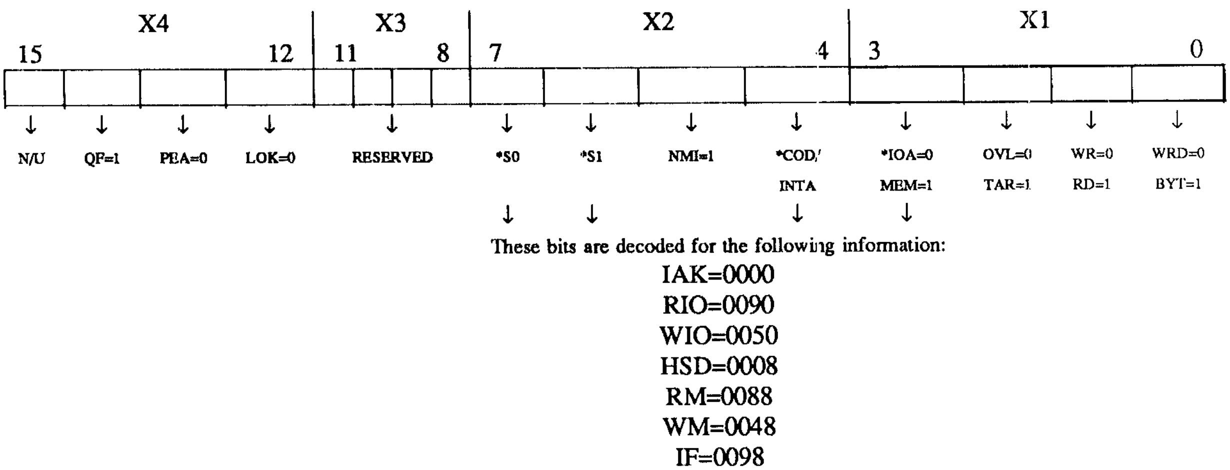
STATUS MNEMONIC TABLE

		1		2		3		4		5
S1	=	TAR	+	RD	+	BYT	+	IAK	+	QF
S2		OVL		WR		WRD		RIO		NMI
								RM		LOK
								WIO		PEA
								WM		
								IF		
								MEM		
								IOA		
								HSD		

Some examples of status comparator assignments:

```
>S1=BYT
>S2=OVL+RD+QF
>S1.3=WR+WRD
>S2.4=RIO
>S1.2=QF
```

Figure 9. Status Translation Table



When you display the value of the status comparators, you will see a 32-bit don't care value rather than the mnemonics you originally assigned them. The Status Translation Table is provided to aid you in decoding the numbers back into the mnemonics.

The don't care mask is the value to the right of the DC. A "0" in a mask bit position enables the status bit in the same position on the left side of the DC, and a "1" in a mask bit position masks or disables the corresponding bit on the left side of the DC.

Determine which bit positions are unmasked (those containing 0's in the mask value). It may be easier to do this by setting the status comparator's display base to binary (`BAS S1 = 2`). Then refer to the translation table and find the unmasked bit positions. Look at the value contained on the left side of the DC and match it with the corresponding value shown underneath the bit position in the table.

```
>S1
$000000A8 DC 0000FF07
```

All bits except bits 3, 4, 5, 6 and 7 are masked. Bit 5 is enabled and a 1 is in the bit 5 of the status value, so NMI was entered.

Bits 3, 4, 6, and 7 are enabled and there is a 1001 in those bits in the status value so RM was entered.

Therefore, the original input was:

```
>S1=NMI+RM
```

COUNT LIMIT COMPARATOR

The count limit comparator, **CTL**, is used to detect when events have occurred a certain number of times. The **CTL** value for group 1 is loaded into a hardware counter which is decremented whenever the action **CNT** is executed (see the following section on actions). If a group switch occurs, the hardware counter can be loaded with the new group's count limit by executing the **RCT** (reset count) action. Otherwise, the hardware counter does not change its limit value when switching groups.

Defining Events

The Event Monitor System is arranged in four independent groups. These groups provide a state-machine capability for debugging difficult problems. EMS control statements are associated with one of the four groups. If no group numbers are mentioned in the EMS control statement, the statement is assigned to group 1. There are two ways to override this default selection of group 1. You can begin the EMS control statement with a group number, or you can add a group number to any one of the event comparator names. For example: `3 WHEN AC1 THEN BRK` is functionally the same as `WHEN AC1.3 THEN BRK`. You cannot mix group numbers within a single EMS control statement.

EVENT

You can define an event to be some combination of address, data, status, count, and logic state probe conditions. Numerous Event Monitor System control statements may be entered and in effect simultaneously. Conflicting statements may cause unpredictable action processing. Parentheses are not allowed in event specifications.

The NOT operator reverses the sense of the comparator output. NOT has higher precedence than either of the conjunctives. `WHEN AC1 AND NOT DC1 THEN BRK` means break whenever any data pattern other than that in DC1 is written to an address in AC1.

AND and OR can be used where needed to form more restrictive event definitions. AND terms have higher precedence than OR terms. `AC1 AND DC1 OR DC2` is the same as `AC1 AND DC1` in one statement and `DC2` in another. If you are looking for two different data values at an address, you would use `AC1 AND DC1 OR AC1 AND DC2`.

The OR operator is evaluated left to right and is useful for simple comparator combinations. For complex event specifications, OR combinations can be replaced with separate EMS Control statements for clarity. `AC1 AND S1 OR AC2 AND S2` is the same as `AC1 AND S1` and `AC2 AND S2`.

Defining Action Lists

The action list in a WHEN/THEN statement defines what the Emulator does when an event is detected. Actions are specified in an action list separated by commas. The action list may have one or more actions defined.

```
<group> WHEN <event> THEN <action>,<action>, ... ,<action>
```

The following table lists all possible actions.

Event Monitor System Actions

Action	Description
BRK	Break emulation
CNT	Count bus cycle
FSI	Force special interrupt
GRO n	Change event group
RCT	Reset count value
TGR	Output trigger signal
TOC	Toggle count state
TOT	Toggle trace state
TRC	Trace bus cycle

The **TRC** and **TOT** actions are described in the Tracing Events section. The **CNT**, **RCT**, and **TOC** actions are described in the Counting Events section. The **FSI** action is described in the Special Interrupt section. The **GRO** action is described in the Changing Event Groups section. The **TGR** action is described in the Trigger Signal section. The **BRK** is described in the Breaking Emulation section.

DISPLAY EVENT SPECIFICATIONS

Command	Result
DES	Displays all of the WHEN/THEN statements currently active from all groups.
DES *<group number>*	Displays all of the WHEN/THEN statements and the comparator values for the specified group.

Examples

Displays the statements and comparators for groups 1 and 2.

```
 >DES 1;RET;DES 2
1 WHEN AC1 THEN BRK
AC1.1 = $007632
AC2.1 = $000000
DC1.1 = $0000
DC2.1 = $0000
S1 .1 = $0000
S2 .1 = $0000
LSA.1 = $0000
CTL.1 = $0000

2 WHEN S1 AND DC1 THEN CNT,TRC
2 WHEN CTL THEN BRK
AC1.2 = $000000
AC2.2 = $000000
DC1.2 = $40FF DC $00FF
DC2.2 = $0000
S1 .2 = $0003 DC $FFFC
S2 .2 = $0000
LSA.2 = $0000
CTL.2 = $0010
```

CLEAR WHEN/THEN STATEMENTS

Command	Result
CES	Clears all of the WHEN/THEN statements currently active.
CES *<group number>*	Clears all of the WHEN/THEN statements for the specified group.

Comments

The comparator values are not affected by the **CES** command.

Breaking Emulation

The **BRK** action stops emulation, returning the system to pause mode. When a break event is detected and emulation is broken, the current CS:IP and event group are displayed on the terminal. Emulation begins at the values displayed if the registers are not altered and you run or step following a break. When entering emulation, the Event Monitor System always begins looking for events specified in group 1.

Breakpoints stop program execution at specific times. After a break you can disassemble the trace memory, look at the LSA bits in the raw trace, check the CPU register values, or begin stepping through your code.

Breakpoint actions may be enabled or disabled by selecting the appropriate run commands. If you enter emulation with the **RBK** or **RBV** run commands, breakpoints are enabled. If you enter emulation with the **RUN** or **RNV** commands, breakpoints are disabled, even if there are event statements specifying the **BRK** action. If emulation is entered with breakpoints disabled, you can enable them while running by entering the **RBK** command. If you enter emulation with breakpoints enabled, you can disable them while running by entering the **RUN** command. The **RNV** and **RBV** commands are not allowed during emulation. These commands load the reset vectors, which cannot be done during emulation.

Breaking can also be qualified by a soft switch, BKX. This switch determines if breaks will occur on instruction execution, or on any access to an address, including prefetches (see page 5-13).

Emulation may also be halted using the **STP** command. The **RST** command and the reset character also break emulation.

(continued)

Examples

Breaks when the instruction at address $3000 is executed.

```
 >ON BKX
 >AC1=3000
 >WHEN AC1 THEN BRK
 >RBK
R>
```

Trace only accesses between 1000 and 113C; break after ten accesses to this address range.

```
 >AC1=1000 to 113C
 >CTL=#10
 >WHEN AC1 THEN CNT,TRC
 >WHEN CTL THEN BRK
 >RBV
R>
```

Break when 55AA is written to I/O port A.

```
 >AC1='PORT_A
 >DC1=55AA
 >S1=WIO
 >WHEN AC1 AND DC1 AND S1 THEN BRK
 >RBK
R>
```

Tracing Events

```
Events:

          TRC
          TOT
```

The Event Monitor System can be set up to selectively trace bus cycles. If all of the conditions specified in the event portion of the WHEN/THEN clause are satisfied, the trace action, **TRC**, causes the specified bus cycle to be recorded into the trace memory.

The toggle trace, **TOT**, allows you to turn tracing on and off. When a **TOT** event is detected, the trace is toggled to the opposite state, either on or off. You can specify a single event that starts and stops trace each time it is detected or specify any number of events that toggle trace on and off.

If there are no event actions that specify **TRC** or **TOT**, all bus cycles are traced. If there is a **TRC** event, only qualified bus cycles are traced. If there is a **TOT** event, trace is off until the **TOT** is detected, then all bus cycles are traced until encountering another **TOT** event.

ITR assembles the Event Monitor System if any comparators or clauses have been changed. Please see PPT (page 5-27) of the ON/OFF switches.

The Event Monitor System must be initialized via the **ITR** command after setting up the WHEN/THEN statements and event monitor registers and before starting the desired peek/poke cycle.

(continued)

This table describes the trace conditions immediately before and immediately after a group change.

Previous Group	**New Group**		
	Nothing Specified	*TRC*	*TOT*
Nothing specified	Trace all cycles	Trace only qualified cycles	No trace until first TOT
TRC	Trace all cycles	Trace only qualified cycles	No trace until first TOT
TOT OFF (not tracing)	Trace all cycles	Trace only qualified cycles	No trace until first TOT
TOT ON (tracing)	Trace all cycles	Trace only qualified cycles	No trace until first TOT

This table describes initial trace conditions.

Action Specified	**Trace Condition**
Nothing	Trace All Cycles
TRC	Trace Only Qualified TRC events
TOT	Trace Nothing until TOT event

Examples

Trace only a specific subroutine. Break at the end of the routine.

```
 >AC1='Sub_start
 >AC2='Sub_end
 >WHEN AC1 THEN TOT
 >WHEN AC2 THEN BRK
 >RBK
R>
```

(continued)

Counting Events

```
Registers:
                                      Value Type - 16 bit integer
      CTL
      CTL.1
      CTL.2
      CTL.3
      CTL.4

      CTL=<EXP>
      CTL<.group>=<EXP>

Events:

      CNT
      RCT
      TOC
```

Events can be defined to selectively count bus cycles. There is one hardware counter and there are four count registers, one register for each group. The hardware counter is automatically loaded with the count limit register for group 1 when entering run mode.

The count, **CNT**, action decrements the hardware counter. When the count reaches zero, the **CTL** event becomes true. If all other conditions specified in the WHEN/THEN clause are satisfied, the appropriate action is taken.

Whenever the reset count, **RCT**, action is specified, the count comparator value for the specified group is loaded into the hardware counter. When switching groups, the current value of the hardware counter is passed along as a global count value unless a **RCT** action is specified in the same list of events that causes the group switch.

The toggle count, **TOC**, command allows you to turn counting on and off. When a **TOC** event is detected, the count is toggled to the opposite state, either on or off. You can specify an event that starts and stops the counter each time it is detected or specify any number of events that toggle the counter on and off.

The current value of the counter cannot be read. You can only detect when you have reached a limit.

This table describes the count conditions immediately before and after a group change.

Previous Group	**New Group**		
	Nothing Specified	*CNT*	*TOC*
Nothing specified	No cycles counted	Count only qualified cycles	No count until first TOC
CNT	No cycles counted	Count only qualified cycles	No count until first TOC
TOC OFF (not counting)	No cycles counted	Count only qualified cycles	No count until first TOC
TOC ON (counting)	No cycles counted	Count only qualified cycles	No count until first TOC

(continued)

This table describes initial count conditions (always group 1).

Action Specified	Count Condition
Nothing	No Cycles Counted
CNT	Count Only Qualified CNT events
TOC	Count Nothing until TOC event

Examples

Count the times that the specified data is written to a specific address. Break if the data is written 20 times.

```
>CTL=#20
>S1=WR
>AC1=4020; DC1=$XXF3
>WHEN AC1 AND DC1 AND S1 THEN CNT
>WHEN CTL THEN BRK
>RBK
R>
```

Look for a read from a specific I/O port. After it is found go to group 2, load the group 2 counter register value into the hardware counter, and set a group 2 address comparator to count every bus cycle (all addresses). Break after 100 bus cycles.

```
 >AC1='IOport
 >S1=RD
 >WHEN AC1 AND S1 THEN GRO 2, RCT
 >CTL.2=#100
 >AC1.2=0 TO -1
 >2 WHEN AC1 THEN CNT
 >2 WHEN CTL THEN BRK
 >RBK
R>
```

(continued)

Trigger Signal

The trigger signal is an output that is available from the BNC connector labelled TRIG on the back panel of the ES1800 chassis and from pin 19 of the optional LSA pod. When a **TGR** event is detected, the trigger signal is asserted, and remains so for the duration of the specified bus cycle. If a trigger event is specified for more than one consecutive bus cycle, the signal stays high for the duration of the consecutive bus cycles.

The trigger signal can be used as a pulse output for triggering other diagnostic equipment. It can also be used with a counter/timer for timing subroutines.

Examples

Trigger a scope when reading data from a UART.

```
>AC1='DATA_PORT
>S1=RIO
>WHEN AC1 AND S1 THEN TGR
```

Determine the duration of a subroutine using the trigger pulse. The trigger pulse can be the input to a counter/timer or a scope. The duration of the subroutine can be determined from the pulse width displayed on the scope or the counter/timer readout.

```
>AC1=2500                    Start of subroutine
>AC1.2=AC1+38E               End of subroutine
>DC1.2=XXXX                  Detect any data pattern

>WHEN AC1 THEN TGR, GRO 2    Go to group 2 when subroutine is entered
>2 WHEN DC1 THEN TGR         Trigger during all cycles while in group 2
>2 WHEN AC1 THEN GRO 1       Go back to group 1 when last instruction
>RUN                         in subroutine is executed.
R>
```

Special Interrupts

```
Registers:

        SIA                        Value Type - 32 Bit Integer

Events:

        FSI
```

The force special interrupt action, **FSI**, allows you to jump to a specified address when a specific event is detected.

The special interrupt address register, SIA, should be set prior to entering the run mode if you are using the **FSI** event. It defines the address your program vectors to when the **FSI** is executed.

When an **FSI** event is detected, an `FSI ACTIVE` message is displayed on the screen. You may also see some unusual cycles in the trace memory at the address where the **FSI** occurred. These are internal cycles that are traced as the execution address is changed. These internal cycles are not purged from trace memory.

The **FSI** event can allow you to patch to your code fast. It can also allow you to write soft shutdown routines for machinery that cannot be halted using a simple breakpoint.

The **FSI** routine residing at the SIA address should terminate with an interrupt return (IRET) instruction. Execution resumes at the address immediately following the instruction that caused the **FSI**. If this is a soft shutdown, you will probably define a breakpoint at the IRET instruction.

(continued)

Examples

Make a patch using overlay memory

```
  >MAP 1000
  >AC1=8F36
  >WHEN AC1 THEN FSI
  >SIA=1000
  >ASM SIA                              Single line assembler - patch code
      .                                 can be assembled here.
      .
      .
  >RUN
R>
```

Assume the program needs to break at a certain address, but the machine cannot be turned off until a soft shutdown routine is executed. Set SIA to the address of the soft shutdown routine. Use an **FSI** action at the break address, then set a breakpoint at the end of the soft shutdown routine.

```
  >SIA='SHUT_down
  >AC1=$7F4E2
  >AC2='SHUT_down + 4E
  >WHEN AC1 THEN FSI
  >WHEN AC2 THEN BRK
  >RBK
R>
```

Changing Event Groups

The four event groups allow you to detect sequential events. When emulation is entered, event monitoring always begins in group 1. The example below describes a common use of the EMS group structure.

You may want to trace a subroutine after it has been called by Module A or Module B, but not if it has been called from Modules C, D, or E. In this case, define the address comparators in group 1 to the address ranges of Modules A and B. When either of these modules is encountered, switch to group 2 and look for the subroutine. After tracing the subroutine, switch back to group 1. Turn on the break on instruction execution (**BKX**) switch so that prefetching instructions do not trigger event actions.

```
>'Module_A =1240 LEN 246
>'Module_B =8750 LEN 408
>'Sub_X =8934 LEN 56
>ON BKX

>AC1='Module_A
>AC2='Module_B

>WHE AC1 OR AC2 THE GRO 2

>AC1.2='Sub_X

>2 WHEN AC1 THE TRC
>2 WHE NOT AC1 THE GRO 1
```

(continued)

The **TRC/TOT** and **CNT/TOC** actions interact in a specific way when event groups are switched. The following state transition tables describe the actions taken when each of the different event combinations are specified.

Previous Group	**New Group**		
	Nothing Specified	*TRC*	*TOT*
Nothing specified	Trace all cycles	Trace only qualified cycles	No trace until first TOT
TRC	Trace all cycles	Trace only qualified cycles	No trace until first TOT
TOT OFF (not tracing)	Trace all cycles	Trace only qualified cycles	No trace until first TOT
TOT ON (tracing)	Trace all cycles	Trace only qualified cycles	No trace until first TOT

Previous Group	**New Group**		
	Nothing Specified	*CNT*	*TOC*
Nothing specified	No cycles counted	Count only qualified cycles	No count until first TOC
CNT	No cycles counted	Count only qualified cycles	No count until first TOC
TOC OFF (not counting)	No cycles counted	Count only qualified cycles	No count until first TOC
TOC ON (counting)	No cycles counted	Count only qualified cycles	No count until first TOC

APPENDIX A

Table of Contents

ES Language Mnemonics

Page

ES LANGUAGE MNEMONICS

List of Commands

Command	Description	Page
>	Pause mode prompt	*4-24*
R>	Run mode prompt	*4-24*
[RETURN]	Return key	*4-2, 4-6*
[RETURN]	Disassemble one instruction (assembler directive)	*6-49*
[RETURN]	Scroll through memory (memory mode)	*4-25, 6-60*
/	Repeat previous command line	*4-2, 4-6, 5-113*
;	Statement separator	*4-6*
*	Repeat command	*4-6, 5-110*
CTRL Q	Start screen scrolling (can be changed)	*4-29*
CTRL R	Reprint current line	*4-6, 4-29*
CTRL S	Stop screen scrolling (can be changed)	*4-29*
CTRL X	Delete line	*4-6, 4-29*
CTRL Z	Reset the emulator (can be changed)	*4-29*

Command	Description	Page
ESC ESC	Escape transparent mode (can be changed)	*4-29, 5-41*
$	Hexadecimal	*4-12*
$	Current offset address (assembler directive)	*6-49*
#	Decimal	*4-12*
%	Binary	*4-12*
\	Octal	*4-12*
=	Equals	*4-3*
()	Parentheses	*4-8, 4-13*
@	Indirection	*4-9, 4-13, 4-18*
*	Multiplication	*4-17*
/	Division	*4-17*
+	Addition	*4-5, 4-17*
-	Subtraction	*4-5, 4-17*
-	Negation	*4-18*
&	Bitwise AND	*4-5, 4-17*
^	Bitwise OR	*4-5, 4-17*
<<	Shift left	*4-5, 4-17*
>>	Shift right	*4-5, 4-17*
!	Inverse, bitwise NOT	*4-17, 4-18*
:	Segment/offset operator or selector	*4-9*

Command	Description	Page
:	Address expression or memory attribute specifier	*4-9, 4-10*
.	Increment Memory Mode address	*6-52, 6-60*
.	Execute macro #2	*4-2, 4-6*
,	Decrement Memory Mode address	*6-52, 6-60*
,	Execute macro #1	*4-2, 4-6*
?	Help menu	*4-2, 4-6, 4-20*
?	Error query	*4-6*
_	Define/execute macro	*5-107*
'	Symbol definition (single quote)	*4-9, 5-120, 5-121, 6-48*
ABS	Absolute value	*4-17, 4-18*
AC1, AC2	Address comparators 1 and 2 (registers)	*5-77, 7-3, 7-10*
ADF	Force address register	*5-74*
ADT	Assemble descriptor table	*6-13*
AND	Logical event AND	*4-15, 4-16, 7-10*
ASM	Line assembler	*6-44*
AX, AH, AL	Accumulator (word, high byte, low byte)	*5-75*
BAS	Set/display base value	*5-83*

Command	Description	Page
BKX	Break on instruction execution switch	*5-13, 7-15, 7-27*
BMO	Block move	*6-39*
BP	Base pointer (register)	*5-75*
BRK	Break	*7-15*
BTE	Bus (RDY) timeout enable switch	*5-14*
BUS	Display status of lines	*6-89*
BX, BH, BL	Base register (word, high byte, low byte)	*5-75*
BYM	Byte mode	*4-26*
BYT	Byte access status	*7-6*
CCT	Computer port control	*5-44*
CEE	Coprocessor error enable switch	*5-16*
CES	Clear WHEN/THEN statements	*7-14*
CK	Internal/external clock selection switch	*5-17*
CLK	Read target system clock	*6-87*
CLM	Clear memory map	*5-70*
CLR	Clear microprocessor data registers	*5-78*
CMC	Clear macros	*5-109*
CNT	Count event	*7-9, 7-20, 7-28*
COM	Establish communication with target	*5-56*
CPY	Copy data to both ports switch	*5-18*
CRC/CRE/CRO	Target cyclic redundancy check	*6-88*
CS	Code segment register	*5-75*
CSA	Code segment access rights register	*5-75*

Command	Description	Page
CSB	Code segment base register	*5-75*
CSEG	Set code segment window (assembler directive)	*6-46*
CSL	Code segment limit register	*5-75*
CTL	Count limit register	*5-77, 7-9, 7-20*
CX, CH, CL	Count register (word, high byte, low byte)	*5-75*
DB	Display memory block	*6-33*
DB	Define constant byte data (assembler directive)	*6-47*
DBE	Descriptor table break enable switch	*5-19*
DC	Don't care	*4-9, 7-6*
DC1, DC2	Data comparators 1 and 2 (registers)	*5-77*
DDT	Disassemble descriptor table	*6-15*
DEL	Delete symbol/section	*5-123*
DES	Display WHEN/THEN statements	*7-12*
DFB	Set/display default base value	*5-88*
DFB	Default base register	*5-76*
DI	Destination index register	*5-75*
DIA	Display ASCII character string	*5-60*
DIS	Display disassembled memory	*6-50*
DM	Display memory map	*5-63*
DMD	Display modes being used by Emulator and processor	*4-26*
DML	Display virtual memory mapping log	*5-64*
DNL	Download	*5-46*

Command	Description	Page
DR	Display microprocessor registers	*5-78*
DRL	Display/load MMU registers	*5-81*
DRT	Display raw trace memory	*5-96*
DS	Data segment register	*5-75*
DSA	Data segment access rights register	*5-75*
DSB	Data segment base	*5-75*
DSL	Data segment limit register	*5-75*
DT	Disassemble trace memory	*5-100*
DTB	Disassemble trace memory backward	*5-104*
DTF	Disassemble trace memory forward	*5-104*
DW	Define constant word data (assembler directive)	*6-47*
DX, DH, DL	Data register (word, high byte, low byte)	*5-75*
END	Line assembler exit	*4-25, 6-44, 6-46*
ENI	Enable interrupts (switch)	*5-21*
EQU	Define/redefine local symbol (assembler directive)	*6-47*
ES	Extra segment register	*5-75*
ESA	Extra segment access register	*5-75*
ESB	Extra segment base register	*5-75*
ESL	Extra segment limit register	*5-75*
FAR	Outside current segment (assembler directive)	*6-49*

Command	Description	Page
FIL	Fill memory with data pattern	*6-37*
FIN	Find data pattern	*6-35*
FLX	Flags register	*5-76*
FSI	Force special interrupt	*7-25*
FSX	Force special interrupt on instruction execution switch	*5-22*
GD0-7	General purpose data registers (0-7)	*5-76, 5-90*
GDB	Global descriptor table base register	*5-75*
GDL	Global descriptor table limit register	*5-75*
GR0-7	General purpose range registers (0-7)	*5-76, 5-92*
GRO	Event monitor system group	*7-11*
HLT	Halt status	*6-89*
HSD	Halt or shutdown status	*5-98, 7-6*
IAK	Interrupt acknowledge status	*5-98, 7-6*
IDA	Interrupt descriptor table access rights register	*5-75*
IDB	Interrupt descriptor table base register	*5-75*
IDL	Interrupt descriptor table limit register	*5-75*
IDX	Repeat index register	*5-111*
IEE	Interrupt error enable (switch)	*5-23*
IF	Instruction fetch status	*5-98, 7-6*
IHE	Ignore halt errors	*5-24*

Command	Description	Page
M	Enter memory mode	*6-52, 6-54*
MAC	Display macros	*5-106*
MAP	Define overlay memory map	*5-66*
MEM	Memory access status	*7-6*
MIO	Enter I/O mode	*6-57*
MMP	Memory mode pointer register	*5-76*
MMP	Display value of MMP	*5-85, 6-62*
MOD	Modulo	*4-5, 4-17*
MSW	Machine status word register	*5-75*
NBC	No bus cycles status	*5-98*
NEAR	Within current segment (assembler directive)	*6-49*
NMI	NMI cycle status	*7-6*
NOT	Logical event NOT	*4-15, 7-10*
NXT	Increment address in memory mode	*6-52, 6-60*
OFF	Display/disable switches	*5-1, 5-9*
ON	Display/enables switches	*5-1, 5-9*
OR	Logical event OR	*4-15, 4-16*
ORG	Set offset into code segment window (assembler directive)	*6-46*
OVL	Overlay memory access status	*7-6*
OVS	Set overlay speed	*5-73*
PEA	Processor extension acknowledge status	*7-6*

Command	Description	Page
PEN	Pause mode protection enable switch	*5-26*
PHY	Physical address mode	*6-4, 6-12*
PPT	Trace peek and poke cycles switch	*5-27*
PRE	Preview mode (assembler directive)	*6-47*
PTV	Convert physical address to virtual	*6-10*
PUR	Clear symbolic memory	*5-125*
QF	Queue flush cycle status	*7-6*
RBK	Run with breakpoints	*6-18, 6-21, 7-15*
RBV	Load vectors and run with breakpoints	*6-18, 6-21, 7-15*
RCT	Reset count limit	*7-20*
RD	Read cycle status	*7-6*
RDY	Ready switch	*5-29*
RET	Display return and line feed	*5-127*
REV	Display software revisions	*5-126*
RIO	Read I/O status	*5-98, 7-6*
RM	Read memory status	*5-98, 7-6*
RMS	Real mode select switch	*5-31*
RNV	Run with new vectors	*6-18, 6-21, 7-15*
RO	Read only attribute	*5-67*
RST	Reset pod microprocessor	*6-29, 7-15*
RUN	Run emulation	*6-18, 6-21, 7-15*
RW	Read/write attribute	*5-66*

Command	Description	Page
S1, S2	Status comparators 1 and 2 (registers)	*5-77*
SAV	Save EEPROM data	*5-8, 5-36*
SEC	Display section	*5-21*
SET	Set/display system parameters	*5-1, 5-3*
SF	Display special function menu	*6-66*
SF 0-3	RAM Tests	*6-67, 6-73*
SF 4-12	Special functions (memory)	*6-74, 6-85*
SF 13	Special functions (Emulator)	*6-86*
SF 24-32	Special functions (I/O)	*6-76, 6-85*
SI	Source index register	*5-76*
SIA	Special interrupt address register selector:offset (virtual mode)	*5-77*
SIA	Special interrupt address register (real mode)	*5-77*
SLO	Slow interrupt enable switch	*5-32*
SP	Stack pointer register	*5-75*
SS	Stack segment register	*5-76*
SSA	Stack segment access register	*5-76*
SSB	Stack segment base register	*5-76*
SSL	Stack segment limit register	*5-76*
STI	Step through interrupts switch	*5-33*
STP	Step and stop	*6-24, 7-15*
SYM	Display symbols	*5-119*
TA	Task access rights register	*5-76*

Command	Description	Page
TAR	Target access status	*7-6*
TB	Task base register	*5-76*
TCT	Terminal port control	*5-43*
TGR	Enable trigger output	*7-24*
TGT	Target memory attribute	*5-67*
THE	THEN used in event statements	*4-4, 7-2*
TL	Task limit register	*5-76*
TO	To	*4-12*
TOC	Toggle counting	*7-20, 7-28*
TOT	Toggle trace memory	*7-17, 7-28*
TR	Task register	*5-76*
TRA	Transparent mode	*5-41*
TRC	Trace event	*7-17, 7-28*
TST	Test register for repeats	*5-94*
UPL	Upload serial data	*5-52*
UPS	Upload symbols	*5-54*
VBL	Verify block data	*6-38*
VBM	Verify block move	*6-40*
VFO	Verify overlay against target	*5-72, 6-42*
VFY	Verify serial data	*5-51*
VIR	Virtual address mode	*6-4, 6-17*
VTP	Convert virtual address to physical	*6-11*

Command	Description	Page
WAI	Wait until emulation is broken	*6-27*
WDM	Word mode	*4-26*
WHEN/THEN	WHEN/THEN Statements	*7-1, 7-2*
WIO	Write IO status	*5-98, 7-6*
WM	Write memory status	*5-98, 7-6*
WR	Write cycle status	*7-6*
WRD	Word access status	*7-6*
X	Don't care	
X	Exit memory or I/O mode	*4-25, 6-59*
X	Exit line assembler	*4-25, 6-44,*
	(assembler directive)	*6-46*
XRA	External range	*7-3*

APPENDIX B

Table of Contents

Error Messages

ERROR MESSAGES

ERROR MESSAGES

The following messages are displayed when the Emulator or target hardware is not functioning properly or when a software problem requires attention.

Messages followed by an asterisk (*) are Emulator hardware related.

Messages followed by two asterisks (**) are target hardware related.

Messages followed by three asterisks (***) are software related.

Using an oscilloscope, verify that the reset line is in fact being held reset. Some operating systems normally hold the microprocessor reset until needed. If the reset line is not being held active at the probe tip, then disconnect the target verify the condition using the null target mode.

*BUSY ASSERTED > .5 SEC***
Indicates a possible problem with the associated line and that the Emulator has detected this line being asserted for greater than .5 seconds.

*BUS TIMEOUT ERROR > 1 SEC***
This indicates a RDY was not returned to the CPU within 1 second. Please see the **BTE** command on the ON/OFF menu.

*DESCRIPTOR TABLE INVALID******	This indicates that the Emulator has been unable to access a valid descriptor table in the target system memory. This access must be done each time the Emulator converts between a virtual and physical address.
*DESCRIPTOR TABLE MODIFIED******	The Emulator has detected a write to one of the target system's descriptor tables. (The DTA soft switch must be ON to enable this message.)
*HANDSHAKE ERROR**	This is an internal Emulator diagnostic error. Type CTRL Z to clear the error. If it persists, call the factory.
*HOLD ACKNOWLEDGE ASSERTED****	Displays when a hold acknowledge is asserted for longer than .5 seconds. It is removed when the microprocessor regains control of the bus. This message is related to two environments: when a DMA controller takes over the bus by asserting the hold line, and when a microprocessor is running in a multiprocessor environment. This message is generally not due to an error, but is a statement of what the processor is doing.
*INTR ASSERTED > .5 SEC****	This indicates a possible problem with the interrupt line, which the Emulator has seen asserted for .5 seconds.
*MEMORY ACCESS VIOLATION******	If the target program attempts to access an area of target that is mapped as illegal (ILG), this error occurs. **DM** helps find areas mapped as illegal. **DRT** helps find where the program was making access.

*MEMORY WRITE VIOLATION****

If the program attempts to write to the overlay memory in an area that is mapped read only, an error occurs. Using the **DM** command and the raw trace (**DRT**) to look for write cycles helps troubleshoot this problem. **DM** helps determine areas mapped as read only. **DRT** helps find where the program performed the write.

*NO BUS CYCLES**

No ALEs were detected for at least 1.0 ms and no other fault conditions such as a halted, waiting, or reset microprocessor were found. Displays when a reset (power-up, CTRL Z or **RST**) is executed from the Emulator controller and no acknowledgment from the Emulator board is received. This situation is due to a pod, Emulator, or controller board problem. Possible solutions are reseating boards, reseating pod cables and cycling power.

*NO CLOCK***

This indicates the microprocessor clock frequency is less than 2 MHz.

At power-up, if there is no clock from the target, you are given the option of selecting an internal clock. If an external clock is selected, and the NO CLOCK message displays, you must use CTRL Z to either choose an internal clock or correct the problem with the external clock.

*NO TARGET POWER***

This displays when the +5V supply in the target system is at approximately +4V or less.

*PEREQ SIGNAL ASSERTED > .5 SEC***	Indicates a possible problem with the associated line and that the Emulator has detected this line being asserted for greater than .5 seconds.
*POD CPU NOT INITIALIZED**	When a reset occurs (power-up, CTRL Z or **RST**), the controller and Emulator begin an initialization routine to establish communication. If the initialization routine does not complete, the POD CPU NOT INITIALIZED message is displayed. This is a pod, Emulator, or controller board problem. Possible solutions are reseating boards, reseating pod cables and cycling power and verifying that the microprocessor is correctly installed in the pod.
*POD CPU NOT RESPONDING**	When a command is executed, the ES language software looks to see if the command has completed. If it has not, then the software checks to see if the microprocessor is still running or if there is an error condition. If an error condition exists, then the appropriate message is displayed. However, if the microprocessor is still running, and no error conditions exist, then POD CPU NOT RESPONDING is displayed. A possible solution is to execute a CTRL Z followed by repeating the command. As a last resort, try cycling the power.
*PROCESSOR HALTED***	This indicates that a halt (HLT) instruction has been executed and that the microprocessor has remained in this state for more than .5 seconds. The microprocessor is still in a run state and commands can still be entered.

*PROCESSOR SHUTDOWN***

This indicates that the Emulator has detected an Intel specified shutdown condition on the CPU status pins.

*PROCESSOR WAITING***

This means that the microprocessor is waiting for a ready to be returned to it. This is displayed only if the microprocessor has been waiting for more than .5 seconds. When the condition is corrected, the message is removed.

NOTE 1: You must use the target ready when overlaying dynamic RAM that uses the ready line to halt microprocessor activity during refresh cycles. When a refresh cycle occurs on many systems the ready line is held in the not ready state until refresh is complete. If an internal ready is used for dynamic RAM, then the microprocessor completes its present bus cycle before the refresh is complete and either writes bad data or reads bad data. The choice of internal or external ready while using overlay memory is made by using the RDY soft switch.

NOTE 2: When overlaying nonexistent code space it is necessary to use the internal RDY. Users may want to overlay nonexistent (meaning an area that is not decoded in their hardware) to patch in code to correct software bugs.

NOTE 3: When selecting internal or external RDY for areas overlaid, it is selecting that particular RDY for all overlay.

*RESET ASSERTED***

This indicates that a reset from the target has been asserted for more than .5 seconds. When this error message is displayed, system operation may be continued. If the error situation persists and another command is executed, the error is displayed again.

Using an oscilloscope, verify that the reset line is in fact being held reset. Some operating systems normally hold the microprocessor reset until needed. If the reset line is not being held active at the probe tip, then disconnect the target and verify the condition using the null target mode.

TARGET SOFTWARE ERRORS: PROTECTION VIOLATION

If the soft switch disabling protection is not asserted, the firmware provides a variety of protection checks for each memory or I/O access requested. If any of these checks fail, an error is flagged to the controller.

Protection violation errors appear in the format illustrated below:

```
PROTECTION VIOLATION: <message>
```

PRESENT

Checks that the requested segment resides in physical memory.

DPL < CPL

Checks that current privilege level is less than or equal to segments DPL.

SEGMENT NOT READABLE

(Peek) checks that segment is readable.

SEGMENT NOT WRITABLE	(Poke) checks that segment is writable.
WRITE TO EXECUTABLE SEGMENT	(Run/step) checks that segment is executable.
INVALID SEGMENT DESCRIPTOR	Checks that descriptor refers to a segment.
I/O PL	Checks that current privilege level (CPL) is less than or equal to the I/O privilege level.
OFFSET > LIMIT	Checks that requested offset is less than or equal to limit.

APPENDIX C

Table of Contents

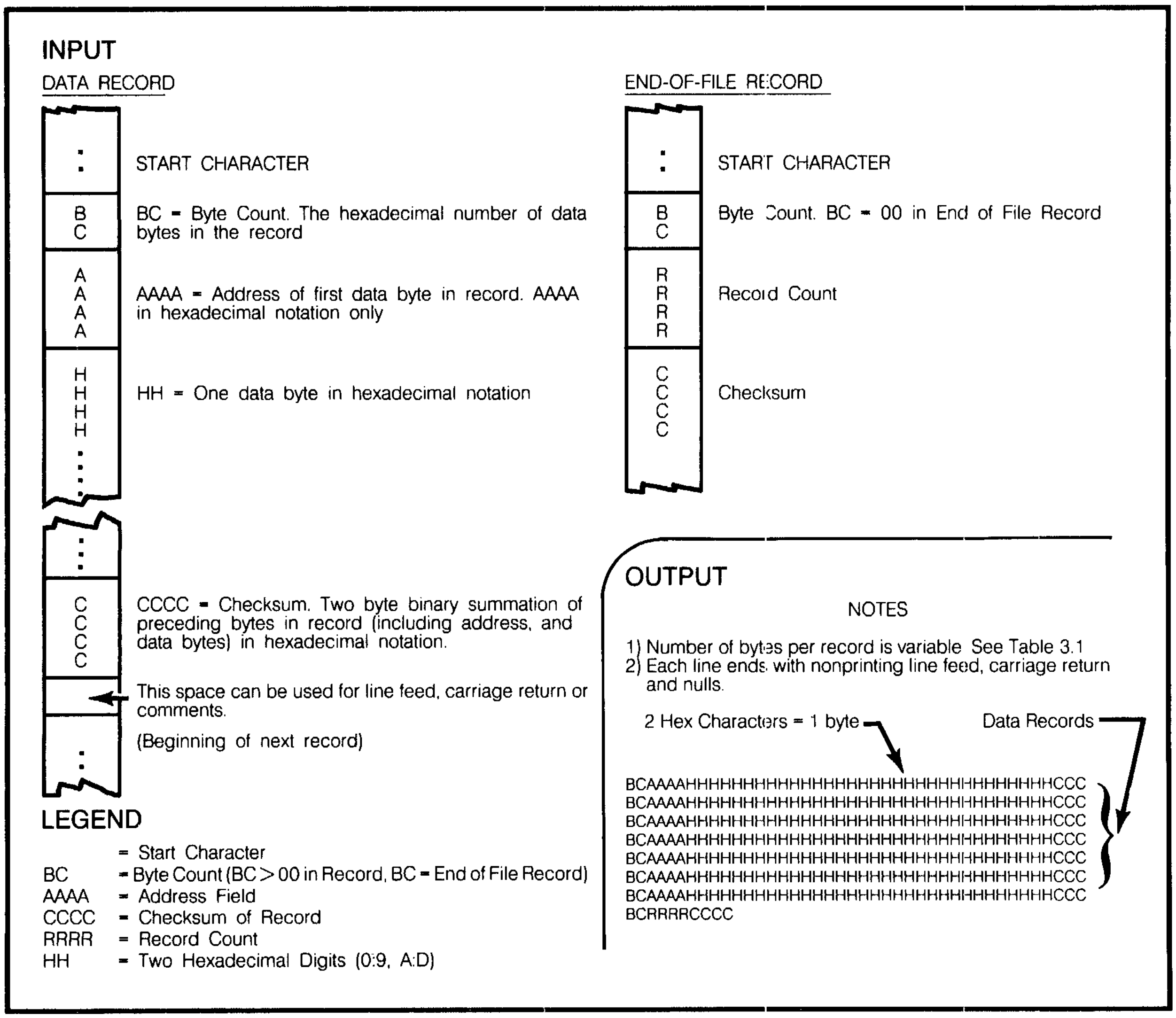
Serial Data Formats

SERIAL DATA FORMATS

In order to download a program into target memory, the ES1800 needs some way to receive this data in an intelligible format. This Appendix describes the downloading formats which the ES1800 understands.

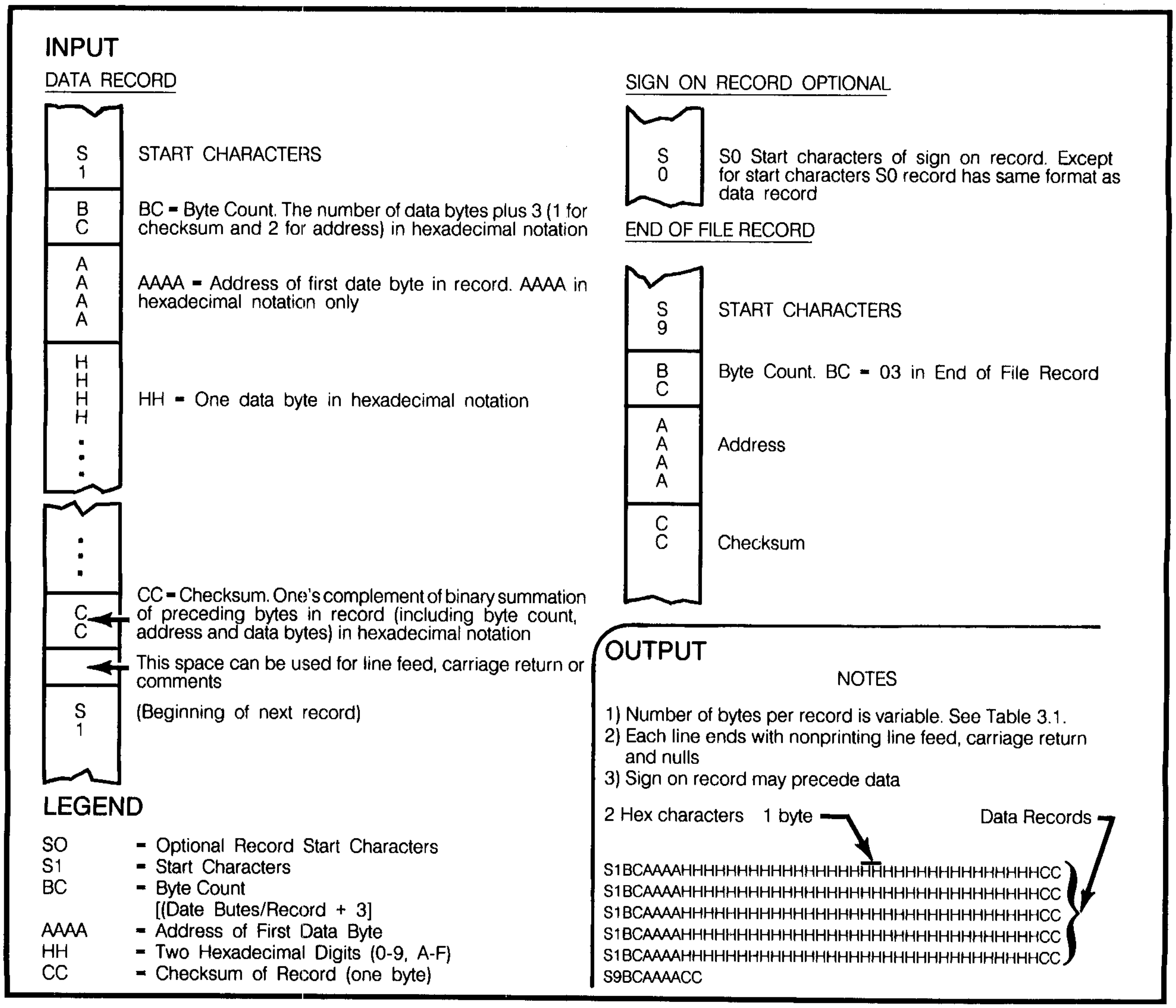
MOS Technology Format

Figure 10. Specifications for MOS Technology Data Files Copyright 1983, Data I/O Corporation; reprinted by permission.



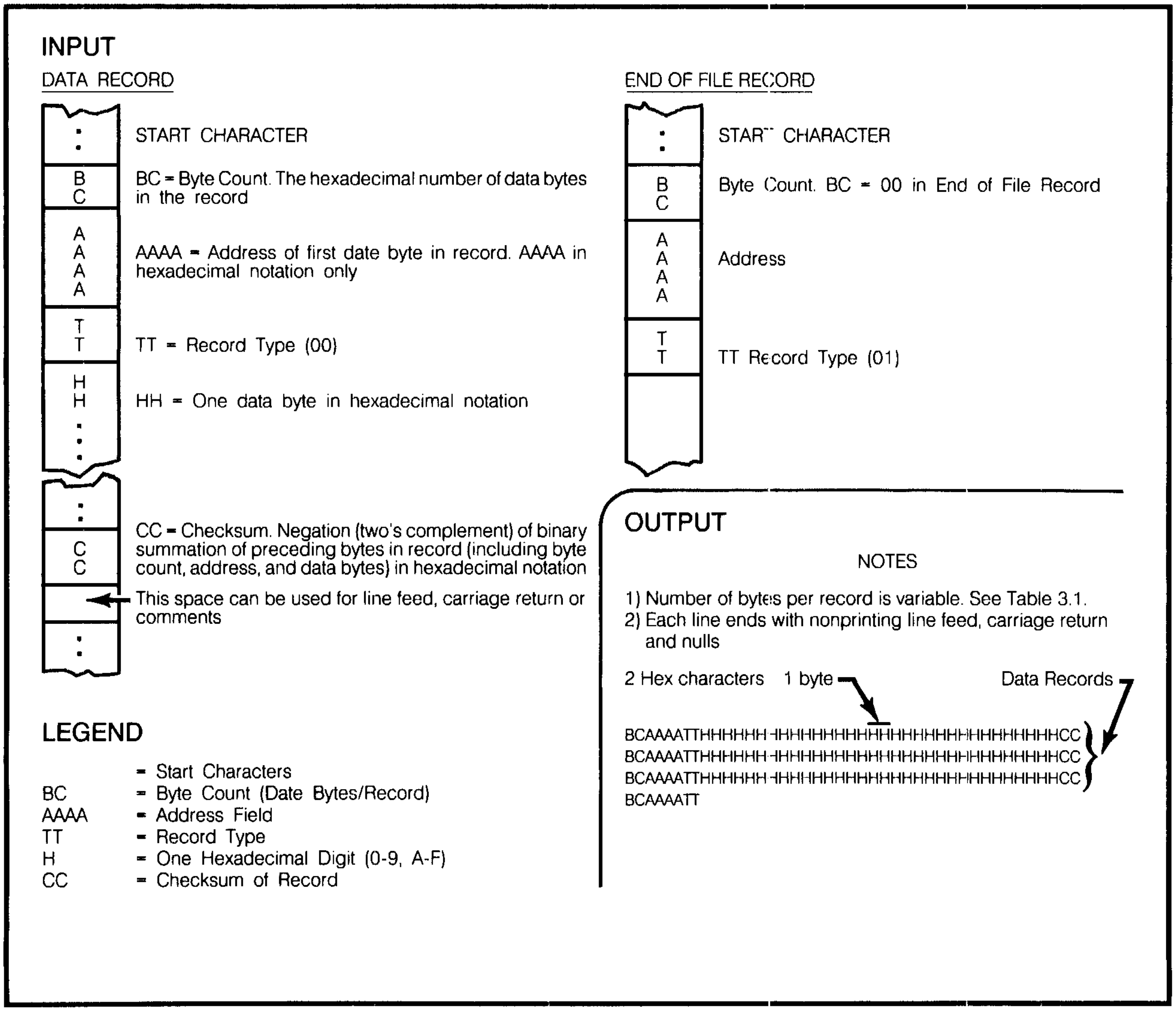
Motorola Exorcisor Format

Figure 11. Specifications for Motorola Exorciser/16-BM Data Files/ Copyright 1983, Data I/O Corporation; reprinted by permission.



Intel Intellec Format

Figure 12. Specifications for Intel Intellec/8/MDS Data Files/ Copyright 1983, Data I/O Corporation; reprinted by permission.

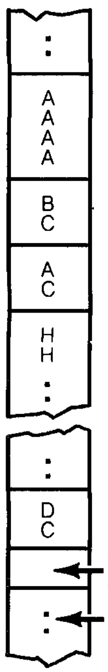


Signetics/Absolute Object File Format

Figure 13. Specifications for Signetics/Absolute Object Data Files Copyright 1983, Data I/O Corporation; reprinted by permission.

INPUT

DATA RECORD



START CHARACTER

AAAA - Address of first date byte in record. AAAA in hexadecimal notation only

BC - Byte Count. The hexadecimal number of data bytes in the record

AC - Address Check. Every byte is exclusive O Red with the previous byte, then rotated left one bit.

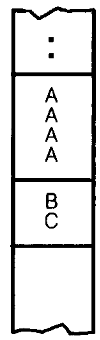
HH - One data byte in hexadecimal notation

DC - Data Check. Every byte is exclusive O Red with the previous byte, then rotated left one bit.

This space can be used for line feed, carriage return or comments

(Beginning of next record)

END OF FILE RECORD



START CHARACTER

Address

Byte Count. BC - 00 in End of File Record

LEGEND

	- Start Characters
AAAA	- Address Field
BC	- Byte Count (Date Bytes/Record)
AC	- Address Check. Checksum of address and byte count
HH	- Two Hexadecimal Digits (0-9, A-F)
DC	- Data Check. Checksum of data in record

OUTPUT

NOTES

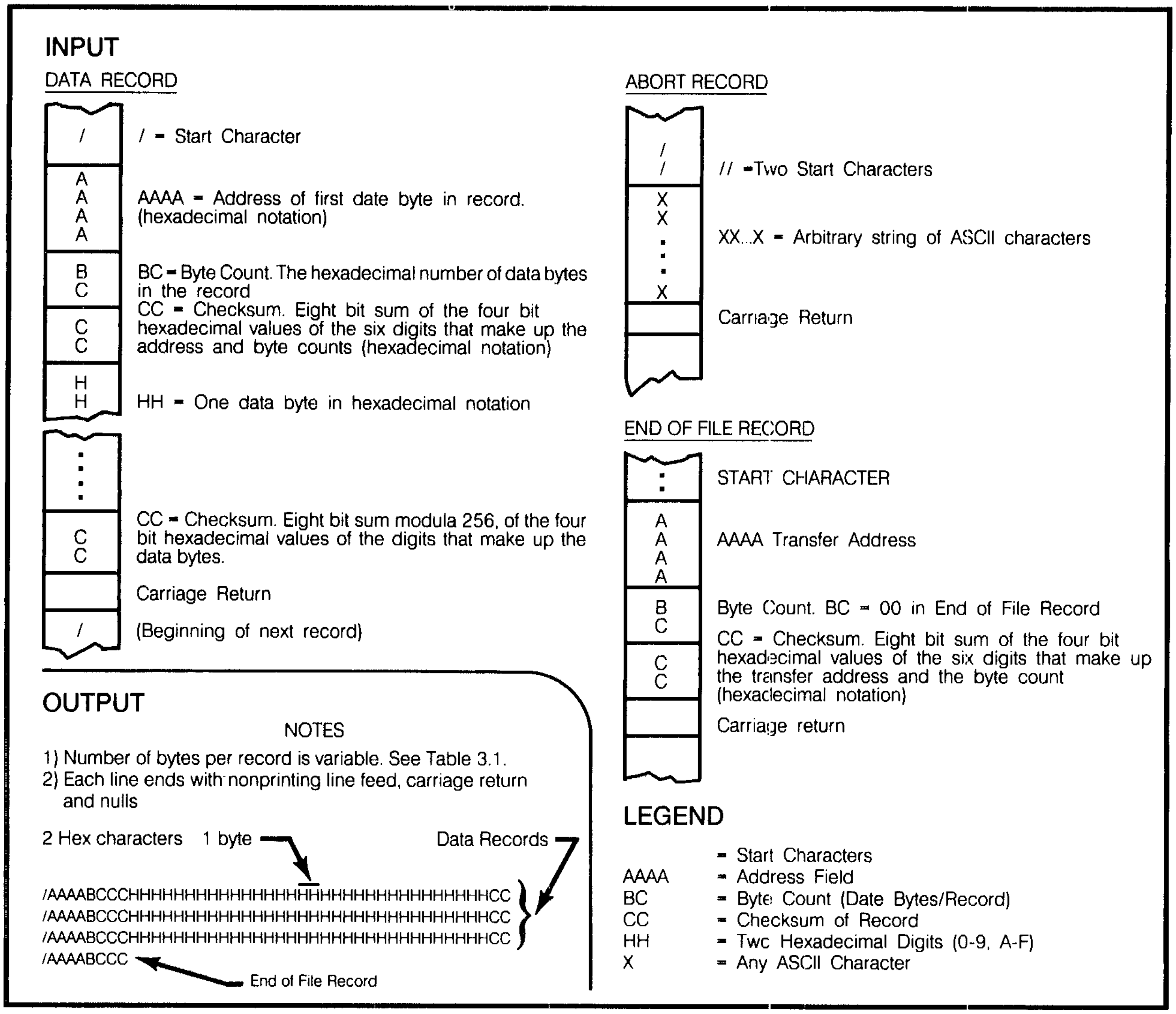
1) Number of bytes per record is variable. See Table 3.1.
2) Each line ends with nonprinting line feed, carriage return and nulls

2 HEX characters 1 byte — Data Records

```
AAAABCACHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
AAAABCACHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
AAAABCACHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
AAAABCAC
```

Tektronix Hexadecimal Format

Figure 14. Specifications for Tektronix Hexadecimal Data Files Copyright 1983, Data I/O Corporation; reprinted by permission.



Extended Tekhex Format

Copyright 1983, Tektronix; reprinted by permission

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.

ITEM	NUMBER OF ASCII CHARACTERS	DESCRIPTION
%	1	A permit sign specified 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 table gives the format of a data block and a termination block.

Extended Tekhex Data Block Format

ITEM	# OF ASCII CHARACTERS	DESCRIPTION
Header	6	Standard header field Block Type = 6
Load Address	2 to 17	The address where the object code is to be loaded: a variable-length number.
Object	2n	n bytes, each represented as two hex digits.

Extended Tekhex Termination Block

Header	6	Standard header field Block type = 8.
Transfer Address	2 to 17	The 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 an 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 you 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 fields 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: 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: 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 could be encoded in a single 96-character block. It is divided into two blocks for purposes of illustration.

Figure 13. Extended Tekhex Data Block

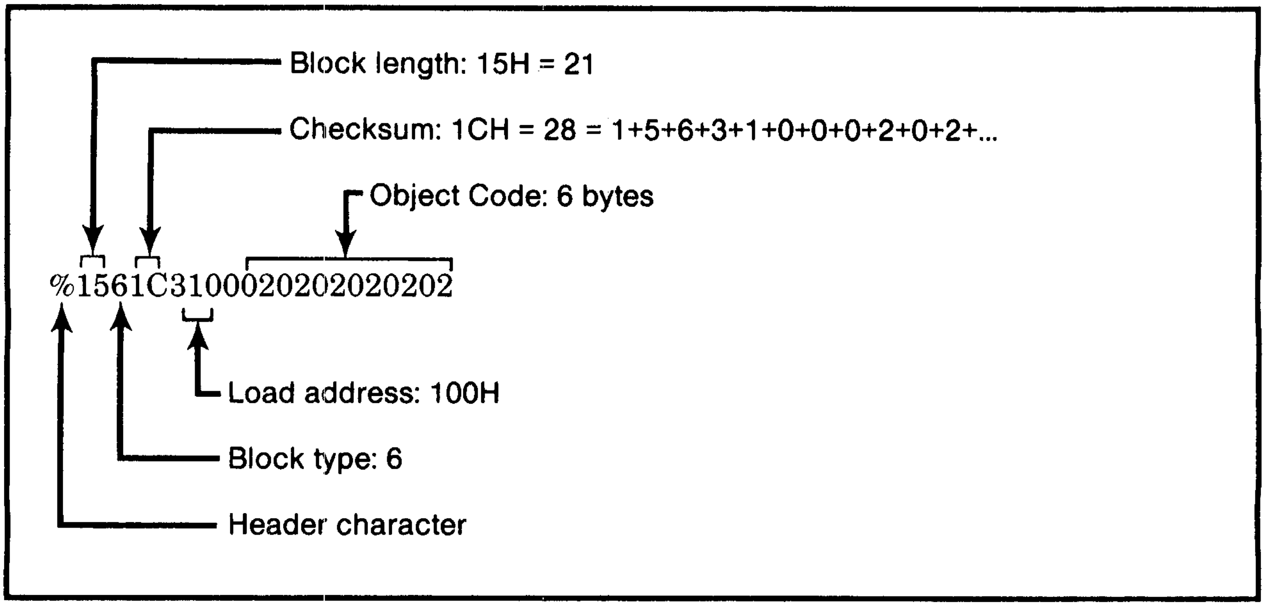


Figure 14. Extended Tekhex Termination Block

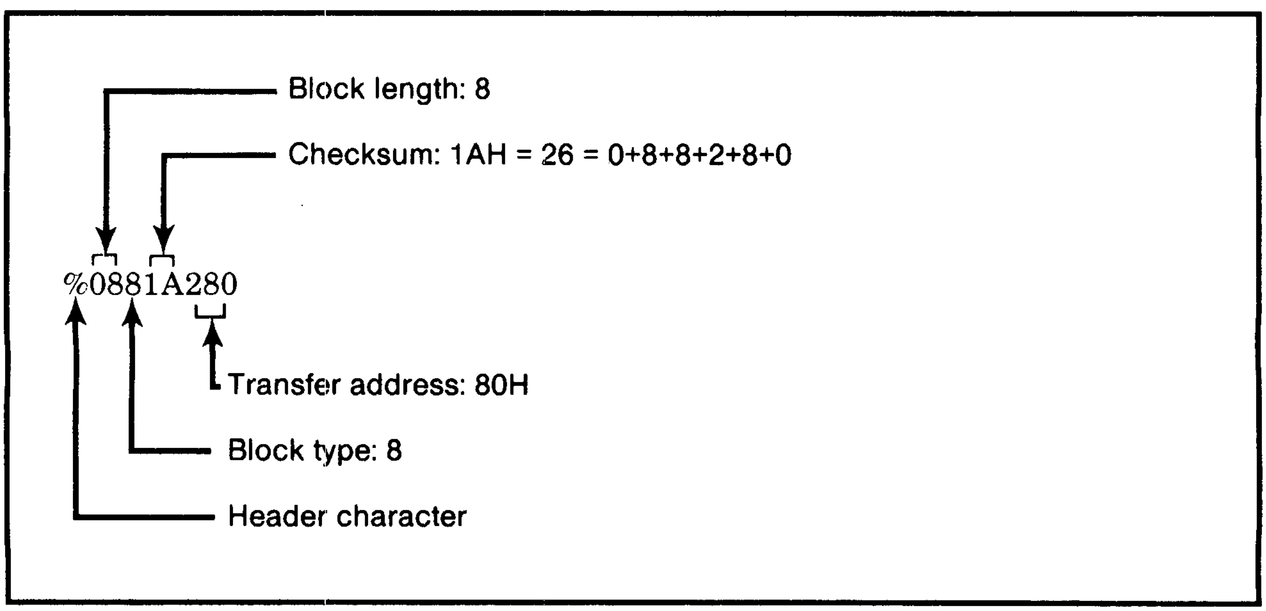
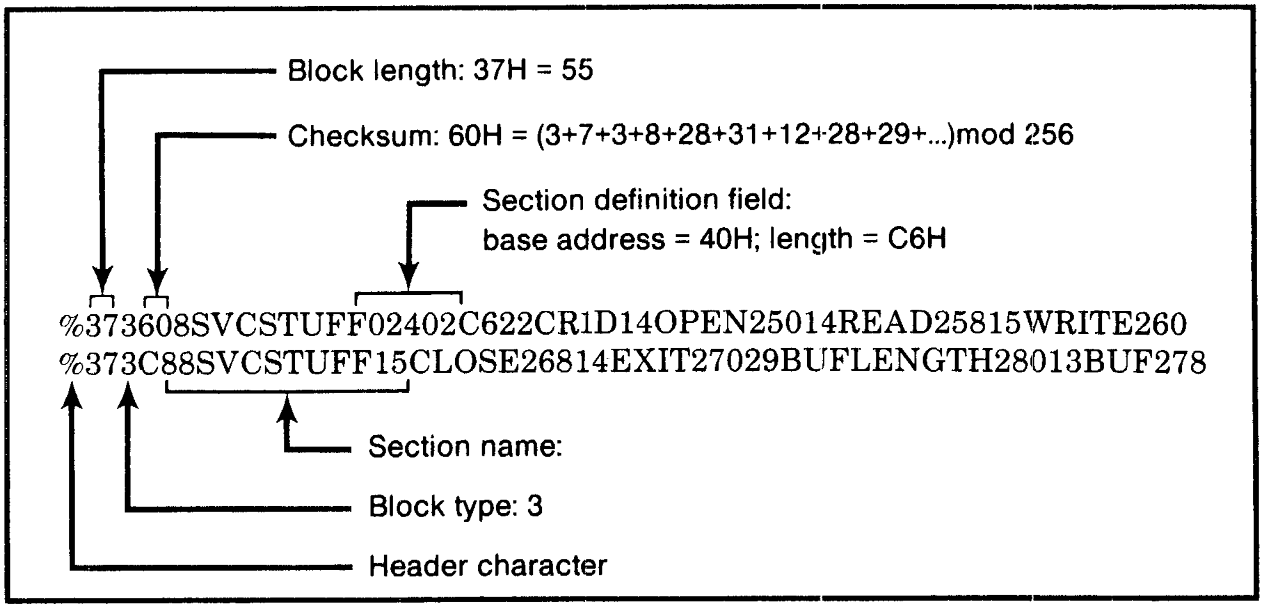


Figure 15. Extended Tekhex Symbol Block



Motorola S-Record Format

S-RECORD CONTENT

When viewed by the user, S-records are essentially character strings made of several fields which identify the record type, record length, memory address, code/data, and checksum. Each type of binary data is encoded as a 2-character hexadecimal number: the first character representing the high-order 4 bits, and the second the low-order 4 bits of the byte.

The 5 fields which comprise an S-record are: type, length, address, code/data and checksum.

The fields are composed as follows:

FIELD	PRINTABLE CHARACTERS	CONTENTS
type	2	s-record type -- S0, S1, etc.
record length	2	The count of the character pairs in the record, excluding the type and record length.
address	4, 6, or 8	The 2-, 3-, or 4-byte address at or which the data field is to be loaded into memory.
code/data	0-2n	From 0 to n bytes of executable code, memory-loadable data, or descriptive information. For compatibility with teletypewriters, some programs may limit the number of bytes to as few as 28 (56 printable characters in S-record).
checksum	2	The least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields.

Each record may be terminated with a CR/LF/NULL. Additionally, an S-record may have an initial field to accommodate other data such as line numbers generated by some time-sharing systems.

Accuracy of transmission is ensured by the record length (byte count) and checksum fields.

S-RECORD TYPES

Eight types of S-records have been defined to accommodate the several needs of the encoding, transportation, and decoding functions. The various Motorola upload, download, and other file-creating or debugging programs, utilize only those S-records which serve the purpose of the program. For specific information on which S-records are supported by a particular program, the user's manual for that program must be consulted.

An S-record format module may contain S-records of the following types:

S0 The header record for each block of S-records. The code/data field may contain any descriptive information identifying the following block of S0-records. Under VERSAdos, the resident linker's IDENT command can be used to designate module name, version number, revision number, and description information which will make up the header record. The address field is normally zeros.

S1 A record containing code/data and the 2-byte address at which the code/data is to reside.

S2 A record containing code/data and the 3-byte address at which the code/data is to reside.

S3 A record containing code/data and the 4-byte address at which the code/data is to reside.

S5 A record containing the number of S1, S2, and S3 records transmitted in a particular block. This count appears in the address field. There is no code/data field.

S7 A termination record for a block of S3 records. The address field may optionally contain the 3-byte address of the instruction to which control is to be passed. There is no code/data field.

S8 A termination record for a block of S2 records. The address field may optionally contain the 3-byte address of the instruction to which control is to be passed. There is no code/data field.

S9 A termination record for a block of S1 records. The address field may optionally contain the 2-byte address of the instruction to which control is to be passed. Under VERSAdos, the resident linker's ENTRY command can be used to specify this address. If not specified, the first entry point specification encountered in the object module input will be used. There is no code/data field.

Only one termination record is used for each block of S-records. S7 and S8 records are usually used only when control is to be passed to a 3- or 4- byte address. Normally, only one header record is used, although it is possible for multiple header records to occur.

CREATION OF S-RECORDS

S-record-format programs may be produced by several dump utilities, debuggers, VERSAdos' resident linkage editor, or several cross assemblers or cross linkers. ON EXORmacs, the Build Load Module (MBLM) utility allows an executable load module to be built from S-records; and has a counterpart utility in BUILDS, which allows an S-record file to be created from a load module.

Several programs are available for downloading a file in S-record format from a host system to an 8-bit microprocessor-based or 16-bit microprocessor-based system. Programs are also available for uploading an S-record file to or from an EXORmacs system.

Example: Shown below is a typical S-record-format module, as printed or displayed:

```
S00600004844521B
S1130000285F245F2212226A000424290008237CA
S11300100002000080008262900185381234100181 3
S113002041E9000084E42234300182342000824A952
S107003000144Ed492
S9030000FC
```

The module consist of one S0 record, four S1 records, and an S9 record.

The S0 record is comprised of the following character pairs:

S0	S-record type S0, indicating that it is a header record.
06	Hexadecimal 06 (decimal 6), indicating that six character pairs (OR ASCII bytes) follow.
00+ 00	Four-character 2-byte address field, zeros in this example.
48 44+ 52	ASCII H, D, and R - "HDR".
1B	The checksum.

The first S1 record is explained as follows:

S1	S-record type S1, indicating that it is a code/data record to be loaded/verified at a 2-byte address.
13	Hexadecimal 13 (decimal 19), indicating that 19 character pairs, representing 19 bytes of binary data, follow.
00+	Four-character 2-byte address field; hexadecimal address
00	0000, where the data which follows is to be loaded.

The next 16 character pairs of the first S1 record are the ASCII bytes of the actual program code/data. In this assembly language example, the hexadecimal opcodes of the programs are written in sequence in the code/data fields of the S1 records:

```
OPCODE          INSTRUCTION

285F            MOVE.L     (A7) +,A4
245F            MOVE.L     (A7) +,A2
2212            MOVE.L     (A2),D1
226A0004        MOVE.L     4(A2),A1
24290008        MOVE.L     FUNCTION(A1),D2
237C            MOVE.L     #FORCEFUNC,FUNCTION(A1)
o               (The balance of this code is continued in the code/data
                fields of the remaining S1 records, and stored in memory
                location 0010, etc.)
2A              The checksum of the first S1 record.
```

The second and third S1 records each also contain $13 (19) character pairs and are ended with checksums 13 and 52 respectively. The fourth S1 record contains 07 character pairs and has a checksum of 92.

The S9 record is explained as follows:

```
S9  S-record type S9, indicating that it is a termination record.
03  Hexadecimal 03, indicating that three character pairs (3 bytes) follow.
00  The address field, zeros.
FC  The checksum of the S9 record.
```

Each printable character in an S-record is encoded in hexadecimal (ASCII in this example) representation of the binary bits which are actually transmitted.

SUBJECT INDEX

- D -