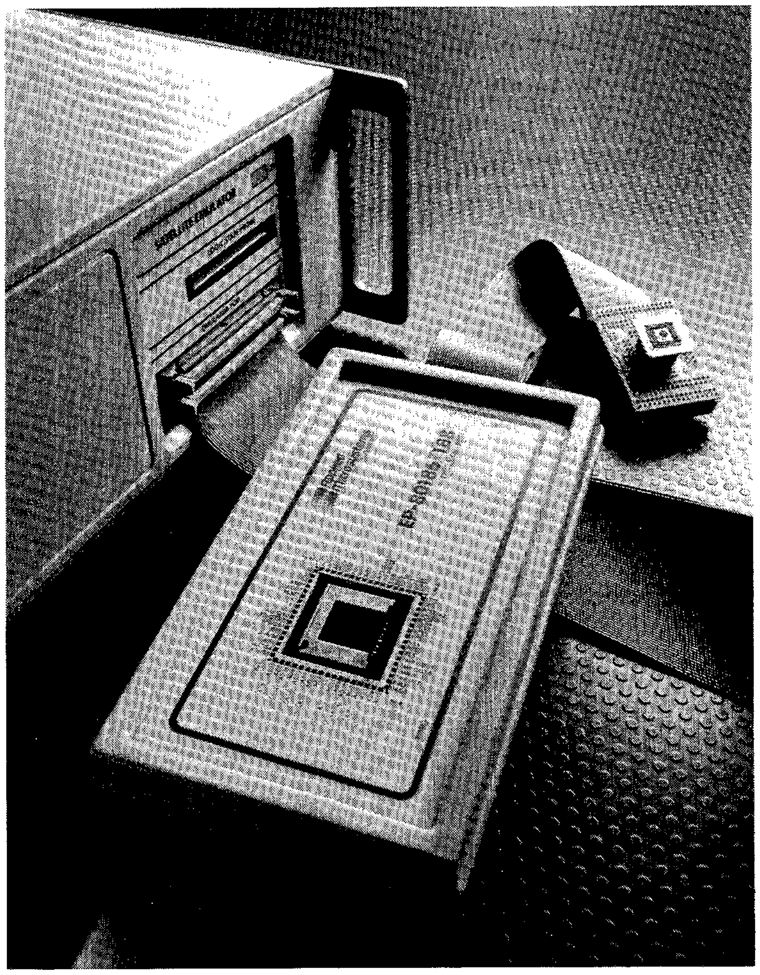
Applied
Microsystems
Corporation



ES 1800 Satellite Emulator Reference Manual For the 80186/188 Microprocessors

Applied
Microsystems
Corporation

ES 1800 Satellite Emulator Reference Manual For the 80186/188 Microprocessors

P/N 920-11437-04
March 1988

Table of Contents

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 80186/88 microprocessor-based systems. This manual assumes the user is familiar with the terminology and capabilities of the 80186/88 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 80186/88 Microprocessors

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 80186/88 microprocessor only. For more complete information on this chip, refer to the Intel hardware reference manual *iAPX 86/88, 186/188 User's 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 80186/88 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 80186/88. 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** are quick references to ES Language mnemonics and 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 in Figure 1) 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.

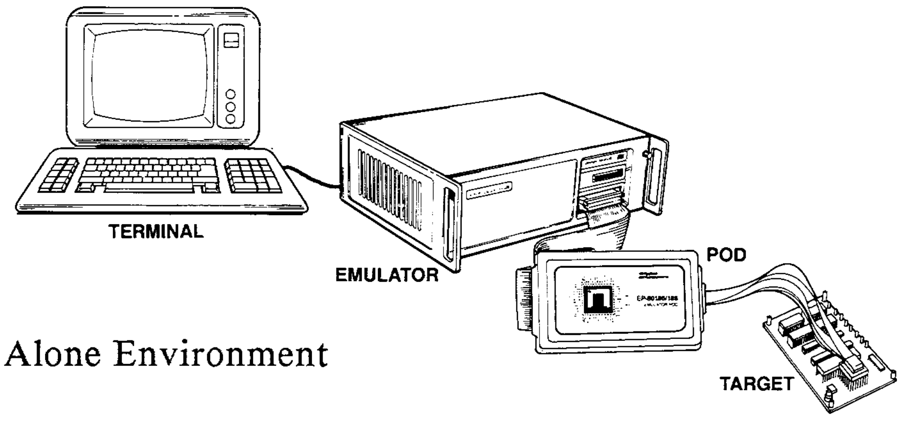
The Emulator can also use data stored in a host development system by setting up a hosted environment (refer to diagram in Figure 1). The Emulator is still under the direct control of the CRT but can load data from the host system's data files.

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.

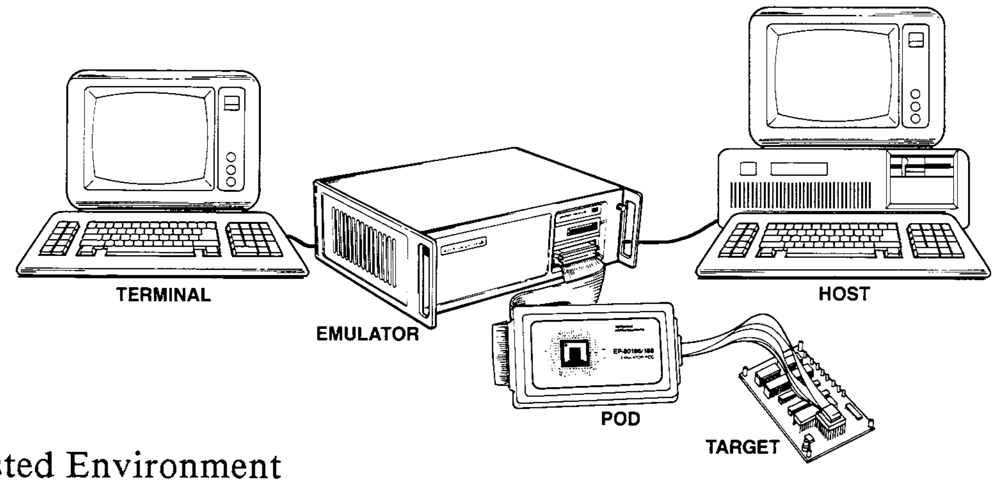
Software for driving the Emulator is available from Applied Microsystems Corporation for the IBM PC and compatibles, SUN, APOLLO and VAX.

The Emulator can also be totally controlled by a host system. This hosted software environment (refer to diagram in Figure 1) requires special host resident software. Drivers and high level debuggers are available from Applied Microsystems for most languages and host systems.

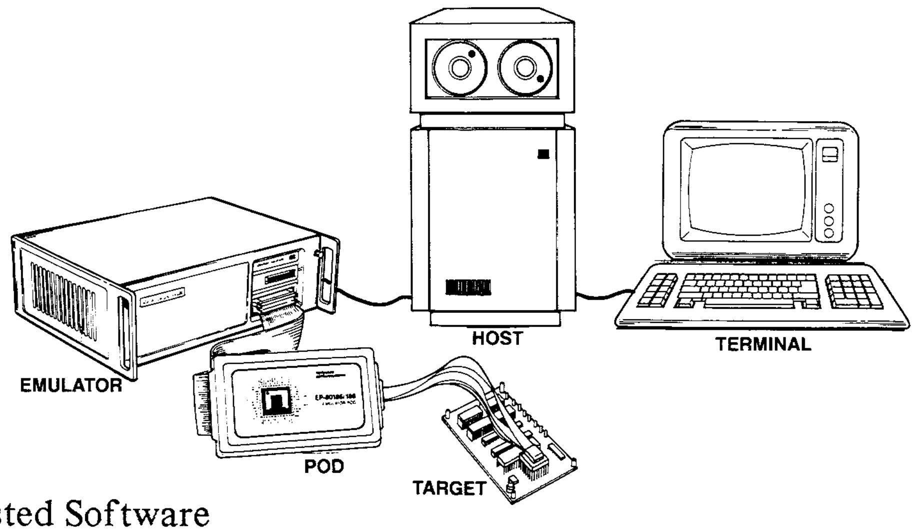
Figure 1. Environments



Stand Alone Environment



Hosted Environment



Hosted Software

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. Overlay memory can then be used to develop code as if a target system were attached. In this mode a 4 MHz clock is supplied to the CPU through a divide by 2 network. The CPU runs at 2 MHz. Unterminated inputs are set inactive.

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

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 3-5 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 3-7 for location illustration.

   NOTE: If you are using an early ES1800 model, the above comment may not apply. Follow the instructions provided at the time of purchase.

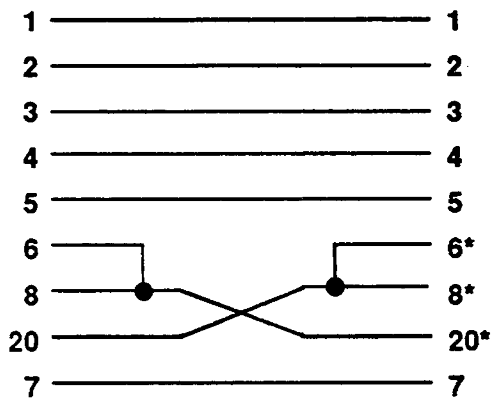
10. Replace front panel and attach the correct pod assembly (see page 3-8). A pod assembly must be connected to the Emulator even if you are not connecting it to a target system.

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

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

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.

Target System Setup

1. Check that the target has a 68 contact leadless chip carrier socket. An adaptor, Part No. 210-00023-00, is available for plastic leaded chip carriers.
2. Using an ohmmeter, check that a good ground exists at the microprocessor socket. Measure from pin 26 and 60 to power supply ground on the target board.
3. Verify that all the power supplies in the target system are functioning properly.
4. Check for a valid clock signal at the target microprocessor socket.
5. Turn off target system power and Emulator power.
6. Plug in the probe tip. (See page 3-14 for probe tip precautions.)

Power-Up Sequence

TARGET SYSTEM PRESENT

1. Turn on the target system.
2. Turn on the Emulator.
3. Reset the target system (see page 6-13).

NO TARGET SYSTEM

1. Verify that the pod is connected to the Emulator (see page 3-8).
2. Be sure there is nothing in contact with the probe tip.
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 5-25). 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 ($\boxed{>}$).

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.

```
>MAP 0 TO XXXX
```

(Where *XXXX* is the ending address (in hex) of the amount of RAM overlay 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
```

For more information, refer to page 5-56.

3. TEST RAM

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 6-50.

4. ENTER PROGRAM

Enter a short program by invoking the line assembler and entering 8086 op codes (see page 6-30).

```
  >ASM 10
 **** 8086/88/186/188 LINE ASSEMBLER VX.XLA ****
 CSEG = XXXX
  0010>NOP
  0011>/
  0012>/
  0013>/
  0014>/
  0015>JMP 10H
  0017>X
```

NOP is a null operation. Each time you type the slash ($\boxed{/}$), you repeat the previous command, so you have entered the equivalent of five lines of NOPs. The $\boxed{X}$ 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 6-7.

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 0:15.

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 CTL THEN BRK
  >DC1=0XXXX
  >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.

10. INITIALIZE PERIPHERAL CONTROL REGISTERS

1. Set up the PCB relocation register. If you do not relocate the peripheral control block from $FF00 in I/O space, then go to step 2.

Enter:

```
>REL = <register value>
```

Refer to the Intel *iAPX 86/88, 186/188 User's Manual* for the proper way to set up the PCB relocation register.

2. Set up the on-chip chip select peripheral. If you do not use on-chip chip selects, then go to step 3.

   Enter:

   ```
   >ON RCS
   ```

   With RCS set to ON, the following will be true:

   Pause-to-run transitions will write the Emulator chip select PCB values into the target PCB.

   Run-to-pause transitions will read the Emulator chip select PCB values from the target PCB.

   Enter:

   ```
   >UMCS = <register value>
   >LMCS = <register value>
   >MPCS = <register value>
   >MMCS = <register value>
   >PACS = <register value>
   ```

   Refer to the Intel *iAPX 86/88, 186/188 User's Manual* for the proper way to set up the register.

3. Set up the on-chip DMA peripheral. If on-chip DMA circuitry is not used, then go on to step 4.

Enter:

```
>USRCO = <register value>
>SRCO  = <register value>
>UDSTO = <register value>
>DSTO  = <register value>
>XCO   = <register value>
>CWO   = <register value>
```

Refer to the Intel *iAPX 86/88, 186/188 User's Manual* for the proper setup.

If you do not need DMA active while paused, then go on to step 4.

Enter:

```
>ON DME
```

4. Set up the on-chip timer peripheral. If on-chip timer circuitry is not used, then go on to step 5.

   Enter:

```
>TCO  = <register value>
>TC1  = <register value>
>TC2  = <register value>
>MA0  = <register value>
>MA1  = <register value>
>MA2  = <register value>
>MB0  = <register value>
>MB1  = <register value>
>MB2  = <register value>
>MCW0 = <register value>
>MCW1 = <register value>
>MCW2 = <register value>
```

Refer to the Intel *iAPX 86/88, 186/188 User's Manual* for the proper setup.

If you need a timer circuit active while paused, then turn on the appropriate emulator software switch, as follows:

```
>ON TE0
>ON TE1
>ON TE2
```

This will turn on timers zero, one, and two respectively.

5. Set up the on-chip interrupt control peripheral. If on-chip interrupt control circuitry is not used, then proceed to step 6.

   Enter:

```
>INT0  = <register value>
>INT1  = <register value>
>INT3  = <register value>
>EOI   = <register value>
>POL   = <register value>
>POL   = <register value>
>MSK   = <register value>
>PLM   = <register value>
>ISV   = <register value>
>IRQ   = <register value>
>IST   = <register value>
>TCR   = <register value>
>DMA0  = <register value>
>DMA1  = <register value>
>DMA2  = <register value>
```

   Refer to the Intel *iAPX 86/88, 186/188 User's Manual* for the proper setup.

6. Display the status of the PCB registers.

```
>PCB
```

   The screen displays the current contents of the PCB registers.

7. Set up overlay and a minimal program. This step assumes you have neither target memory nor a valid program located at the startup location (*FFFF0). If you have target memory and a valid program, then go on to step 8.

   Enter:

```
>MAP $FF800;DM
```

   This maps in overlay from $FF800 to $FFFFF and displays the memory map.

   Enter:

```
>ON RDY
```

   This ensures that reads and writes to overlay memory use the Emulator's internal ready signal.

   Enter:

```
>ASM
```

   This invokes the single-line assembler to enter a sequence of NOP instructions.

Enter:

```
>CSEG = 0FFFF
```

This sets the assembler to an absolute address of $FFFF0.

Enter:

```
>NOP
>NOP
>NOP
```

This throw-away program initializes the on-chip peripheral circuitry.

Enter:

```
>X
```

This exits the assembler.

8. Activate the on-chip peripherals. The following tasks should have been accomplished before reaching this point:

   - The state of all on-chip peripherals should have been set up via the PCB registers.
   - The Emulator's ON and OFF software switches have been properly set up.
   - A program resides at the start up location ($FFFF0).

Enter:

```
>AC1 = <stopping point>
```

This defines the stopping point of the program which should follow the initialize section.

The on-chip peripherals are activated by either a read from, or write to appropriate registers. The setting of the Emulator's switches to ON guarantees the chosen peripheral registers will be written and read following the execution of at least one instruction cycle. Therefore, set up AC1, as either:

```
AC1 = $FFFF2
```

if manually initializing and using the NOP program in step 7, or

```
AC1 = <stopping point>
```

if using your own PCB initializing program.

Enter:

```
>WHEN AC1 THEN BRK
```

This allows a breakpoint when AC1 is recognized during emulation.

Enter:

```
>RST;RBV
```

RST sends a reset signal to the target system via the RESET OUT line.

RBV sets CS:IP registers to the absolute address of $FFFF0, activates the Event Monitor System, and initiates a real-time run.

SECTION 3

Table of Contents

Hardware

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!

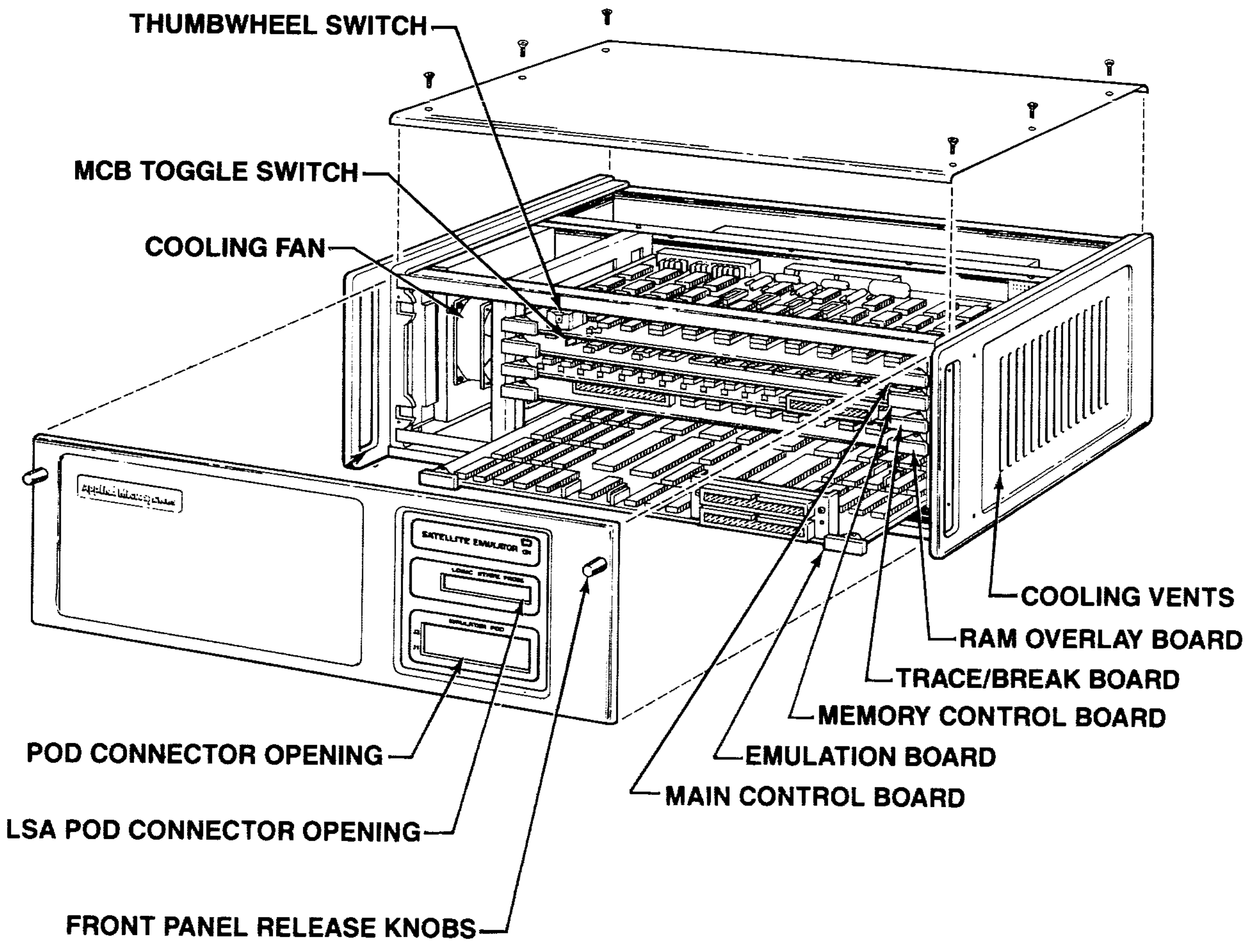
Failure to ground the system properly may create a shock hazard.

Emulator Control Boards

Removing the front panel of the Emulator chassis exposes the chassis card cage as shown in Figure 2. 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.
RAM Overlay Board(s)	The RAM overlay 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 page 1-4).

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

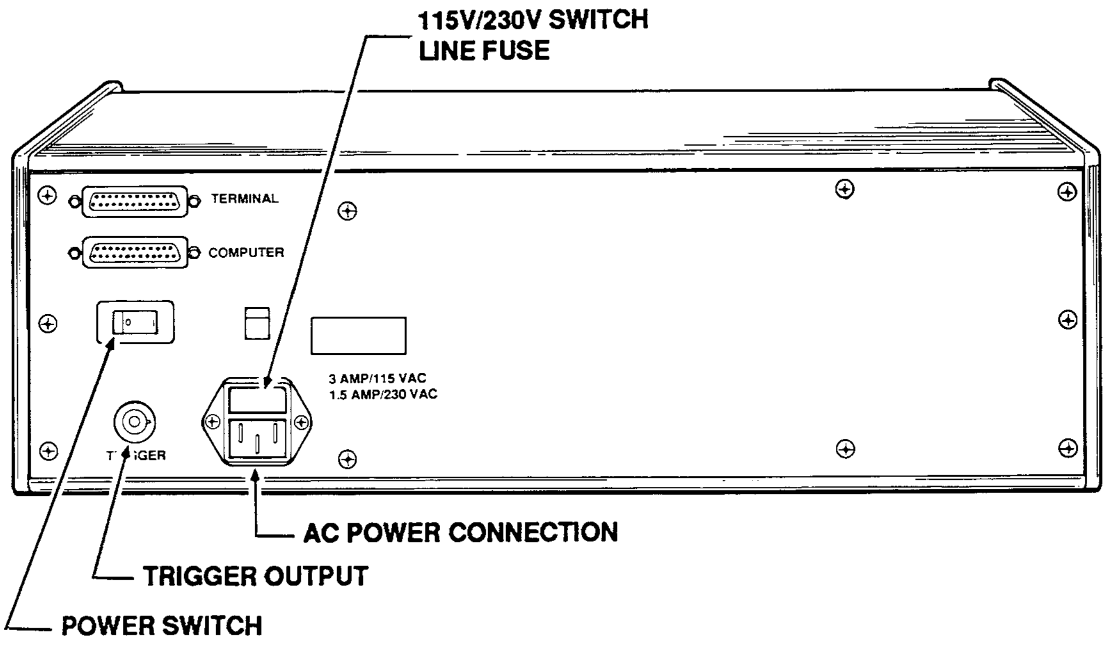
Before powering up, two items should be checked:

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 Sequence, page 2-5.)

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 counterclockwise.

Figure 3. Rear Panel



Pod Assembly

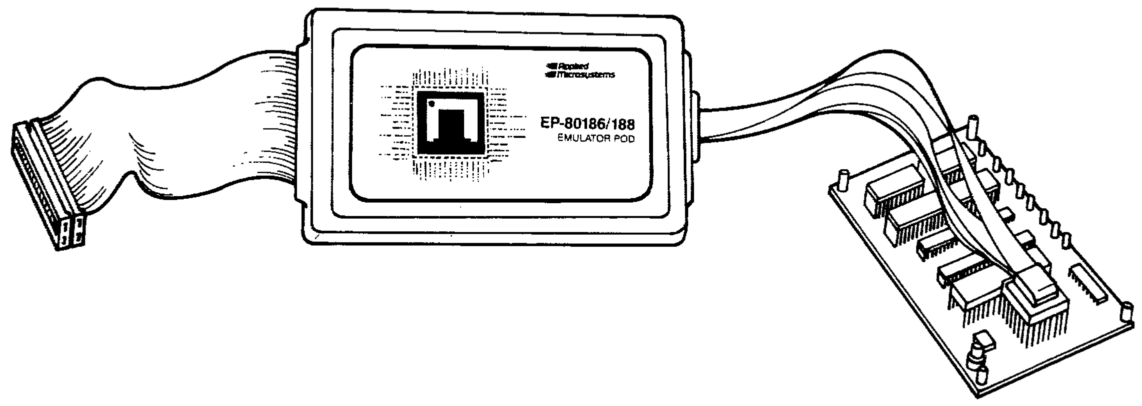
The pod assembly is the link between the ES1800 Emulator and the target system. A 40-inch ribbon cable assembly connects the pod assembly to the Emulator board. An 11-inch ribbon cable assembly ends in a probe tip that is normally inserted into the microprocessor socket in the target system.

The proper pod assembly is determined by the microprocessor being emulated.

To install the probetip, remove the retainer clip from the LCC socket, place the probe tip in the socket as you would the microprocessor, then replace the retainer clip. Always check that pin 1 is aligned correctly.

On older model probes, the tip may be removed using the thumbscrew, to allow for easier access to the retainer clip.

Figure 4. Pod Assembly

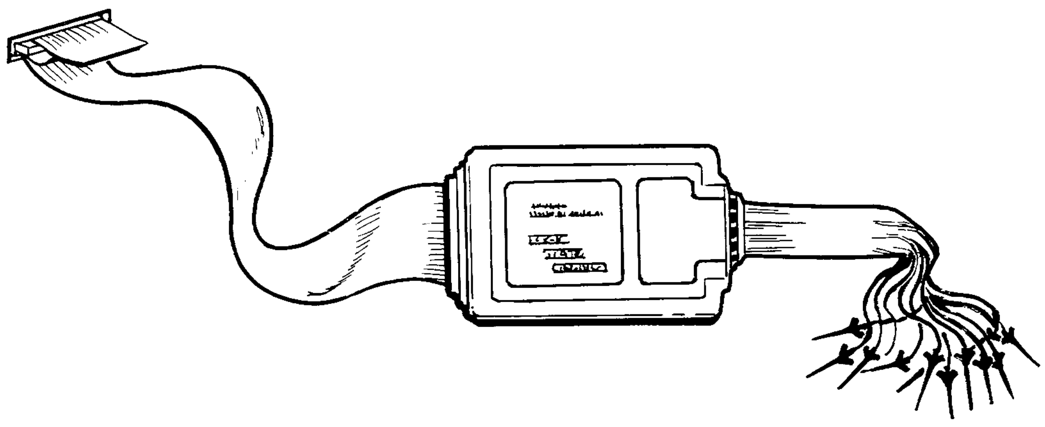


Logic State Analyzer (LSA) Assembly

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. This allows triggering an oscilloscope or external logic analyzer for events that are set up in the Event Monitor System with a "then TGR" statement.

Figure 5. Logic State Analyzer Pod Assembly



Serial Ports

Both the terminal port and the computer port 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 page 5-30, 5-35, 5-36.)
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.

Both Emulator serial ports use the same pin assignment. All pin assignments and voltage levels conform to Electronics Industries Association (EIA) RS232C standards. The following chart lists the signals present 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 (DC1) and XOFF (DC3). 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 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!

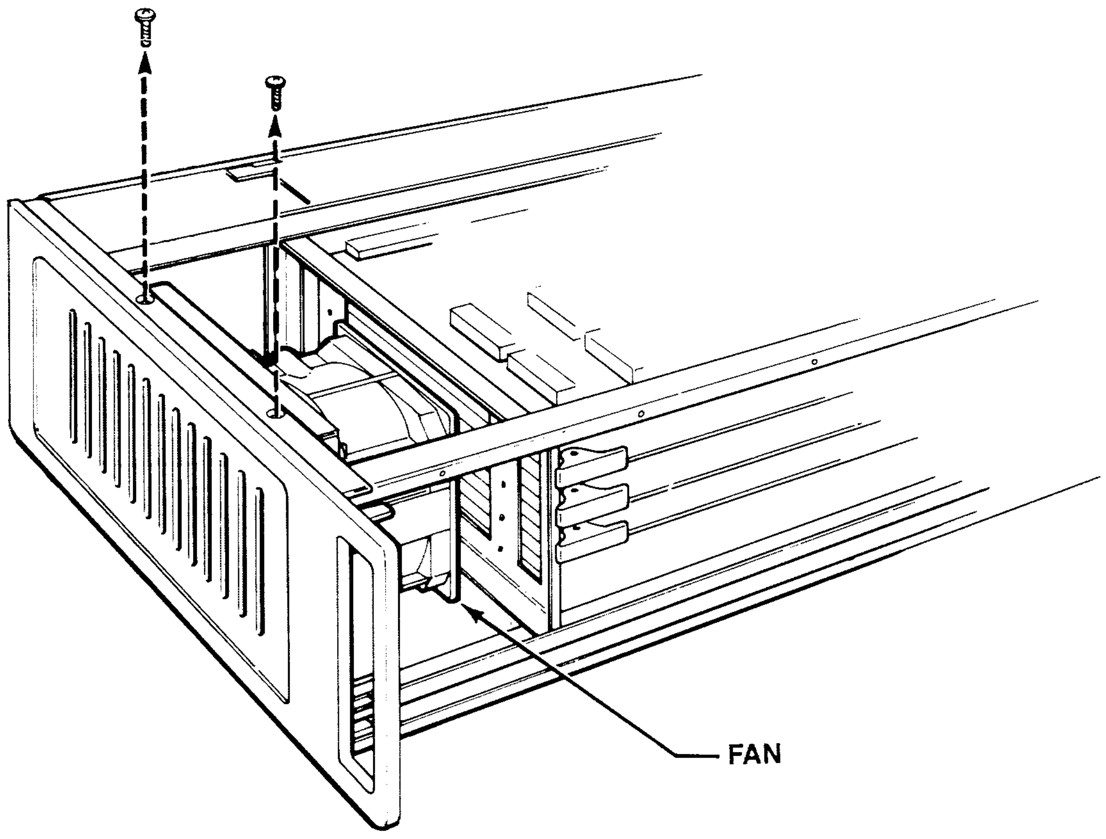
   Electrical shock and moving fan parts are dangerous. Make sure you unplug the unit before proceeding.

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.)

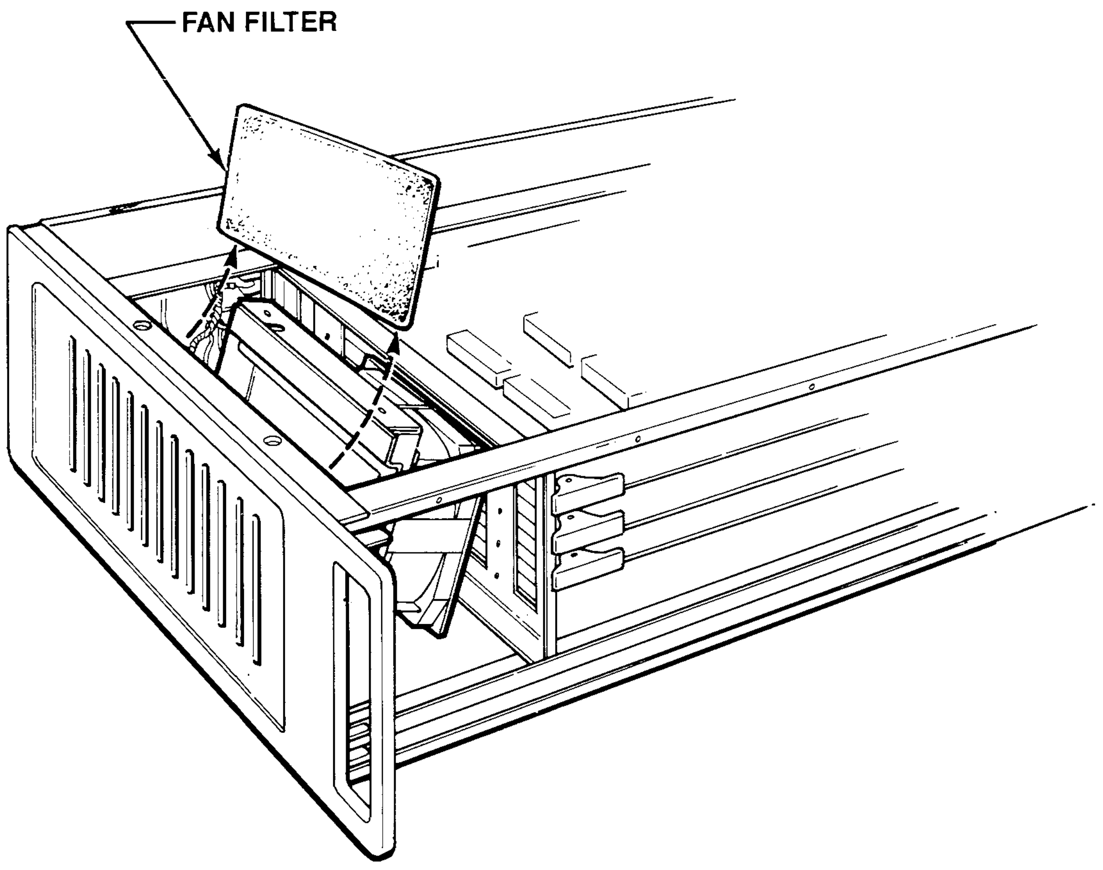
4. Unscrew the two screws at the top of the chassis which hold the fan in place.

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
          Short cable set
          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 5-124). You will be asked for the revision number and serial number when you call. Also, record the serial number located on the back of the chassis.

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. Emulator and target system not compatible
	3. LDV not executed before RUN (vector not loaded).
Emulator does 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

ES1800 Emulator Specifications

INPUT POWER

Standard	90 to 130 VAC
	47 to 63 Hz
	consumption less than 130W
Optional	180 to 260 VAC
	47 to 63 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 in. 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

ES LANGUAGE

Structure of the ES Language

The command language used to control 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 structure of the ES language 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 ($\boxed{<>}$) are mandatory and must be entered as part of the command.

Items shown in square brackets ($\boxed{[]}$) 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 $\boxed{?}$ 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 & OFF`
<^>	`DC2.3 ^ OFF00`
Mul.Op	
<*>	`2 * 3`
</>	`OFAC / %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 4-23 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 5-102).

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 ($\boxed{*}$) 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 6-40), entering a $\boxed{\text{RETURN}}$ on an empty line causes the next location to be read. Entering a value followed by $\boxed{\text{RETURN}}$ will cause that value to be written to memory. I/O mode (page 6-38), the line assembler (see page

6-30), memory disassembler (page 6-36), and main help menu (page 4-20) 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 ($\boxed{/}$) will cause the previously entered command line to be repeated.

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

This command single steps three times.

The comma ($\boxed{,}$) executes macro 1, and the period ($\boxed{.}$) 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 ($\boxed{?}$) also has two uses. It can be entered after the command interpreter detects an error and beeps. If you are "beeped," enter a $\boxed{?}$ and the command processor will give you an error message describing the problem it detected.

A $\boxed{?}$ entered at any other time (i.e., not after an error), causes a two-page Help Menu to be displayed. A $\boxed{\text{RETURN}}$ 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.

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. It is legal to have multiple indirections, e.g., `@@GR0 = @@@(SS:SP + 6)`. Byte mode and word mode affect the length of data transferred to or from the target by the `@` operator.

The `:` operator mimics the arithmetic combination of segment and pointer registers in the 80186/88 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. 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., $\boxed{\texttt{CS:IP}}$ is fine but $\boxed{\texttt{CS:(IP+3)}}$ 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., $\boxed{\texttt{2+-1}}$ is illegal, but $\boxed{\texttt{2+(-1)}}$ and $\boxed{\texttt{-1+2}}$ 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 you have the symbolic debug option installed in your ES1800 Emulator, you can use symbolic references. Every symbol must begin with a single quote ($\boxed{\texttt{'}}$). Symbols are composed of 1 to 64 printable characters followed by a space or $\boxed{\text{RETURN}}$. 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 ($\boxed{*}$) 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 5-80 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. $\boxed{\texttt{1234 DC 0FF0}}$ is explicit. $\boxed{\texttt{1XX4}}$ is equivalent to $\boxed{\texttt{1FF4 DC 0FF0}}$. Don't cares are useful for setting the Event Monitor System event comparators (see page 7-2).

The third number type is range. A range is specified by entering a start address and a length or an endpoint. $\boxed{\texttt{200 LEN 20}}$ is the same as $\boxed{\texttt{200 TO 21F}}$. Ranges can be either internal (default) or external. An explicit range type can be specified by using the prefix **IRA** or **XRA**. $\boxed{\texttt{0 LEN 100}}$ is the same as $\boxed{\texttt{IRA 0 LEN 100}}$. The $\boxed{\texttt{!}}$ operator inverts the type of a range value. $\boxed{\texttt{!(0 LEN 100)}}$ is the same as $\boxed{\texttt{XRA 0 LEN 100}}$, 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 5-70 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.

Assignment Command

Svalues are the names of registers or symbolic references. 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, commands such as `@@SS:SP = 0` are legal and useful.

Registers are grouped into three types: integer only, don't dare, 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 5-70 for a list of all registers and their data types.

The indirection operator $\boxed{@}$ allows expressions to include values transferred to or from the target system memory address space. The expression becomes the address of a target system byte.

In physical mode, more than one $\boxed{@}$ 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 $\boxed{@}$ 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.

Parentheses may be used to affect the processing of the $\boxed{@}$ operator:

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

In the first example the indirection operator is applied to $\boxed{\text{GD4}}$. The command interpreter accesses the target system location pointed at by $\boxed{\text{GD4}}$, 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 $\boxed{\text{GD4}}$.

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.

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 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. 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 6-50).

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 SYSTEM             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 - DISPLAY ALL CPU REGISTERS
 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
 MMS = ALT, COD, DAT, STA              DEFINES STATUS LINES FOR MEMORY ACCESS
 X - EXIT MEMORY MODE                  M X - VIEW/CHANGE MEMORY AT X

MEMORY MAPPING:                        OVE = CD, DAT
 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/IOA + IAK/RIO/WIO/HLT/IF/RM/WM/NBC
                        - + 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
```

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.

```
**** 8086/88/186/188 LINE ASSEMBLER V2.6LA ****
CSEG = 0000
0100 >
```

The line assembler displays a 16-bit address prompt. This prompt contains an $\boxed{\text{R}}$ 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 $\boxed{\text{RETURN}}$ is entered as the only character on the line. The data field is affected by byte and word mode. If 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.

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.
Line Assembler	The line assembler has a single 16-bit address prompt. Exit by entering an $\boxed{\text{X}}$ 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 $\boxed{\text{RETURN}}$ displays the next page of disassembled memory. A $\boxed{\text{SPACE}}$ 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 $\boxed{\text{X}}$.
I/O Mode	I/O mode has an address prompt. Exit by entering an $\boxed{\text{X}}$.
Transparent Mode	No characters are generated by the ES1800. Exit by entering the two- character escape sequence (default is $\boxed{\text{ESC}}$ $\boxed{\text{ESC}}$), or reset (default $\boxed{\text{CTRL}}$ $\boxed{\text{Z}}$).

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 $\boxed{?}$ 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 $\boxed{?}$. 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., $\boxed{\text{NO TARGET CLOCK}}$ or $\boxed{\text{RESET ASSERTED}}$. 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. $\boxed{\text{DR}}$ and $\boxed{\text{BRK}}$ are both legal, but when entered together as $\boxed{\text{DR BRK}}$ 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. $\boxed{\text{6 len 1}}$ and $\boxed{\text{10 to 5}}$ are examples of invalid ranges.
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. $\boxed{\text{LDV}}$, $\boxed{\text{RNV}}$, and $\boxed{\text{RBV}}$ 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 80186/88 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.
93	INVALID CONTROL REGISTER.
94	ARGUMENT NOT SYMBOLIC. Requires a symbolic argument.

SECTION 5

Table of Contents

System Commands

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

SET COMMAND

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

(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 5-29) for information on communicating with a host computer.

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

```
  >ON
      ES SWITCH SETTINGS
 LD/SAV 4:  LOAD/SAVE SWITCH SETTINGS IN EEPROM
 EXAMPLES:  >ON BKX+CK
            >OFF FSX+CPY

 VALUE   NAME      DESCRIPTION

  OFF    BKX     BREAK ON INSTRUCTION EXECUTION (NOT PREFETCH)
  ON     CK      SELECT INTERNAL CLOCK
  OFF    CPY     COPY DATA TO TERMINAL & COMPUTER PORTS
  ON     FSX     FSI ON INSTRUCTION EXECUTION (NOT PREFETCH)
  ON     RDY     SELECT INTERNAL READY WHEN ACCESSING OVERLAY
  ON     STI     ENABLE STEP THROUGH INTERRUPTS
  OFF    DME     ENABLE DMA DURING PAUSE
  OFF    TE0     ENABLE TIMER 0 DURING PAUSE
  OFF    TE1     ENABLE TIMER 1 DURING PAUSE
  OFF    TE2     ENABLE TIMER 2 DURING PAUSE
  OFF    RCS     ENABLE CHIP SELECT REGISTERS DISPLAY
  OFF    CDH     CLEAR DHLT BIT IN IST REGISTER ON PAUSE TO RUN
  >
```

ON *<switch>*	Sets the specified switch to the ON position.

(continued)

OFF *<switch>*	Sets the specified switch to the OFF position.

Comments

Some ON/OFF switches cannot be set during run mode.

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

BKX	Break on instruction execution
CDH	Clear DMA halt
CK	Internal/external clock
CPY	Copy data to both serial ports
DME	Enable DMA
FSX	Force special interrupt on instruction execution
RCS	Read chip select control registers
RDY	Internal/external ready signal
STI	Step through interrupts
TE*<0,1,2>*	Timers

You may turn on or off multiple switches by listing them with a $\boxed{+}$ 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 5-25).

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 5-27).

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.

```
>GR0 = RDY+CK

>ON GR0
>OFF GR0
```

(continued)

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 $\boxed{.}$ and $\boxed{,}$ as the first character on each line (see page 5-104).

```
>_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.
	Default: OFF

Comments

The 80186/88 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 before the instruction is executed (it was prefetched). Set this switch to force the break to occur only on address execution.

CLEAR DMA HALT

Command	Result
ON CDH	DMA is reenabled during pause-to-run.
OFF CDH	During pause-to-run, DMA status is unchanged from status while paused. 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, page 5-38 the overlay memory section, page 5-54 and the RDY switch, page 5-20.

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 5-29 for more information on the terminal and computer ports.

Command	Result
ON DME	The DMA controllers are active during pause. The values in DMA0 and DMA1 registers are not reloaded to the physical PCB upon pause-to-run. The following also occurs: On a run-to-pause transition the IST register is copied to the internal RAM table. The DHLT bit is then cleared, causing DMA cycles to resume. All DMA cycles are directed to the target system.
OFF DME	The DMA controllers are not active during pause mode. Default: OFF

Comments

All DMA cycles are disabled immediately upon a run-to-pause transition.

If the target system uses an external dynamic memory controller for refresh, DME must be set to OFF. This prevents memory read signals from going out to the target in pause mode.

If internal DMA is used, then DME should be ON.

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.
	Default: ON

Comments

The 80186/88 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.

READ CHIP SELECT

Command	Result
ON RCS	All chip select control registers are read upon run-to-pause.
OFF RCS	The chip select control registers are only read and loaded to the internal RAM table if they have been set manually with a value during pause mode. The transition from pause to run mode causes only those chip select registers that have been modified during pause mode to reload to the physical PCB. The displayed values of chip select registers do not show what is actually in the PCB. Default: OFF

Comments

The RCS software switch does not affect the UMCS chip select control register.

Reading the chip select control registers enables their corresponding outputs. Use the RCS software switch only after the chip select control registers are set.

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 below.)

Comments

This command is valid only in pause mode.

A "ready signal" denotes the end of a memory cycle. See the Intel *iAPX 86/88, 186/188 Users 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.

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

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 6-7.

Command	Result

ON TE*<0,1,2>*	The specified PCB timer (0,1 or 2) is active during pause mode. The transition between emulation modes executes as follows:
	`>RUN TO PAUSE`
	The value in the specified timer register is loaded to internal RAM and can be modified using a `<reg> = <value>` command.
	`>PAUSE TO RUN`
	The value in the internal RAM table of the specified timer is not loaded. Reloading this value destroys the data generated during pause mode.

OFF TE*<0,1,2>*	The specified PCB timer (0,1,2) is not active during pause mode. The transition between emulation modes executes as follows:

(continued)

```
RUN TO PAUSE
```

The mode control word register (MCW0, MCW1, MCW2) for the specified timer is loaded to the internal RAM table. The timer is then disabled by clearing bit 15 of the mode control word register.

```
PAUSE TO RUN
```

The value in the internal RAM table of the specified timer is reloaded to the physical PCB. This restores the timer to its configuration when last running in the target system.

Default: OFF

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.

(continued)

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	5-3
1 - Contents of Emulator registers	5-70
2 - Event Monitor System WHEN/THEN statements	7-1
3 - Overlay map	5-54
4 - Software switch settings	5-9
5 - Macros	5-102

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 8086 Emulator board and pod.

Examples

```
>SAV 1
```

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

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 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

(continued)

Examples

```
  >LD 3
```

The overlay memory map in the EEPROM is copied into internal RAM. Use the **DM** command to verify the new map. (See page 5-55.)

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

(continued)

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 1-4 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 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.

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 5-7 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

(continued)

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.

TRANSPARENT MODE

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 5-29).

(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.

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 5-30 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 (see page 5-7. Use transparent mode (**TRA**) to verify host system format and the address in the file. (See page 5-33.)

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 5-33).

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

(continued)

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 ($\boxed{>}$) 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 $\boxed{\text{ACK \$6}}$.
- 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 5-7.)

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 $\boxed{\text{RETURN}}$ instead of the transparent mode escape sequence after entering the host copy command
Read-after-write verify error.	1. Target hardware problem. 2. Overlay memory not mapped in download range. Address is indicated by misverify message.

(continued)

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 5-57.)

This command is also useful for determining differences between object files. Follow instructions for downloading a file on page 5-38.

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, page 5-29.)

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 5-29.

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 (see **SET** parameter #26, page 5-7).

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 page 5-29.)

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 Serial Communications, page 5-29 and Symbols, page 5-111.

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.

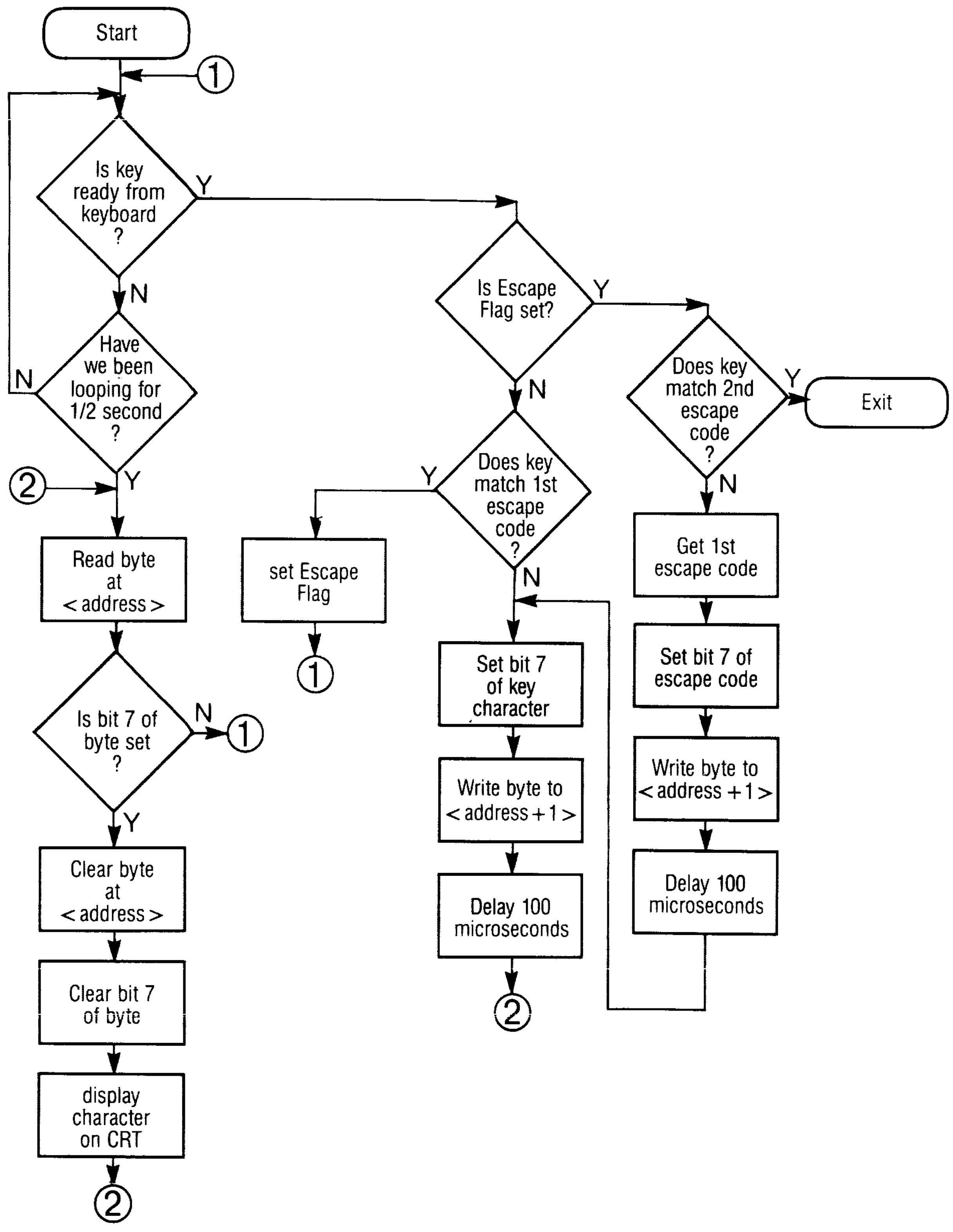
The **COM** routine does not check the byte at *<address>* + 1 to see if the target program has received it. Generally, the target program will be substantially faster than the **COM** routine and will always receive one character before the **COM** routine can transmit the next.

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


Start
1
Is key ready from keyboard ?
Y
N
Have we been looping for 1/2 second ?
N
Y
2
Read byte at < address >
Is bit 7 of byte set ?
N
1
Y
Clear byte at < address >
Clear bit 7 of byte
display character on CRT
2
Is Escape Flag set?
Y
N
Does key match 1st escape code ?
Y
set Escape Flag
1
N
Set bit 7 of key character
Write byte to < address + 1 >
Delay 100 microseconds
2
Does key match 2nd escape code ?
Y
Exit
N
Get 1st escape code
Set bit 7 of escape code
Write byte to < address + 1 >
Delay 100 microseconds

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 $\boxed{\$00}$ from target memory. Affects real time operation when entered in run mode. See page 6-1.

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.

DISPLAY MEMORY MAP

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.

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* **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 (see Overlay Memory Enable, page 5-61.

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.

You can save and restore the contents of the overlay map by using the EEPROM **LD/SAV** commands (see pages 5-27 and 5-25). You cannot save the contents of overlay memory in EEPROM.

(continued)

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 OFFFFO (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 0FFFF0:RO              Need Reset Vector mapped as ROM
 >GR0 = 1000 LEN 8000            Will save some typing
 >MAP GR0                        Map 32K bytes for code space
 >DNL                            Download first program into overlay
  (Download commands and records)
 >MAP GR0:TGT                    Unmap code space (The data is still
                                 in overlay memory)
 >MAP GR0 + 10000                Remap but at higher address range.
                                 The first program now "exists" again
                                 but in a higher address range.
 >MAP GR0                        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 GR0:TGT;MAP GR0 + 20000    Relocates second program out of the way
 >MAP GR0 +10000:TGT;MAP GR0     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.

OVERLAY MEMORY ENABLE

Command	Result
OVE = CD + DTA	The overlay memory decodes both code and data space.
OVE = CD	Only code status space accesses are decoded by overlay memory.
OVE = DTA	Only data status space accesses (including **ALT**, **DAT** and **STA** space) are decoded by overlay memory.

Comments

Overlay memory responds to an access only if a mapped address and the current **OVE** status match the cycle being executed. For more information about the four status spaces, see segment description in the raw trace section (page 5-96), and the *iAPX 86/88, 186/188 Users Manual.*

CD is code space. The processor encodes it as code status.

DTA is data space. The processor encodes it as data, alternate data or stack status.

Overlay memory cannot be divided between **CD** and **DTA** on the same map. It is either all one (**CD**), or the other (**DTA**), or all both (**CD+DTA**).

To display the value of the current status being used for memory access, use the **MMS** command (page 5-82).

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.

Use the **VFO** command (page 5-63) to verify the memory move.

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.

(continued)

Registers

This section includes information on using the registers and a complete list of all the registers in the Emulator.

The registers can be logically divided into four groups:

- microprocessor registers
- general Emulator registers
- target Peripheral Control Block (PCB) registers, those used only in iRMX mode and those used in non-iRMX mode
- Event Monitor System registers

Each Emulator or Event Monitor System 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.

Each register has a separate display base. The display base is viewed and changed with the **BAS** command (see page 5-80). 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.

PERIPHERAL CONTROL BLOCK (PCB) REGISTERS

Because of the dynamic nature of some PCB registers, they are handled slightly differently than regular CPU registers. The following sections describe the problems and their solutions.

GENERAL PCB HANDLING

When the Emulator *exits* run mode, all memory and I/O space is searched for the PCB. When the PCB is located, it is moved to locations $FF00-$FFFF in I/O space. All register values are then copied to a table in internal RAM and uploaded to the ES controller. These register values are the ones displayed in response to the **PCB** command. The values in this table are modified by commands such as:

```
MCWO=$1234
```

or

```
IST=$5678
```

RELOCATION OF THE PCB

The PCB is completely relocatable anyplace in memory or I/O. It contains an interrupt controller, two timers, three counters, two DMA channels and chip select circuitry for decoding memory and I/O space. There are many details to understand and remember when dealing with the PCB. These details are pointed out in the following subsections.

Since the PCB is relocatable, there are several things that need to be understood concerning the registers in the PCB. On the run-to-pause transition the firmware takes a copy of the CPU registers and the registers in the PCB and stores them first in a RAM table on the Emulator board and then passes a copy of the registers to ESL. The copy that is sent to ESL is what is shown to the user. When you make a change to any of the registers, that change is simply stored in the RAM table kept by ESL. If you then

(continued)

ask to look at those registers you see the change made, but the change is only to the RAM table and not to the CPU.

Prior to the transition from pause to run the registers are passed to the firmware from ESL. The registers are then loaded into the CPU, and control is turned over to the target. So if you want to load a register into the CPU, you first need to equate the register to the correct value and then put the ES1800 into either run mode or single step.

On the run-to-pause transition, the firmware locates the PCB and moves it back to the power-up location of 0FF20 in I/O space. This is done because some users actually move the PCB to some other location. The firmware moves the PCB to its default location so that it will not write over the top of the PCB while in pause mode.

If you use the **MIO** command to write to the PCB and change the contents of the registers, two questions may arise:

The first is if you try to write to their PCB at the location you moved it to and can't find it. The second question may occur if you write to the PCB by using the **MIO** command and then look at the PCB registers through the ESL command (**PCB**) and find that the register you changed in the PCB was not changed in the ESL RAM table. If after you make the change to the PCB via the **MIO** command you then execute the **RUN** command and then wonder why the CPU didn't use the value loaded into the PCB.

The answer to the first question is simply that the PCB is moved to the default location, so you will not find the PCB in the spot you moved it to. The PCB is always moved back to the correct location on the pause-to-run transition.

The answer to the second question is that the values in the ESL RAM table are only loaded there from the PCB on the run-to-pause transition. Also the values loaded back into the PCB on the pause-to-run transition are from the ESL RAM table and therefore write over the top of anything that the user puts into the PCB. To avoid this problem, use the ESL command format provided for changing the values of the registers (see page 5-79).

The commands *do not* modify the current contents of the physical PCB until the next pause-to-run transition.

When the Emulator *enters* run mode, the PCB register values contained in the RAM table mentioned above are reloaded into the physical PCB. The PCB is then moved back to its location in the target address space and the Emulator enters the target system.

EXCEPTIONS

The Emulator may be configured to allow some or all of the integrated peripherals controlled by the PCB to continue operating during pause mode.

TIMERS

The ON/OFF switches TE0, TE1, and TE2 are used to selectively enable/disable the integrated timers during pause mode (see page 5-23).

If the switch is set to ON, the timer registers are handled as described in the general procedure upon the run-to-pause transition. On the pause- to-run transition, none of the timers' values are reloaded to the physical PCB, as this would destroy the data generated during pause mode.

If the switch is set to OFF (disable time during pause mode), the mode control (MCWO) for the particular timer is copied to the RAM table upon run to pause; the timer is then disabled by clearing bit 15 of the mode control word. Upon pause-to-run, the value in the RAM table is reloaded to the physical PCB. This restores the timer to its configuration when last running in the target system.

DMA CONTROLLERS

The ON/OFF DME switch selectively enables/disables DMA operation during pause mode. Note that all DMA cycles are disabled immediately upon run-to- pause transition by the asserting of an NMI to the CPU, which then sets bit 15 of the IST register (DHLT bit).

If the switch is set to ON DME, the IST register is copied to the RAM table. The DHLT bit is then cleared, causing DMA cycles to resume. All DMA cycles are steered to the target system.

(continued)

Upon pause-to-run transition, the RAM table value of the IST register is reloaded to the physical PCB. If you want DMA activity to continue when reentering run mode, be sure the CDH soft switch is turned on.

No DMA register values are reloaded to the physical PCB with this setting.

If the switch is set to OFF DME, the DMA registers are handled as described in *General PCB Handling.*

CHIP SELECT REGISTERS

The ON/OFF RCS switch controls the Emulator's reading of the LMCS, MMCS, MPCS, and PACS registers upon run-to-pause transition.

If the switch is set to ON RCS, all chip select registers are read and restored as described in "General PCB Handling."

If the switch is set to OFF RCS, these chip select registers are read and copied to the RAM table only if you have manually set the register value during pause mode (i.e., LMCS=1234). This is necessary because reading of these chip select registers enables them to drive the 80186/88's chip select lines.

Upon pause-to-run transition, only the registers that have been modified during pause mode (i.e., LMCS=1234) are reloaded to the physical PCB. Note that when the switch is OFF, the displayed values of the chip select registers (LMCS, MMCS, MPCS, PACS) do not show what is actually in the PCB.

When attempting to peek and poke into target space it is necessary to set up the CS registers first so the address is decoded and the correct CS line toggled. The CS registers can be set up either by running the code in the target system or by setting up each of the registers in ESL and then executing an **STP** to load them into the CPU.

When making the transition from run to pause, whether it is during the run mode or during a step, the CPU picks up its NMI vector from the internal world, but it uses the target ready to complete the bus cycle. The NMI vectors are located at address 8, 9, A and B, which fall into the area of LMCS. So if the LMCS is not set up and a break occurs or a step, then the ES hangs up waiting for a ready from the target.

When reading the contents of the CS registers the value returned is often different from the value written into the register. This is a function of the CS registers having some bits that are read only. LMCS register bits 3, 4 and 5 are always high. MMCS register bits 3 through 8 are always high. PACS register bits 3 through 5 are always high. UMCS register bits 3 through 5, 14 and 15 are always high.

INTERRUPT CONTROLLER REGISTERS

Upon pause-to-run, the poll status register (POS) and its value are copied to their own table entry as well as the entry for the poll register (POL). This is necessary to prevent setting the IS bit of any pending interrupt. Both registers also show the same data in the PCB.

Because these two registers are Read Only, they are not reloaded to the physical PCB upon pause-to-run transition.

On the run-to-pause transition all interrupts are disabled because there is no way for the ES to handle interrupts during pause. This means that both external and internal interrupt sources are ignored and not processed. Interrupts are restored to their previous condition upon the pause-to-run transition. If any of the interrupts that occurred during pause are still pending upon the pause-to-run transition, they are serviced at that time.

(continued)

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 (low and high)	Int	16,8,8
CS	code segment	Int	16
CX, CL, CH	count (low and high)	Int	16,8,8
DI	destination index	Int	16
DS	data segment	Int	16
DX, DL, DH	data (low and high)	Int	16,8,8
ES	extra segment	Int	16
FLX, FLL, FLH	flags (low and high)	Int	16,8,8
IP	instruction pointer	Int	16
SI	source index	Int	16
SP	stack pointer	Int	16
SS	stack segment	Int	16

General Emulator Registers

Name	Description	Type	Length (bits)
DFB	default base	Int	8
GD0-GD7	general purpose data	DC	32
GR0-GR7	general purpose range	Range	32
IDX	repeat index register	Int	32
IOP	I/O mode pointer	Int	16
LIM	repeat limit register	Int	32
MMP	memory mode pointer	Int	32
MMS	memory mode status	DC	16
OVE	overlay enable	DC	8
TST	terminator for repeats	Int	32

Target Peripheral Control Block (PCB) Registers

Name	Description
REL	relocation register
UMCS	upper memory chip select control
LMCS	lower memory chip select control
MMCS	mid-range memory chip select control (base address)
MPCS	mid-range memory chip select control (block size)
PACS	peripheral chip select control
TC0	timer #0 count register
TC1	timer #1 count register
TC2	timer #2 count register
MA0	timer #0 max count A register
MA1	timer #1 max count A register
MA2	timer #2 max count A register
MB0	timer #0 max count B register
MB1	timer #1 max count B register
MCW0	timer #0 mode control word register
MCW1	timer #1 mode control word register
MCW2	timer #2 mode control word register
USRC0	dma #0 upper 4 bits of source address
USRC1	dma #1 upper 4 bits of source address
SCR0	dma #0 lower 16 bits of source address
SCR1	dma #1 lower 16 bits of source address
UDST0	dma #0 upper 4 bits of destination address
UDST1	dma #1 upper 4 bits of destination address
DST0	dma #0 lower 16 bits of destination address
DST1	dma #1 lower 16 bits of destination address
XC0	dma #0 transfer count
XC1	dma #1 transfer count

(continued)

Target Peripheral Control Block (PCB) Registers *(cont.)*

Name	Description
CW0	dma #0 control word
CW1	dma #1 control word

PCB Registers Used Only in iRMX Mode

Name	Description
EOI	specific end of interrupt register
MSK	mask register
PLM	priority level mask register
ISV	in service register
IRQ	interrupt request register
IST	interrupt status register
IV	interrupt vector register
DMA0	level #2 interrupt control register (dma #0)
DMA1	level #3 interrupt control register (dma #1)
TMR0	level #0 interrupt control register (timer #0)
TMR1	level #4 interrupt control register (timer #0)
TMR2	level #5 interrupt control register (timer #0)

PCB Registers Used in Non-iRMX Mode

Name	Description
POL	poll register
POS	poll status register
MSK	mask register
PLM	priority level mask register
ISV	in service register
IRQ	interrupt request register
IST	interrupt status register
IV	interrupt vector register
TCR	timer interrupt control register
DMA0	dma #0 interrupt control register
DMA1	dma #1 interrupt control register
INT0	interrupt control register #0
INT1	interrupt control register #1
INT2	interrupt control register #2
INT3	interrupt control register #3

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	Int	32

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
0000:0000  .........  0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
```

Command	Result
<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 6-5).

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 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 $\boxed{.}$ 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
```

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
PCB	Displays contents of the peripheral control block registers.

Examples

```
 >PCB
** RELOCATION REGISTER      REL = 20FF

** CHIP SELECT CONTROL      UMCS   LMCS   MMCS   MPCS   PACS
                            FFFB   0000   0000   0000   0000

** TIMER REGISTERS
                                   TC     MA     MB     MCW
                       TIMER 0     0000   0000   0000   0000
                       TIMER 1     0000   0000   0000   0000
                       TIMER 2     0000   0000   ----   0000

**DMA REGISTERS
                    USRC   SRC    UDST   DST    XC     CW
        CHANNEL 0   0000   0000   0000   0000   0000   0000
        CHANNEL 1   0000   0000   0000   0000   0000   0000

**INTERRUPT CONTROL REGISTERS

EOI    POL    POS    MSK    PLM    ISV    IRQ    IST
0000   0000   0000   0000   0000   0000   0000   0000

TCR    DMA0   DMA1   INT0   INT1   INT2   INT3
0000   0000   0000   0000   0000   0000   0000
 >
```

LOADING REGISTERS

Command	Result
<register> = *<value>*	Sets *<register>* to *<value>*.
<register>	Displays register.

Examples

```
>UMCS = $FFFB
```

```
>REL = $20FF
```

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, or binary, the Emulator displays a question mark ($\boxed{?}$) preceding the base value when asked to display the base in effect.

Refer to the default base command, **DFB** (page 5-86), 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 STATUS REGISTER

Command	Result
MMS	Displays the value of the current status being used for memory accesses. Refer to the *iAPX 86/88, 186/188 Users Manual* for details.
MMS = *<MMS register>*	Sets the status space for memory accesses.

Comments

The four registers that can be used for data accesses are:

```
    ALT - alternate
    DAT - data
    COD - code
    STA - stack
```

Examples

Memory commands use the alternate data register to access memory.

```
  >MMS = ALT
```

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 6-15.

(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 6-36.

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 the target hardware, but have downloaded a symbol table containing them, then use the symbol names instead of looking up the hardware specifications.

```
>GR2 = 'RAM LEN 'RAM_len        Initialize GR2
>SF 0,GR2                       Run a RAM test on your RAM
>AC1 = GR2                      Set a breakpoint on any RAM access
>WHE AC1 THE BRK
```

TEST REGISTER

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

Comments

See Repeat Operators (page 5-107) 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

```
>DRT #50
LINE  ADDRESS   DATA  R/W       M/IO    BCYC SEQ QUE  LSA  -  8  7  -  0
 #69  001000  > 0FB9  R   OVL    M      IF    C  F 0  %11111111 %11111111
 #68  001002  > BE00  R   OVL    M      IF    C    2  %11111111 %11111111
 #67  001004  > 2000  R   OVL    M      IF    C    2  %11111111 %11111111
 #66  001006  > 00BF  R   OVL    M      IF    C    1  %11111111 %11111111
 #65  001008  > A522  R   OVL    M      IF    C    2  %11111111 %11111111
 #64  00100A  > A4F3  R   OVL    M      IF    C    2  %11111111 %11111111
 #63  00100C  > 8103  R   OVL    M      IF    C    3  %11111111 %11111111
 #62  002000  > FF50  R   OVL    M      RM    D    4  %11111111 %11111111
 #61  002200 <  FF50  W   OVL    M      WM    D    4  %11111111 %11111111
 #60  00100E  > FF00  R   OVL    M      IF    C    3  %11111111 %11111111
 #59  001010  > 02B9  R   OVL    M      IF    C    1  %11111111 %11111111
 #58  002002  >   3E  R   OVL    M      RM    D    1  %11111111 %11111111
 #57  002202 <    3E  W   OVL    M      WM    D    1  %11111111 %11111111
 #56  002003  > FF    R   OVL    M      RM    D    1  %11111111 %11111111
 #55  002203 <  FF    W   OVL    M      WM    D    1  %11111111 %11111111
 #54  002004  >   00  R   OVL    M      RM    D    1  %11111111 %11111111
 #53  002204 <    00  W   OVL    M      WM    D    1  %11111111 %11111111
 #52  002005  > 00    R   OVL    M      RM    D    1  %11111111 %11111111
 #51  002205 <  00    W   OVL    M      WM    D    1  %11111111 %11111111
 #50  002006  >   FF  R   OVL    M      RM    D    1  %11111111 %11111111
```

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.
ADDRESS DATA	The address displayed is where the bus cycle took place, along with the data written to, or read from, that address.

(continued)

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

TAR/OVL	TAR/OVL indicates whether the access was in the target memory area or in the Emulator's overlay (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:

IAK	interrupt acknowledge
RIO	read from I/O
WIO	write to I/O
HLT	halt
IF	instruction fetch
RM	read memory
WM	write memory
NBC	no bus cycles
X87	8087 microprocessor instruction

SEG	SEG indicates what type of segment is being used by the program for data accesses. The possibilities are:

A -	Alternate Data
C -	Code
D -	Data
S -	Stack

Refer to *iAPX 86/88, 186/188 Users Manual* for definition of these segment types.

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 $\boxed{\text{Q}}$ 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
```

```
>DT 0
SEQ# ADDR OPCODE  MNEMONIC   OPERAND FIELDS     BUS CYCLE DATA

0028 000A 8B4600  MOV     AX,WORD PTR [BP+0]      0800>10C5
0027 000D 050100  ADD     AX,1
0024 0010 EBF4    JMP     SHORT 0006
0020 0006 904600  MOV     WORD PTR [BP+0],AX      0800<10C6
0019 0009 90      NOP
0018 000A 8B4600  MOV     AX,WORD PTR [BP+0],AX   0800<10C6
0017 000D 050100  ADD     AX,1
0014 0010 EBF4    JMP     SHORT 0006
0010 0006 904600  NOP
0009 0009 90      NOP
0008 000A 8B4600  MOV     AX,WORD PTR [BP+0]      0800>10C7
0007 000D 050100  ADD     AX,1
>
```

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.

(continued)

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.

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 5-98, the **DRT** command, page 5-94, and the slash command, page 5-110.

(continued)

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 5-27 and 5-25) 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
 >
```

DEFINE/EXECUTE MACROS

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
```

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=
```

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 ($\boxed{*}$) 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** variable 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
```

(continued)

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** variable 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
```

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
>/
```

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 ($\boxed{:}$). 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.

(continued)

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.

```
 >GR0=1000 LEN 2A
 >DIS GR0
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
```

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
```

(continued)

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
```

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
```

SYMBOL DEFINITION

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.

Comments

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 you end the symbol name by typing a space character, the 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.

Examples

`'testing` is recognized as the symbol.

```
>'testing =GR0
```

`'testing=GR0` is recognized as the symbol name. The name will probably not be found and you will get an error message.

```
>'testing=GR0
```

```
>'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
 >
```

(continued)

Miscellaneous Commands

DISPLAY THE SOFTWARE REVISION DATES

Command	Result
REV	Displays the software revision dates for ESL and the firmware.

Comments

This command is valid only in pause mode.

When you call AMC customer service, they 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 2.2
WED AUG 6 16:50:26 PDT 1986 - FW 3.12
 >
```

DISPLAY A BLANK LINE

Command	Result
RET	Outputs a $\boxed{\text{RETURN}}$, linefeed.

Comments

This command improves readability when displaying a lot 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

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.

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  = FFFFH
IP  = 0
FLX = F002H
```

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.

3. Issuing the reset character stops emulation. (See page 4-25 and 5-4 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 5-54 for overlay memory access rights). The condition that caused the error is displayed.

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** stops emulation and returns 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

$\boxed{R>}$ indicates that the Emulator is in run mode. $\boxed{>}$ indicates that the Emulator is in pause mode.

See the Switch section under **STI**, page 5-22, for more information about stepping.

Do not attempt to STP through an NMI vector fetch. This causes the emulator to hang. It is possible to STP through the NMI interrupt routine, but not the NMI vector fetch. All other vector fetches can be STP'ed through.

(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 = FFFFH
        IP = OH
        FLX = F002H
```

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 5-64).

(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
FFFF:0000  .......... 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 4-25 and 5-4 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 = FFFFH
CS = 0
FLX = F002H
```

Comments

The **RST** command can be issued from either run or pause mode. When in pause mode, the **RST** command resets the mecroprocessor and loads the reset vectors (**LDV**). While in run mode the microprocessor is reset in the target environment and emulation continues. This causes the microprocessor to start fetching instructions from the reset vector. **RST** does not affect the barget 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.

Examples

In the example below, the Emulator is in run mode. The microprocessor is reset in the target environment and emulation continues.

(continued)

```
R> RST
R>
```

In the next example, the Emulator is in pause mode. The microrprocessor 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 5-61) 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 Commands that accept data parameters truncate the data entered to the current default data length. If you enter `FIN 0 LEN 20,23F6` 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. If memory commands are executed while in run mode, remember that you are not emulating in real-time.

(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

DISPLAY MEMORY BLOCK

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 ($\boxed{/}$) (page 5-110). 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 5-3).

DB affects real-time operation when entered in run mode (see page 6-1).

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 pages 4-9 and 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

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** parameter #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 6-15.)

The **FIL** command can be verified using the **VBL** (Verify BLock) command (page 6-22).

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 6-15.)

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*>. The current value of MMS specifies the relocation register used during the transfer.
BMO<*range*>,<*space*>,<*address*>	Moves <*range*> to the new <*address*>. The <*space*> argument specifies the memory mode status to use during the transfer.
BMO<*range*>,<*address*>,<*space*>	Moves <*range*> to the new <*address*>. The range is read from the space specified in the MMS register. The block is written to <*space*>.
BMO<*range*>,<*space*>,<*address*>,<*space*>	Moves <*range*> to the new <*address*>. The range is read from <*space*> specified in the argument following the range. The block is written to <*space*> specified in the argument following the address.

(continued)

Comments

This command is valid in pause mode only.

The following rules of thumb may make the numerous forms of this command less confusing.

- If there is no space specified for the source argument, **MMS** is always used (page 5-82).
- If no space is specified for the destination address, the source space is always used.
- A non-overlapping block move can be verified using the **VBL** command.

Examples

Move a range to a new location in data space.

```
>MMS=DAT
>BMO 100 TO 500, 1000
```

or

```
>BMO 100 to 500, DAT, 1000
```

Move 20 bytes from the stack in stack space to the value pointed to by the data register in data space.

```
>BMO SS:SP LEN 20, STA, DX, DAT
```

Command	Result
VBM*<range>*,*<address>*	Verifies move of *<range>* to the new *<address>*. The current value of MMS specifies the relocation register used during the transfer.
VBM*<range>*,*<space>*,*<address>*	Verifies move of *<range>* to the new *<address>*. The *<space>* argument specifies the memory mode status used during the transfer.
VBM*<range>*,*<address>*,*<space>*	Verifies move of *<range>* to the new *<address>*. The range is read from the space specified in the MMS register. The block is written to the *<space>* specified in the argument following the address.
VBM*<range>*,*<space>*,*<address>*,*<space>*	Verifes move of *<range>* to the new *<address>*. The range is read from *<space>* specified in the argument following the range. The block was written to

(continued)

the *<space>* specified in the argument following the 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 5-63, 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 5-61.

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 5-61.

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
**** 8086/88/186/188 LINE ASSEMBLER V2.6LA ****
CSEG = XXXX
0000 >X
 >
```

Command	Result
ASM <arg>	Assembly begins at the specified address.

```
 >ASM <address>
**** 8086/88/186/188 LINE ASSEMBLER V2.6LA ****
CSEG = XXXX
0000 >END
 >
```

Command	Result
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 80186/88 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:
ORG	Sets 64K byte offset into the code segment window:
END or X	Exits line assembler to the command level:

CSEG:

```
1012 >CSEG D400H
1012 >
```

ORG:

```
1012 >ORG 3ACH
03AC >
```

END or X:

```
58FD >X
**** END OF LINE ASSEMBLY ****
>
```

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 OFDEOH
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 FDEOH
756A >
```

RETURN	Disassembles one instruction at the current address:
$	Current assembler offset address.
NEAR	Within current line assembly segment.
FAR	Outside current line assembly segment.

```
5D0A >
5D0A 3306AD78       XOR AX,WORD
     PTR 781DH
5DE >
```

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 8086 assembly language programming and have the *iAPX 86/88, 186/188 User's Manual* by Intel.

Page length is defined by the CRT length parameter in the **SET** menu (page 5-3).

A disassembly command with an integer argument or no argument enters a special disassembly mode. The disassembly can be continued by typing a <space> or $\boxed{\text{RETURN}}$. Exit disassembly by typing any other character.

<space>	Continues disassembling one line at a time.
$\boxed{\text{RETURN}}$	Continues disassembling one page at a time.
any char except <space> or $\boxed{\text{RETURN}}$	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 $\boxed{\text{RETURN}}$, the next address and its data are displayed. If a value is entered before the $\boxed{\text{RETURN}}$, 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 and enters memory mode. 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 $\boxed{.}$ and comma $\boxed{,}$ 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 pages 5-83 and 6-48.

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 $\boxed{\text{RETURN}}$ 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 $\boxed{.}$ increments the address and a $\boxed{,}$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 pages 5-85 and 6-47.

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 page 6-15).

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 4-24.)

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 6-30).

```
 >M 1000
 $001000 $00  >0,1,2,3,4,5,6,7,8
 $001009 $00  >X
 >ASM MMP
****  8086/88/186/188 LINE ASSEMBLER  V2.6LA ****
CSEG = 0000

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 page 6-15).

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 6-47.)

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.

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 "next" and "last" 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 and memory mode is entered at the current value of MMP.

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 6-38).

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 6-38.)

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 6-52).

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 4-25 and 5-4). The scope loops can be executed in either byte or word mode.

Memory scope loops access the memory space defined by the current MMS (Memory Mode Status) register.

I/O scope loops access the target system's I/O space.

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 10 {30},<RANGE><CR>                 FORCE NOP TO INCREMENTING ADDRESS
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:

	Pattern	
Sequence	BYM	WDM
1	0000	0000 0000
2	0001	0000 0001
3	0011	0000 0011
4	0111	0000 0111
5	1111	0000 1111
6	1110	0001 1111
7	1100	0011 1111
8	1000	0111 1111
9		1111 1111
10		1111 1110
11		1111 1100
12		1111 1000
13		1111 0000
14		1110 0000
15		1100 0000
16		1000 0000

(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 Semi-conductor 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 4-25 and 5-4).

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 4-25 and 5-4).

TOGGLE DATA AT ADDRESS

Command	Result
SF 4 *<address>*,*<data>*	*<data>* is written to the specified address in the memory space defined by MMS.
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 4-25 and 5-4).

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 using MMS as status space register.
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 4-25 and 5-4).

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 using MMS as status space register.
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 4-25 and 5-4).

Examples

```
>SF 6, 10,$FFFF
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```

The data pattern written to address 10 is:

(continued)

```
          (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 using MMS as status space register 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 4-25 and 5-4).

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 using MMS as status space register, 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 4-25 and 5-4).

Examples

```
  >SF 8,1000,05
 YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```

(continued)

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 using MMS as status space register, 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 4-25 and 5-4).

Examples

```
>SF 9, 100,$FFFF
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```

This page intentionally left blank.

WRITE INCREMENTING VALUE

Command	Result
SF 11, *<address>*	Consecutively writes a constantly incrementing value to the specified memory address using MMS as status space register.
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 4-25 and 5-4).

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 using MMS as status space register.
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 page 4-25 and 5-4.)

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 ($\boxed{>}$) 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 *<range>*	The system calculates a cyclic redundancy check on all addresses in *<range>*.
CRE *<address range>*	Calculates a cyclic redundancy check on even addresses.
CRO *<address range>*	Calculates a cyclic redundancy check on odd addresses.

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** operators 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 SEVERAL STATUS LINES

Command	Result
BUS	The bus status is displayed:

Comments

The status of the following three bus lines is displayed:

NMI	Non-maskable interrupt
ARDY	Asynchronous ready
SRDY	Synchronous ready
INT0	Interrupt 0
INT1	Interrupt 1
INT2/INTA0	Interrupt 2 or interrupt acknowledge 0
INT3/INTA1	Interrupt 3 or interrupt acknowledge 1
TEST	Test input

Examples

```
  >BUS
 NMI   ARDY   SRDY   INT0   INT1   INT2/INTA0   INT3/INTA1   TEST
  0     1      0      0      0         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 three 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 Intel 80186/80188 microprocessors multiplex address and data lines. The ES1800 demultiplexes those signals so that the Event Monitor System "sees" all signals at the same time. The EMS essentially takes a picture of the microprocessor's signals at the beginning of every T4 state (refer to Intel manual, *iAPX 86/88, 186/188 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
```

ODD BOUNDARIES

The address comparators in the 80186 may need to be specially set up. The 80186 is a 16-bit chip with a prefetch QUE and byte based instructions. This causes problems with breaking on instructions that occur on odd boundaries.

This section describes three distinct conditions, and suggestions for resolving them.

1. 80186 prefetches an instruction.

   When the 80186 prefetches an instruction, it outputs the even address. Both bytes are fetched, and the actual (odd) address of the byte in question is never seen. This means that you can't set the Event Monitor System to break on the odd address.

2. 80186 jumps to an odd address.

   When the 80186 jumps to an odd address, the odd address does appear on the bus, and only that byte is fetched. In this case, the Event Monitor System works as expected.

3. Only the low byte is read.

   If only the low byte is read, the even address appears on the bus, and the odd byte is not read. This means you can't set the Event Monitor System to break on the odd address.

The ES 1800 Event Monitor System can be set up to resolve conditions 1 and 3, and to guarantee correct operation in condition 2.

Assume the byte in question is at $04001. This byte could be accessed by the address $04001 or $04000. If the address $04001 is on the bus, then the byte is accessed. If the address $04000 is on the bus, and the bus cycle is a 16-bit cycle, then the byte is accessed. If the address $0400 is on the bus, and the bus cycle is an 8-bit cycle, then the byte is not accessed.

This Event Monitor System setup handles this condition:

```
AC1=$04000
AC2=$04001
S1=WRD
WHEN AC1 AND S1 OR AC2 THEN BRK
```

AC1 contains the even address. S1 is the word bus cycle condition. If both are true, the high or odd byte has been accessed. AC2 contains the actual odd address. If it is true, then the byte is always being accessed. If neither is true, then the byte is not being accessed.

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 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 $\boxed{X}$ 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 = GD0 + $F
>DC1.4 = @'data table + 56
```

STATUS COMPARATORS

The status comparators are assigned values from the list of status constants. Many of these constants can be combined to specify a complex comparator value. The list on the next page shows the available mnemonics. Any of these statuses can cause events.

STATUS MNEMONICS

ALT	Alternate Data Access	QD1-6	Queue Depth (1-6)
BYT	Byte Access	QF	Queue Flush Cycle
COD	Code Access	RD	Read
DAT	Data access	RIO	Read IO Status
HLT	Halt Status	RM	Read Memory Status
IAK	Interrupt Acknowledge Status	STA	Stack Access
IF	Instruction Fetch Status	TAR	Target Access
IOA	IO Access	WIO	Write IO Status
MEM	Memory Access	WM	Write Memory Status
NBC	No Bus Cycle Status	WR	Write
NMI	NMI Cycle	WRD	Word Access
OVL	Overlay Access	DMA	DMA Cycle

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 2-8 and any or all from column 9. Mnemonics are combined using an addition operator ($\boxed{+}$) as a Boolean AND.

STATUS MNEMONIC TABLE

```
S1 = TAR  +  RD  +  BYT  +  MEM  +  ALT  +  HLT  +  QD1  +  QF
S2   OVL     WR     WRD     IOA     COD     IAK     QD2     NMI
                                    DAT     NBC     QD3     DMA
                                    STA     RIO     QD4
                                            RM      QD5
                                            WIO     QD6
                                            WM      QD6
                                            IF
```

Some examples of status comparator assignments:

```
>S1=BYT
>S2=OVL+RD+DAT
>S1.3=WR+IOA
>S2.4=RIO
>S1.2=QF
```

BREAKING ON NMI

Although it may be tempting to use the NMI status to break on NMI, do not use this status with the break action. Setting a breakpoint on an NMI fetch will cause the emulator to hang, requiring a Ctrl-Z to recover. To break on an NMI, set the event system to break on the starting address of the NMI interrupt routine. The NMI status may be used as a qualifier for other EMS actions.

Figure 7. Status Translation Table

STATUS TRANSLATION TABLE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	NMI			X87							QF	MEM/ IOA	TAR/ OVL	RD/ WR	BYT/ WRD
		SEGMENT				CPU STATUS			QUE DEPTH			EMULATOR STATUS			
	NMI=0	ALT=0 STA=1 COD=2 DAT=3		X87=1		IAK=0 RIO=1 WIO=2 HLT=3 IF=4 RM=5 WM=6 NBC=7			QD1=1 QD2=2 QD3=3 QD4=4 QD5=5 QD6=6		QF=0	MEM=1 IOA=0	TAR=1 OVL=0	RD=1 WR=0	BYT=1 WRO=0

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
$00000504 DC 0000B8F8
```

All bits except bits 2, 8, 9, 10 and 14 are masked. Bit 14 is enabled and a 0 is in the bit 14 of the status value, so **NMI** was entered.

Bits 8,9, and 10 are enabled and there is a 101 (5) in those bits in the status value so **RM** was entered.

Bit 2 is enabled and there is a 1 in bit 2 of the status value so **TAR** was entered.

Therefore, the original input was:

```
>S1=NMI+RM+TAR
```

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 Defining Action Lists, page 7-9). 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 will 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.

Example

<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.

The EMS resolves conflicting EMS control statements. The **TOC** action in the first statement:

```
>WHEN AC1 THEN TOC
>WHEN AC1 THEN CNT
```

is changed to **CNT**.

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.

(continued)

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.

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.

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	Trace Nothing Until First TOT
TRC	Trace All Cycles	Trace Only Qualified Cycles	Trace Nothing Until First TOT
TOT (Not Tracing)	Trace All Cycles	Trace Only Qualified Cycles	Trace Nothing Until first TOT
TOT (Tracing)	Trace All Cycles	Trace Only Qualified Cycles	Trace All 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

(continued)

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>
```

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.

(continued)

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	Count All Until First TOC
CNT	No Cycles Counted	Count Only Qualified Cycles	Count Nothing Until First TOC
TOC (Not Counting)	No Cycles Counted	Count Only Qualified Cycles	Count Nothing Until first TOC
TOC (Counting)	No Cycles Counted	Count Only Qualified Cycles	Count All Until first TOC

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>
```

(continued)

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>
```

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. This is asserted as a TTL-level high signal. 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.

(continued)

```
  >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

ES LANGUAGE MNEMONICS

List of Commands

Command	Description	Page
>	Pause mode prompt	*4-23*
R>	Run mode prompt	*4-23*
RETURN	Return key	*4-6*
/	Repeat previous command line	*4-7,*
;	Statement separator	*4-6*
*	Repeat command	*4-6*
CTRL Q	Start screen scrolling (can be changed)	*4-25*
CTRL R	Reprint current line	*4-25*
CTRL S	Stop screen scrolling (can be changed)	*4-25*
CTRL X	Delete line	*4-25*
CTRL Z	Reset the emulator (can be changed)	*4-25*
ESC ESC	Escape transparent mode (can be changed)	*4-25*
$	Hexadecimal	*4-12*
#	Decimal	*4-12*

Command	Description	Page
%	Binary	*4-12*
\	Octal	*4-12*
=	Equals	*4-12*
()	Parentheses	*4-13*
@	Indirection	*4-9*
*	Multiplication	*4-17*
/	Division	*4-17*
+	Addition	*4-17*
-	Subtraction	*4-17*
-	Negation	*4-15*
&	Bitwise AND	*4-17*
^	Bitwise OR	*4-17*
<<	Shift left	*4-17*
>>	Shift right	*4-17*
!	Inverse, bitwise NOT	*4-15*
:	Intel Segment/offset operator	*4-9*
:	Memory block attribute	*6-15*
.	Increment Memory Mode address	*6-38*
.	Execute macro #2	*5-104*
,	Decrement Memory Mode address	*6-38*
,	Execute macro #1	*5-104*
?	Help menu	*4-20*
?	Error query	*4-7*
_	Define/execute macro	*5-104*
'	Symbol definition (single quote)	*4-10*

Command	Description	Page
ABS	Absolute value	*4-16*
AC1, AC2	Address comparators 1 and 2	*7-3*
ALT	Alternate data access	*7-5*
AND	Logical event AND	*4-15*
ASM	Line assembler	*6-30*
AX, AL, AH	Accumulator (low and high)	*5-70*
BAS	Set/display base value	*5-80*
BKX	Break on instruction execution	*5-13*
BMO	Block move	*6-23*
BP	Base pointer	*5-70*
BRK	Break	*7-14*
BUS	Display status of lines	*6-76*
BX, BL, BH	Base register (low and high)	*5-70*
BYM	Byte mode	*4-24*
BYT	Byte access status	*7-5*
CCT	Computer port control	*5-36*
CD	Overlay enable for code access	*5-61*
CDH	Clear DMA halt switch	*5-14*
CES	Clear WHEN/THEN statements	*7-13*
CK	Internal/external clock selection	*5-14*
CLK	Read target system clock	*6-74*
CLM	Clear memory map	*5-60*
CLR	Clear microprocessor data registers	*5-74*
CMC	Clear macros	*5-106*
CNT	Count event	*7-19*
COD	Code status	*7-5*

Command	Description	Page
COM	ASCII communication	*5-48*
CPY	Copy switch	*5-16*
CRC/CRE/CRO	Target cyclic redundancy check	*6-75*
CS	Code segment	*5-70*
CTL	Count limit	*7-7*
CX, CL, CH	Count register (low and high)	*5-70*
DAT	Data access status	*7-5*
DB	Display memory block	*6-17*
DC	Don't cares	*7-4*
DC1, DC2	Data comparators 1 and 2	*7-4*
DEL	Delete symbol/section	*5-120*
DES	Display WHEN/THEN statements	*7-11*
DFB	Set/display default base value	*5-86*
DI	Destination index	*5-70*
DIA	Display ASCII character string	*5-52*
DIS	Display disassembled memory	*6-36*
DM	Display memory map	*5-55*
DMA	DMA cycle	*5-67*
DME	Enable DME switch	*5-17*
DNL	Download	*5-38*
DR	Display microprocessor registers	*5-74*
DRT	Display raw Trace Memory	*5-94*
DS	Data segment	*5-70*
DT	Disassemble Trace Memory	*5-98*
DTA	Overlay enable for data access	*5-61*

Command	Description	Page
DTB	Disassemble Trace Memory backward	*5-101*
DTF	Disassemble Trace Memory forward	*5-101*
DX, DL, DH	Data register (low and high)	*5-70*
END	Line assembler exit	*6-32*
ES	Extra data segment	*5-70*
FIL	Fill memory with constant data	*6-21*
FIN	Find byte or word	*6-19*
FLX, FLL, FLH	Flags register (low and high)	*5-70*
FSI	Force special interrupt	*7-25*
FSX	Force special interrupt on instruction execution (switch)	*5-18*
GD0-7	General purpose data registers (0-7)	*5-88*
GR0-7	General purpose range registers (0-7)	*5-90*
GRO	Event monitor system group	*7-27*
HLT	Halt status	*7-5*
IAK	Interrupt acknowledge status	*7-5*
IDX	Repeat index register	*5-107*
IF	Instruction fetch status	*7-5*
ILG	Illegal memory access attribute	*5-57*
IOA	IO access status	*7-5*
IOP	IO mode pointer	*6-47*
IP	Instruction pointer register	*5-70*

Command	Description	Page
IRA	Internal range	*7-3*
LD	Load EEPROM data	*5-27*
LDV	Load vectors	*6-9*
LEN	Length (specifies range)	*4-11*
LIM	Repeat limit register	*5-107*
LOV	Load Overlay Memory	*5-62*
LSA	Logic State Probe comparator	*7-2*
LST	Decrement address in memory mode	*6-46*
M	Enter memory mode	*6-40*
MAC	Display macros	*5-103*
MAP	Define overlay memory map	*5-56*
MEM	Memory status	*7-5*
MIO	Enter I/O mode	*6-43*
MMP	Memory mode pointer	*5-83*
MMS	Memory mode status	*5-82*
MOD	Modulo	*4-17*
NBC	No bus cycles status	*7-5*
NMI	NMI cycle status	*7-5*
NOT	Logical event NOT	*4-15*
NXT	Increment address in memory mode	*6-46*
OFF	Display/disable switches	*5-9*
ON	Display/enables switches	*5-9*
OR	Logical event OR	*4-15*

Command	Description	Page
OVE	Overlay memory enable	*5-61*
OVL	Overlay memory status	*7-5*
PCB	Display PCB registers	*5-77*
PUR	Clear symbolic memory	*5-121*
QD1-6	Queue depth (1-6) status	*7-5*
QF	Queue flush cycle status	*7-5*
RBK	Run with breakpoints	*6-5*
RBV	Load vectors and run with breakpoints	*6-5*
RCS	Read chip select switch	*5-19*
RCT	Reset count limit	*7-19*
RD	Read status	*7-5*
RDY	Ready switch	*5-20*
RET	Display return and line feed	*5-124*
REV	Display software revisions	*5-123*
RIO	Read I/O status	*7-5*
RM	Read memory status	*7-5*
RNV	Run with new vectors	*6-5*
RO	Read only attribute	*5-57*
RST	Reset	*6-13*
RUN	Run emulation	*6-5*
RW	Read/write attribute	*5-56*
S1, S2	Status comparators 1 and 2	*7-5*
SAV	Save EEPROM data	*5-25*
SEC	Display section	*5-117*
SET	Set/display system parameters	*5-3*

Command	Description	Page
SF	Display special function menu	*6-52*
SF 0-3	RAM Tests	*6-52*
SF 4-12	Special functions (memory)	*6-52*
SF 24-32	Special functions (I/O)	*6-52*
SI	Source index register	*5-70*
SIA	Special interrupt address	*7-25*
SP	Stack pointer register	*5-70*
SS	Stack segment register	*5-70*
STA	Stack data status	*7-5*
STI	Step through interrupts	*5-22*
STP	Step and stop	*6-7*
SYM	Display symbols	*5-116*
TAR	Target access status	*5-96*
TCT	Terminal port control	*5-35*
TE0-2	Timer switches	*5-23*
TGR	Enable trigger output	*7-23*
TGT	Target memory attribute	*5-57*
THE	THEN used in event statements	*7-1*
TO	To	*4-11*
TOC	Toggle counting	*7-19*
TOT	Toggle trace memory	*7-16*
TRA	Transparent mode	*5-33*
TRC	Trace event	*7-16*
TST	Test register for repeats	*5-92*

Command	Description	Page
UPL	Upload	*5-44*
UPS	Upload symbols	*5-46*
VBL	Verify block data	*6-22*
VBM	Verify block move	*6-25*
VFO	Verify overlay against target	*6-28*
VFY	Verify serial data	*5-43*
WAI	Wait until emulation is broken	*6-11*
WDM	Word mode	*4-24*
WHEN/THEN	WHEN/THEN Statements	*7-9*
WIO	Write IO status	*7-5*
WM	Write memory status	*7-5*
WR	Write status	*7-5*
WRD	Word status	*7-5*
X	Don't care	*7-4*
X	Exit memory mode	*6-32*
XRA	External range	*7-3*

APPENDIX B

Table of Contents

Error Messages

ERROR MESSAGES

Error Messages

Error messages are divided into 3 categories:

1. Target Hardware
2. Emulator Hardware
3. Target Software

TARGET HARDWARE ERROR MESSAGES

HOLD ACKNOWLEDGE/BUS GRANTED	This message is displayed when a hold acknowledge has been asserted for longer than 2.2 ms. When the microprocessor regains control of the bus, the message is removed. This message is caused by one of two conditions: When a DMA (direct memory access) controller takes over the bus by asserting the hold line, or when the microprocessor is running in a multiprocessor environment. This message is generally not an error message but rather a statement of what the processor is doing.

NO BUS CYCLES

This error message indicates that no ALE's (Address Latch Enable) were detected for at least 2.2 ms or longer, and no other error conditions are found.

When no ALE's are detected the controller checks for other fault conditions, including proper target VCC, a functional clock, and whether the processor is halted, waiting, reset or the bus is granted. If any of these other conditions exist then the appropriate message for that condition is displayed. If no other fault condition is found, the NO BUS CYCLES message is displayed.

NO CLOCK

The microprocessor must have a clock frequency within the range of 1.2 Mhz to 9 Mhz or the message NO CLOCK is displayed.

If there is no clock from the target, the user is given the option of selecting an internal clock when the Emulator is powered up (see page 5-9). However after an external clock has been selected and the NO CLOCK message is displayed, the only way to return to an internal clock is to reset the system.

PROCESSOR HALTED

A halt (**HLT**) instruction has been executed and the microprocessor has remained halted for greater than 2.2 ms. The microprocessor is in a run state and commands can still be entered at the keyboard.

NOTE 1: It is not possible to break on a **HLT** instruction or status. If you want to break on the **HLT** instruction it is necessary to set a breakpoint at an address one instruction before the **HLT**.

NOTE 2: Normally when a **HLT** instruction is executed, the microprocessor waits for a reset or an interrupt to bring it out of that condition. When single stepping, the emulator uses an **NMI** to return to the internal world. Therefore when a **HLT** instruction is encountered it is executed and the processor goes on to the next instruction because the microprocessor was satisfied by the **NMI** that took it out of the **HLT** condition.

PROCESSOR WAITING

The microprocessor is waiting for a RDY (ready) to be returned. This message displays only if the microprocessor has been waiting for greater than 2.2 ms. When the condition has been corrected the message is removed.

NOTE 1: It is necessary to use target RDY when overlaying dynamic RAM that uses the RDY line to halt microprocessor activity during refresh cycles. When a refresh cycle occurs on many systems the RDY line is held in the NOT RDY state until the refresh is complete. If an internal RDY is used, the microprocessor will not honor the REFRESH cycles and dynamic memory will be corrupted. The choice of internal or external RDY while using overlay memory is made by using the RDY switch (see page 5-20).

NOTE 2: When overlaying nonexistent code space it is necessary to use the internal RDY. Users may want to overlay nonexistent code space (an area not decoded in their hardware) to patch in code.

NOTE 3: When selecting internal or external RDY for areas overlayed, that particular RDY is selected for all overlay. It is not possible therefore to overlay both dynamic RAM and nonexistent RAM at the same time.

RESET ASSERTED

This indicates that a reset from the target has been asserted for greater than 2.2 ms. When the reset is released then the message is removed. However, if the reset is less than 2.2 ms the message is not displayed. Using an oscilloscope, verify that the reset line is in fact being held reset. There are some operating systems that may normally hold the microprocessor reset until needed. If the reset line is not being held reset at the probe tip, unplug the emulator and verify the condition in the NULL TARGET mode.

EMULATOR HARDWARE ERROR MESSAGES

POD CPU NOT INITIALIZED

When a reset occurs, (power up, CTRL Z, or **RST**) the controller and the emulator begin an initialization routine to establish communication. If this initialization routine fails to complete, this message is displayed. This is an internal pod, emulator, controller board problem. Correct the problem by reseating boards, cycling power, and verifying that the microprocessor is correctly installed in the pod, or replacing the microprocessor in the pod.

POD CPU NOT RESPONDING

Whenever a **STP** command is executed, or a memory command is executed during emulation, the ES language software looks to see if any errors occurred during execution of the command. The emulator then checks if the command completed. If it did not complete the emulator 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 the message POD CPU NOT RESPONDING is displayed. Correct the problem by resetting the system and repeating the command.

SYSTEM RESET ERROR

When a reset (power up, CTRL Z, or **RST**) has been executed from the emulator controller and the emulator board does not acknowledge this, then a SYSTEM RESET ERROR message displays. This situation is an internal pod, emulator, or controller board problem. Try reseating boards, reseating pod cables, and cycling power.

TARGET SOFTWARE ERROR MESSAGES

MEMORY ACCESS VIOLATION

The target program has attempted to access an area of target mapped as illegal (ILG). **DM** assists in determining which areas are mapped as illegal. **DRT** helps determine where the program was making the access.

MEMORY WRITE VIOLATION

If the target program attempts to write to the RAM overlay in an area that is mapped READ ONLY, this error occurs. Use the **DM** command and the raw trace (**DRT**) to look for write cycles. **DM** assists in determining which areas are mapped as illegal. **DRT** helps determine where the program was making the access.

APPENDIX C

Table of Contents

Serial Data Formats

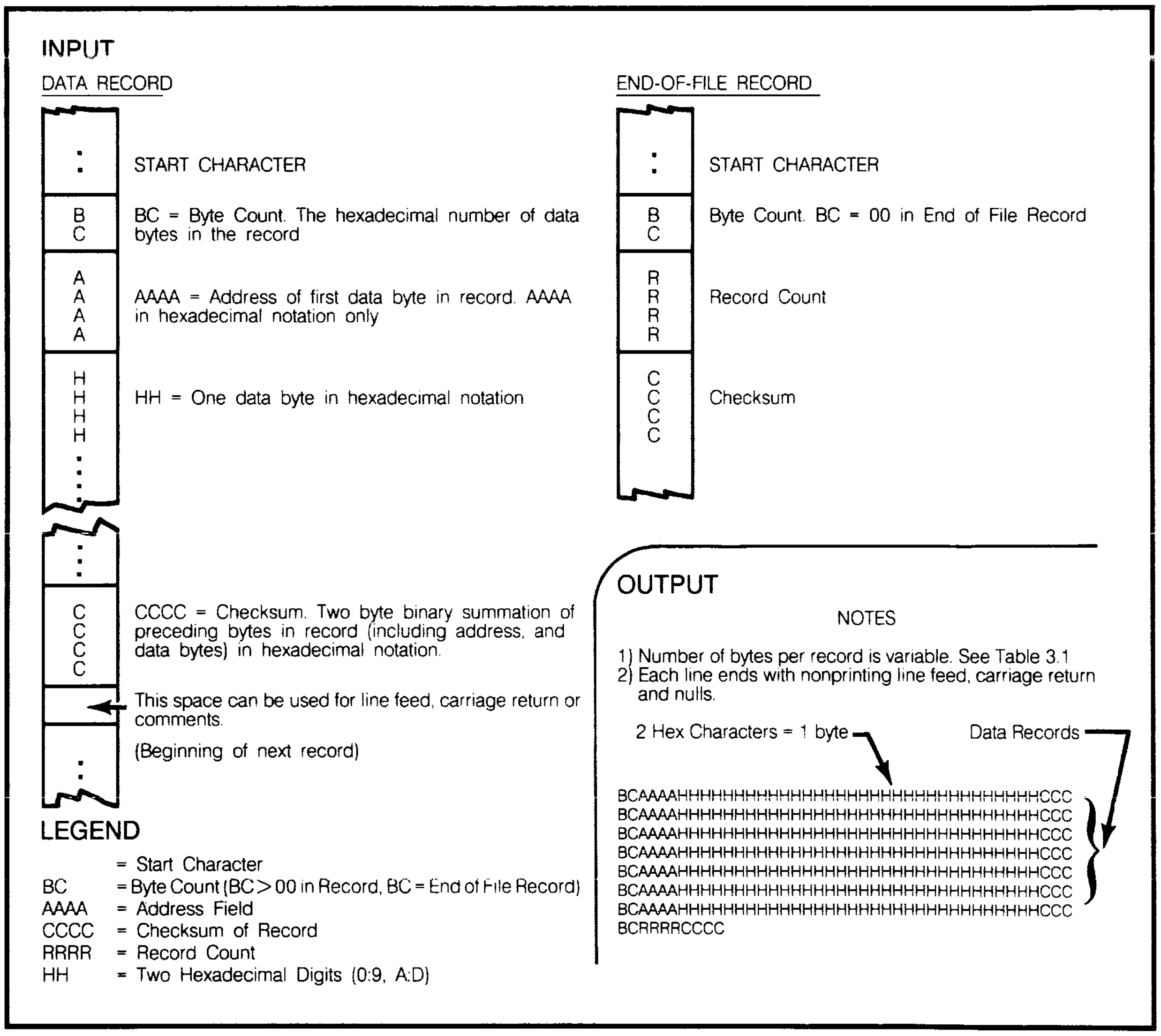
Appendix C

SERIAL DATA FORMATS

In order to download a program into target memory, the ES 1800 needs some way to receive this data in an intelligible format. This Appendix describes the downloading formats which the ES 1800 understands.

MOS Technology Format

Figure 8. Specifications for MOS Technology Data Files Copyright 1983, Data I/O Corporation; reprinted by permission.



INPUT

DATA RECORD

START CHARACTER

BC = Byte Count. The hexadecimal number of data bytes in the record

AAAA = Address of first data byte in record. AAAA in hexadecimal notation only

HH = One data byte in hexadecimal notation

CCCC = Checksum. Two byte binary summation of preceding bytes in record (including address, and data bytes) in hexadecimal notation.

This space can be used for line feed, carriage return or comments.

(Beginning of next record)

END-OF-FILE RECORD

START CHARACTER

Byte Count. BC = 00 in End of File Record

Record Count

Checksum

LEGEND

	= Start Character
BC	= Byte Count (BC > 00 in Record, BC = End of File Record)
AAAA	= Address Field
CCCC	= Checksum of Record
RRRR	= Record Count
HH	= Two Hexadecimal Digits (0:9, A:D)

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.

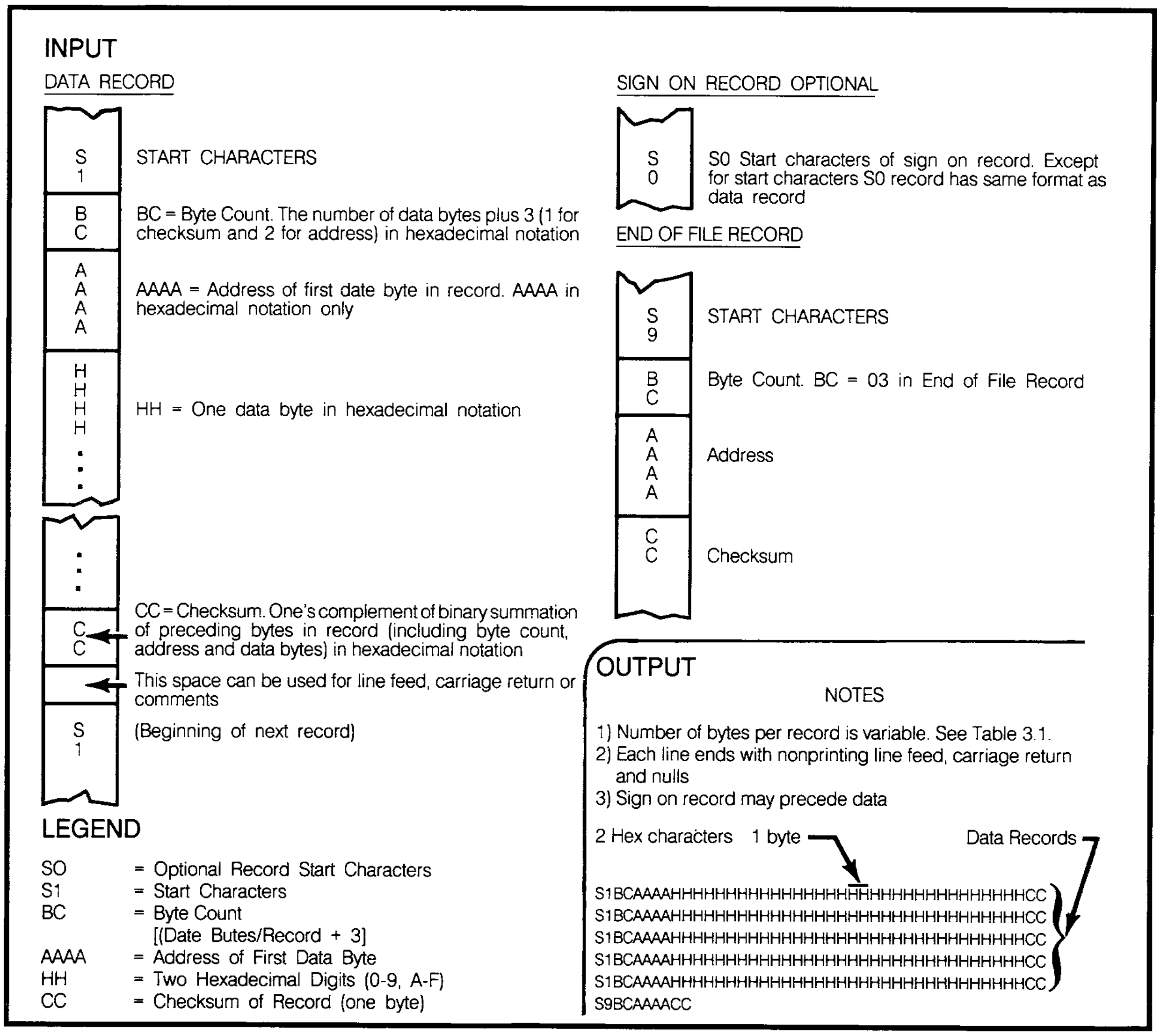
2 Hex Characters = 1 byte

Data Records

```
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCC
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCC
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCC
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCC
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCC
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCC
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCC
BCRRRRCCCC
```

Motorola Exorcisor Format

Figure 9. Specifications for Motorola Exorciser/16-BM Data Files/ Copyright 1983, Data I/O Corporation; reprinted by permission.



INPUT

DATA RECORD

S 1	START CHARACTERS
B C	BC = Byte Count. The number of data bytes plus 3 (1 for checksum and 2 for address) in hexadecimal notation
A A A A	AAAA = Address of first date byte in record. AAAA in hexadecimal notation only
H H H H • • •	HH = One data byte in hexadecimal notation
• • •	
C C	CC = Checksum. One's complement of binary summation of preceding bytes in record (including byte count, address and data bytes) in hexadecimal notation
	This space can be used for line feed, carriage return or comments
S 1	(Beginning of next record)

SIGN ON RECORD OPTIONAL

S 0	S0 Start characters of sign on record. Except for start characters S0 record has same format as data record

END OF FILE RECORD

S 9	START CHARACTERS
B C	Byte Count. BC = 03 in End of File Record
A A A A	Address
C C	Checksum

LEGEND

SO	= Optional Record Start Characters
S1	= Start Characters
BC	= Byte Count [(Date Butes/Record + 3]
AAAA	= Address of First Data Byte
HH	= Two Hexadecimal Digits (0-9, A-F)
CC	= Checksum of Record (one byte)

OUTPUT

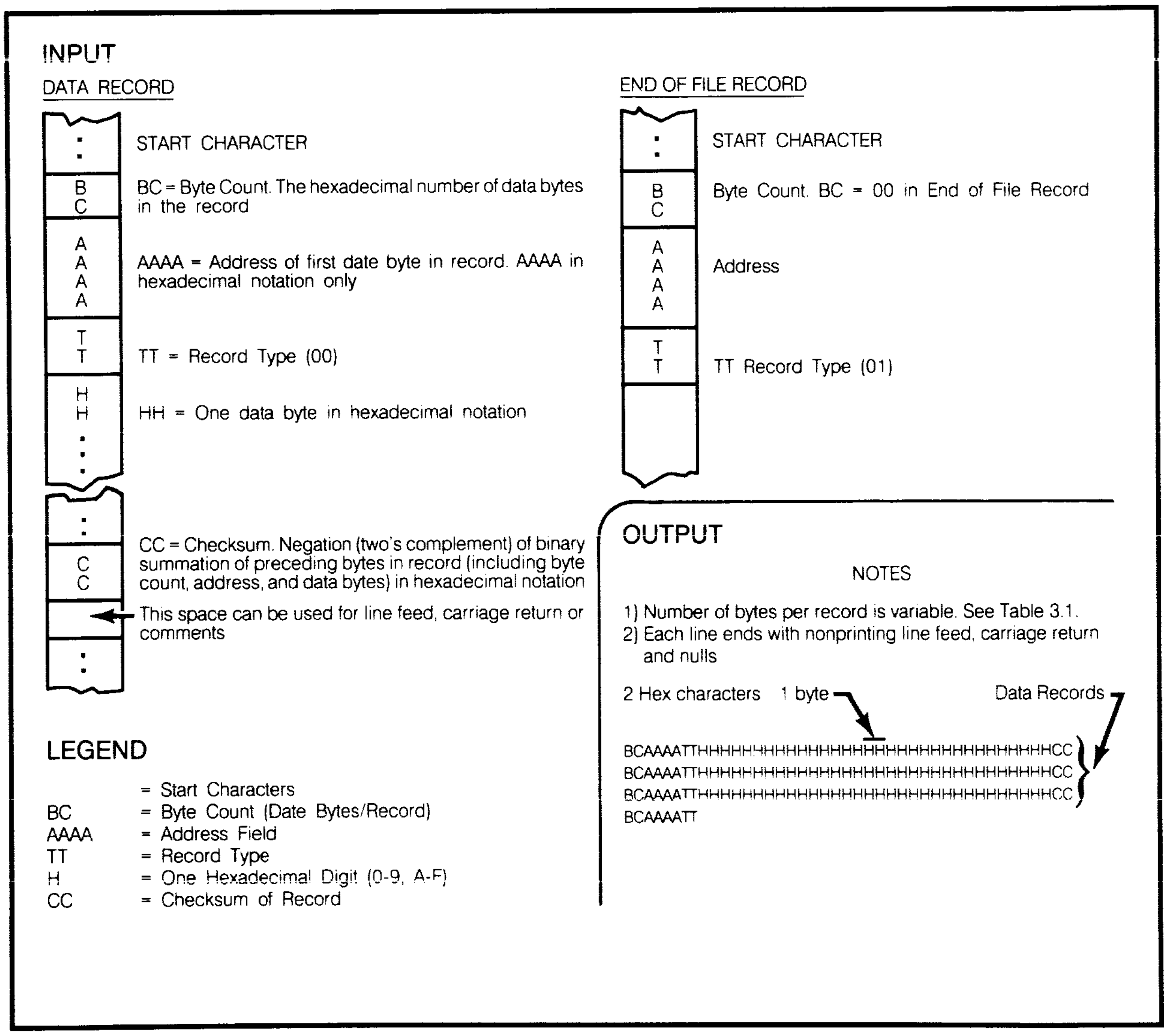
NOTES

1) Number of bytes per record is variable. See Table 3.1.
2) Each line ends with nonprinting line feed, carriage return and nulls
3) Sign on record may precede data

```
S1BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
S1BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
S1BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
S1BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
S1BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
S9BCAAAACC
```

Intel Intellec Format

Figure 10. Specifications for Intel Intellec/8/MDS Data Files/ Copyright 1983, Data I/O Corporation; reprinted by permission.



INPUT

DATA RECORD

START CHARACTER

BC = Byte Count. The hexadecimal number of data bytes in the record

AAAA = Address of first date byte in record. AAAA in hexadecimal notation only

TT = Record Type (00)

HH = One data byte in hexadecimal notation

CC = Checksum. Negation (two's complement) of binary summation of preceding bytes in record (including byte count, address, and data bytes) in hexadecimal notation

This space can be used for line feed, carriage return or comments

END OF FILE RECORD

START CHARACTER

Byte Count. BC = 00 in End of File Record

Address

TT Record Type (01)

LEGEND

:	= Start Characters
BC	= Byte Count (Date Bytes/Record)
AAAA	= Address Field
TT	= Record Type
H	= One Hexadecimal Digit (0-9, A-F)
CC	= Checksum of 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

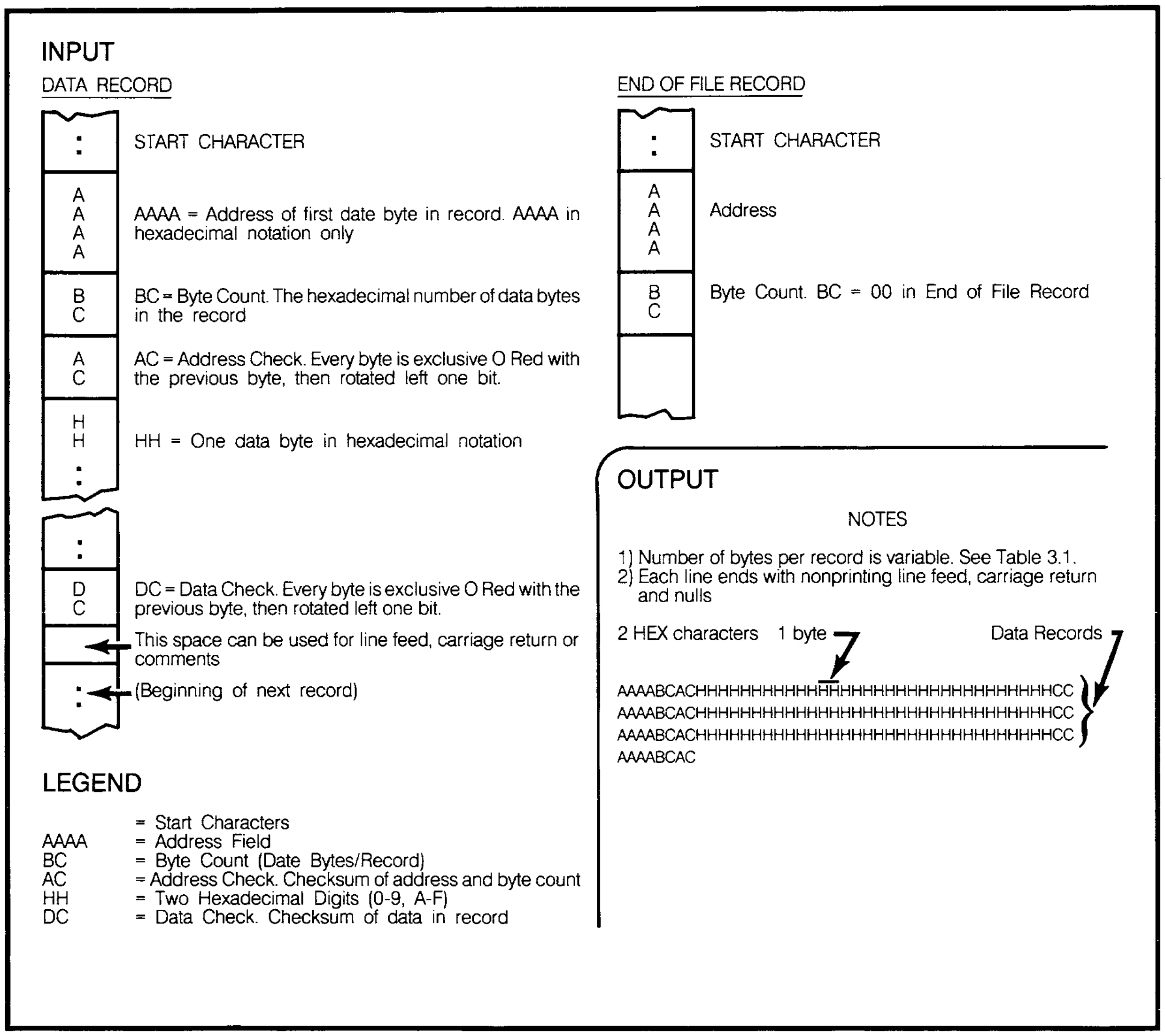
2 Hex characters 1 byte

Data Records

```
BCAAAATTHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
BCAAAATTHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
BCAAAATTHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
BCAAAATT
```

Signetics/Absolute Object File Format

Figure 11. Specifications for Signetics/Absolute Object Data Files Copyright 1983, Data I/O Corporation; reprinted by permission.



INPUT

DATA RECORD

:	START CHARACTER
A A A A	AAAA = Address of first date byte in record. AAAA in hexadecimal notation only
B C	BC = Byte Count. The hexadecimal number of data bytes in the record
A C	AC = Address Check. Every byte is exclusive O Red with the previous byte, then rotated left one bit.
H H :	HH = One data byte in hexadecimal notation
:	
D C	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
A A A A	Address
B C	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

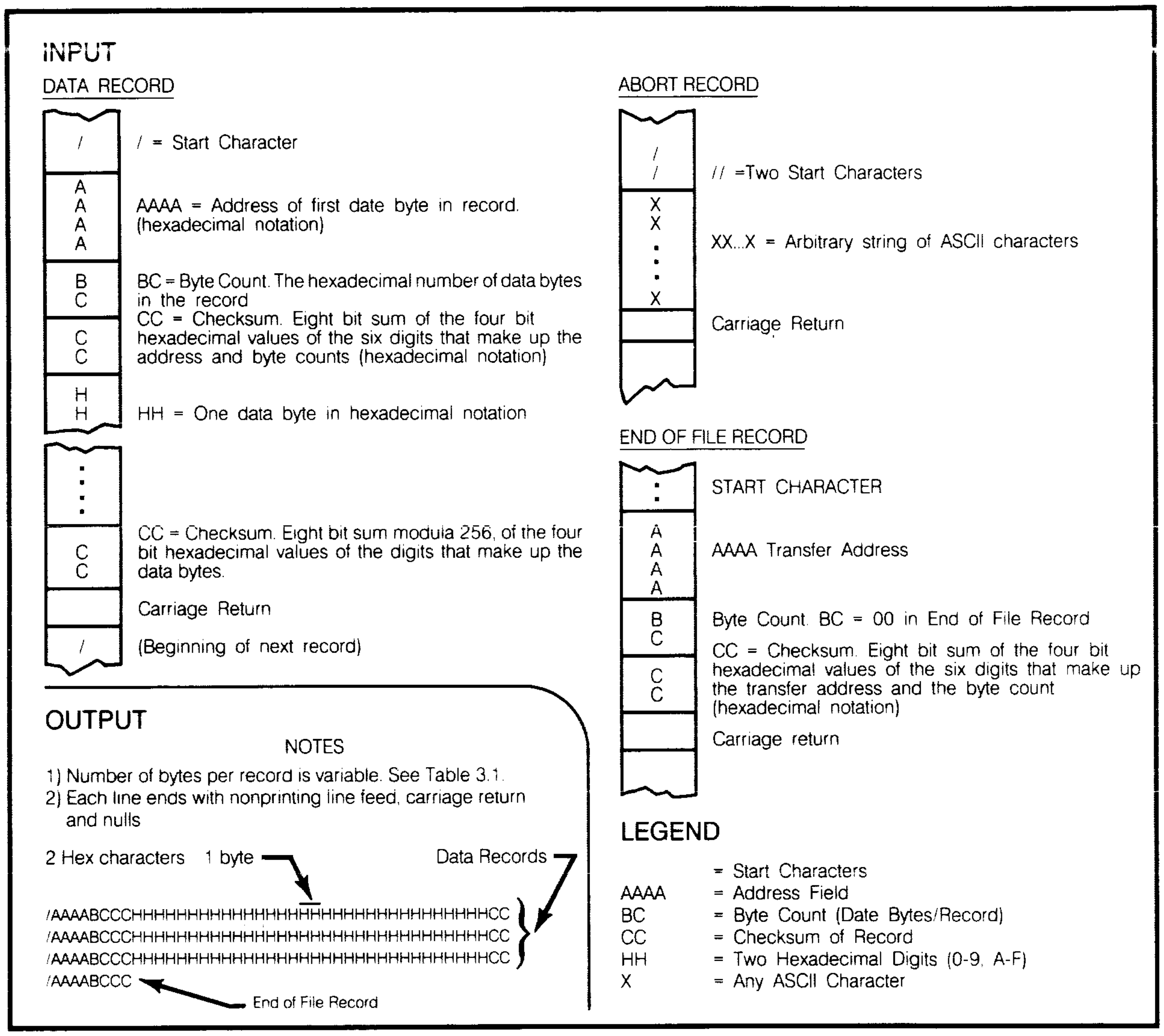
1) Number of bytes per record is variable. See Table 3.1.
2) Each line ends with nonprinting line feed, carriage return and nulls

2 HEX characters 1 byte — Data Records

```
AAAABCACHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
AAAABCACHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
AAAABCACHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
AAAABCAC
```

Tektronix Hexadecimal Format

Figure 12. Specifications for Tektronix Hexadecimal Data Files Copyright 1983, Data I/O Corporation; reprinted by permission.



/ = Start Character

AAAA = Address of first date byte in record. (hexadecimal notation)

BC = Byte Count. The hexadecimal number of data bytes in the record

CC = Checksum. Eight bit sum of the four bit hexadecimal values of the six digits that make up the address and byte counts (hexadecimal notation)

HH = One data byte in hexadecimal notation

CC = Checksum. Eight bit sum modula 256, of the four bit hexadecimal values of the digits that make up the data bytes.

Carriage Return

(Beginning of next record)

ABORT RECORD

// =Two Start Characters

XX...X = Arbitrary string of ASCII characters

Carriage Return

END OF FILE RECORD

START CHARACTER

AAAA Transfer Address

Byte Count BC = 00 in End of File Record

CC = Checksum. Eight bit sum of the four bit hexadecimal values of the six digits that make up the transfer address and the byte count (hexadecimal notation)

Carriage return

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

/AAAABCCCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
/AAAABCCCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
/AAAABCCCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
/AAAABCCC  <---- End of File Record
```

LEGEND

	= Start Characters
AAAA	= Address Field
BC	= Byte Count (Date Bytes/Record)
CC	= Checksum of Record
HH	= Two Hexadecimal Digits (0-9, A-F)
X	= Any ASCII Character

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.

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

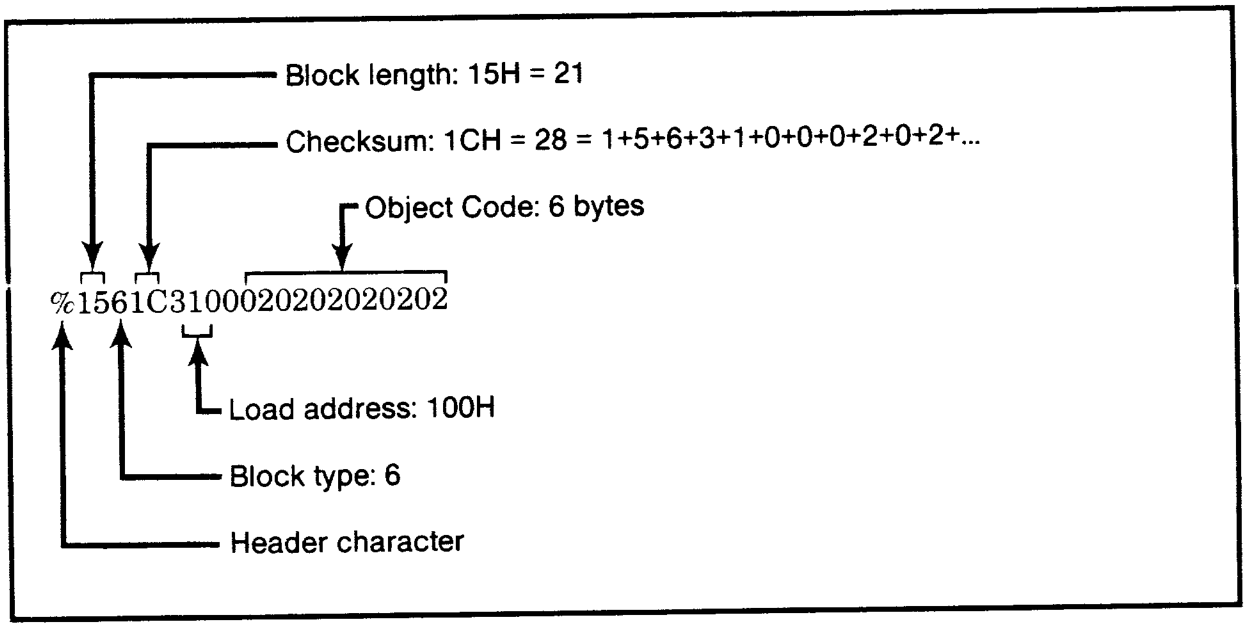


Figure 14. Extended Tekhex Termination Block

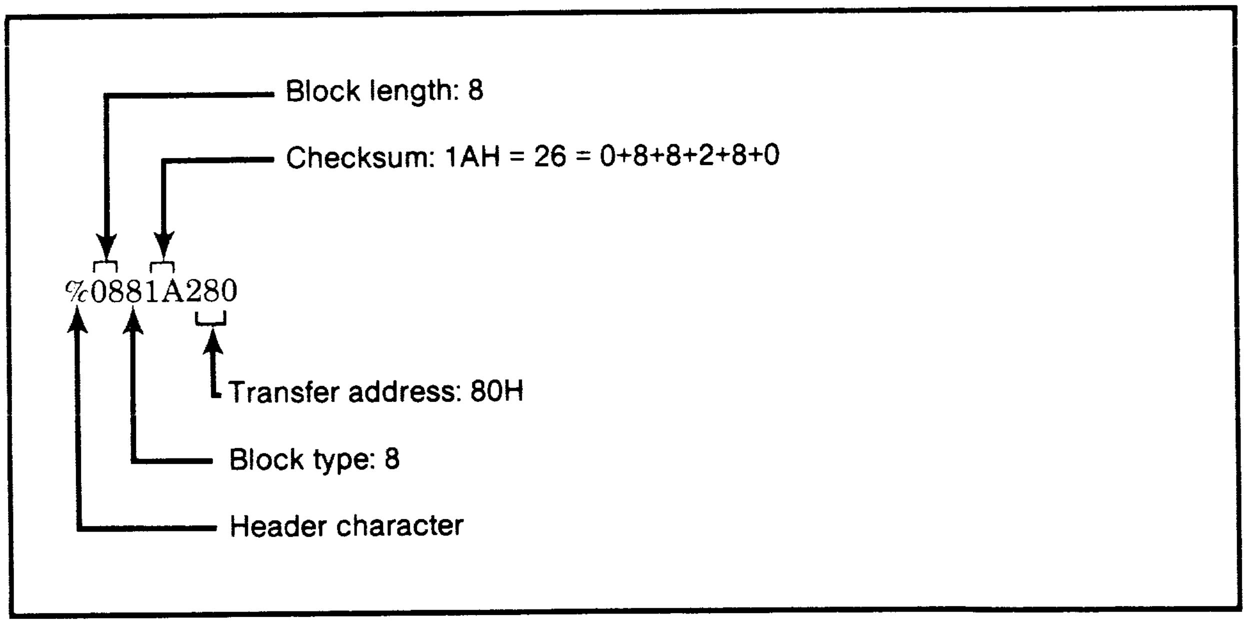
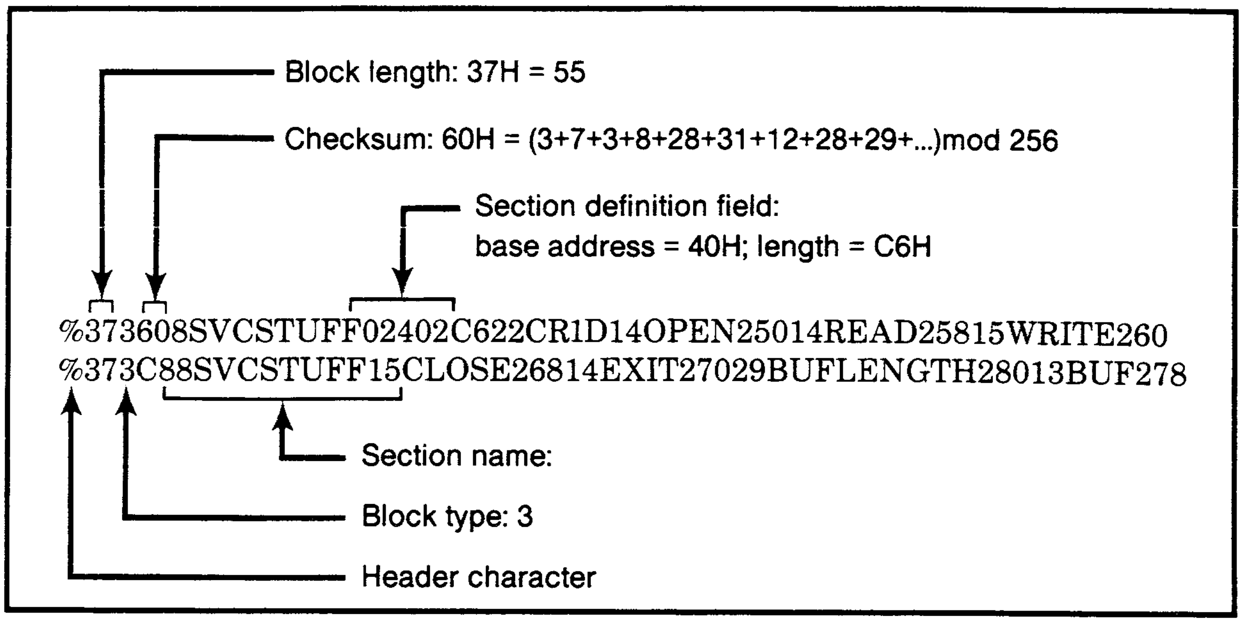


Figure 15. Extended Tekhex Symbol Block


Block length: 37H = 55
Checksum: 60H = (3+7+3+8+28+31+12+28+29+...)mod 256
Section definition field:
base address = 40H; length = C6H
%373608SVCSTUFF02402C622CR1D14OPEN25014READ25815WRITE260
%373C88SVCSTUFF15CLOSE26814EXIT27029BUFLENGTH28013BUF278
Section name:
Block type: 3
Header character

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
S1130000285F245F2212226A000424290000082337CA
S113001000020000800082629001853812341001813
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

Applied
Microsystems
Corporation

5020 148th Ave. N.E.
P.O. Box 97002
Redmond, WA 98073-9702

(206) 882-2000
1-800-426-3925
TRT TELEX 185196
FAX (206) 883-3049

Applied Microsystems Corporation maintains a worldwide network of Direct Sales Offices and Representative Organizations committed to quality service and support. For the address and phone number of the Applied Microsystems Corporation Sales Office nearest you, call 1-800-426-3925 (in WA, call 206-882-2000).

Applied Microsystems' products are available throughout Europe. For the address and phone number of your nearest distributor, contact: Applied Microsystems Corporation, Ltd. Chiltern Court, High Street, Wendover, Aylesbury, Bucks, HP22 6EP, England. Tel 44(0) 296-625462