ES 1800 Emulator
User's Manual
80186/88, 80C186/C188
and 80C186EB/C188EB
Microprocessors

January 1992
P/N 922-17003-00
Replaces P/N 922-00003-06
Copyright © 1992 Applied Microsystems Corporation.
All rights reserved.
DEC and VAX are trademarks of Digital Equipment Corporation
Ethernet is a trademark of Xerox Corporation
GeneProbe II, GeneLink, and ACCESS are trademarks of Genesis Microsystems Corporation.
Hewlett-Packard is a registered trademark of Hewlett-Packard Corporation
Intel and Link-86 are registered trademarks of Intel Corporation
InterTools is a trademark of Intermetrics Systems Software, Inc.
Lattice is a trademark of Lattice Corporation
MetaWare and High C are registered trademarks of MetaWare Incorporated
Microsoft, MS-DOS, and CodeView are registered trademarks of Microsoft Corporation.
PC, IBM XT and IBM AT are trademarks of IBM Corporation
Phar Lap is a trademark of Phar Lap Software.
Soft-Scope is a registered trademark of Concurrent Sciences
Sun, Sun-3, Sun-4, NFS, and PC-NFS are trademarks of Sun Microsystems, Inc.
Turbo C is a registered trademark of Borland International, Inc.
UNIX is a registered trademark of AT&T.
VALIDATE is a registered trademark of Applied Microsystems Corporation
376, 386, 387, 486, and iRMX are trademarks of Intel Corporation
## Contents

- No Target System ................................................................. 2-5
- Getting Started with ESL .................................................... 2-5
- Test Run of System ............................................................... 2-6
  1. Initialize The ES 1800 ....................................................... 2-6
  2. Map Overlay Memory ....................................................... 2-7
  3. Test RAM ........................................................................ 2-7
  4. Enter Program ................................................................. 2-7
  5. Verify The Program ........................................................ 2-8
  6. Run The ES 1800 ............................................................... 2-8
  7. Stop The Program ............................................................ 2-8
  8. Display The Trace Buffer .................................................. 2-9
  9. Set A Breakpoint ............................................................. 2-9
  10. Initialize Peripheral Control Registers
      8018x, 80C18x only ....................................................... 2-9
  11. Initialize Peripheral Control Registers
      80C18xEB only ............................................................. 2-14

## Section 3
### HARDWARE

- Emulator Chassis ............................................................... 3-1
- System Grounds ................................................................. 3-1
- Emulator Control Boards .................................................... 3-1
- ES 1800 Chassis Front Panel .............................................. 3-5
- ES 1800 Chassis Rear Panel ................................................ 3-5
- Pod .................................................................................... 3-7
- Saving Desk Space ............................................................. 3-9
- Time Stamp Module .......................................................... 3-11
- Logic State Analyzer (LSA) .................................................. 3-12
  LSA Timing Strobe ............................................................. 3-12
- Ports ................................................................................... 3-14
  Serial Ports ....................................................................... 3-14
  Data Requirements ............................................................ 3-15
- Maintenance ........................................................................ 3-17
  Cables .............................................................................. 3-17
  Probe tip ........................................................................... 3-17
  Cleaning the Fan Filter ....................................................... 3-17
  Parts .................................................................................. 3-19
- Troubleshooting ................................................................. 3-20
- ES 1800 Emulator Specifications ............................................ 3-21
  Input Power ...................................................................... 3-21
  Environmental ................................................................. 3-21
  Physical ............................................................................. 3-21
Section 4
PREPARING FOR EMULATION

Terms ........................................................................................................ 4-2
Establish Communication with the Emulator ...................................... 4-3
  Serial Communication ................................................................... 4-3
  Setup Commands ........................................................................ 4-5
  Port Dependent Commands ......................................................... 4-5
  SCSI Communication .................................................................. 4-5
Set Up Target Environment ................................................................ 4-6
  Map Overlay Memory ............................................................... 4-7
  Download Files ........................................................................ 4-9
  Check Registers .......................................................................... 4-11
  Set Up Soft Switches .................................................................. 4-27
Run Your Program ............................................................................ 4-29
  Break Emulation ......................................................................... 4-30
Set Up Breakpoints .......................................................................... 4-32
  Set Up the Event Monitor System ............................................. 4-32
  Structure .................................................................................... 4-33
  Define Events ............................................................................ 4-34
  Define WHEN/THEN Statements .............................................. 4-40
  Event Monitor System Examples .............................................. 4-41
  Using Software Debuggers ...................................................... 4-45
Isolate a Problem .............................................................................. 4-47
  Run Program from Overlay ...................................................... 4-48
  Examine the Trace Memory ....................................................... 4-48
  Check CPU Registers ............................................................... 4-49
  Single Step Through Program .................................................. 4-50
  Miscellaneous Useful Commands .............................................. 4-50
Modify Your Program ........................................................................ 4-51
  Memory Commands .................................................................... 4-52
  Line Assembler .......................................................................... 4-53
  Memory Mode ............................................................................ 4-54
  I/O Mode ................................................................................... 4-55
Shortcuts ............................................................................................ 4-56
  Use Symbols Rather than Addresses ........................................ 4-57
  Repeat Operators ....................................................................... 4-61
  Macros ....................................................................................... 4-62
  General Purpose Registers ....................................................... 4-63
  Save Setup to EEPROM ........................................................... 4-63
  Configure System for Two Users .............................................. 4-63
  Clear Commands ....................................................................... 4-64
Section 5
BRINGING UP HARDWARE
RAM Tests ................................................................. 5-2
Scope Loops .............................................................. 5-2
Miscellaneous Special Functions ............................... 5-3

Section 6
TIME STAMP MODULE
Possible Measurements .............................................. 6-1
Using the Time Stamp Counter Value as a Condition .... 6-2
Installation ............................................................. 6-3
Hardware Installation ............................................... 6-3
Software Installation ............................................... 6-4
Using the Time Stamp Module ................................... 6-5
Getting Started ........................................................ 6-5
Steps for Using the Time Stamp Module ..................... 6-6
Examples ................................................................. 6-12
Measuring Elapsed Time ........................................... 6-12
Counting Occurrences ............................................. 6-20
Using the Time Stamp Counter Value as a Condition .... 6-24

Section 7
ALPHABETICAL COMMAND REFERENCE
Introduction ............................................................ 7-1
@: Read/Write Memory ............................................. 7-2
*: Symbol and Section Definition ............................... 7-4
/: Repeat Command Line .......................................... 7-6
*: Repeat Command Line .......................................... 7-7
_: Define/Use Macros .............................................. 7-8
ASM: Line Assembler ............................................. 7-9
BAS: Set/Display Register Default Base .................... 7-12
BKX: Break On Instruction Execution ....................... 7-13
BMO: Block Move ................................................... 7-14
BRK: Break Emulation ............................................ 7-16
BUS: Display Status Of Bus Status Lines ................... 7-18
BTE: Bus Timeout Enable (80C18x and 80C18xEB only) 7-19
BTO: Bus Timeout Register (80C18x and 80C18xEB only) 7-20
BYM: Set Global Data Length .................................... 7-21
CCT: Computer Port Control .................................... 7-23
CDH: Clear DMA Halt (8018x and 80C18x only) .......... 7-24
CES: Clear When/Then Statements ............................ 7-25
CK: Internal/External Clock ..................................... 7-26
CLK: Read Target System Clock ............................... 7-27
CLM: Clear Memory Map ......................................... 7-28
CLR: Clear CPU Registers .............................................................. 7-29
CMC: Clear Macros ......................................................................... 7-30
CNT: Decrement Hardware Counter ............................................... 7-31
COM: Communication With Target Programs ................................ 7-34
CPY: Copy Data To Both Ports ....................................................... 7-38
CRC,CRE,CRO: Target Cyclic Redundancy Check ....................... 7-39
CTS: Convert Time Stamp .............................................................. 7-40
DB: Display Memory Block ............................................................. 7-41
DEL: Delete A Symbol Or Section .................................................. 7-43
DES: Display Event Specifications ................................................. 7-44
DFB: Default Base ........................................................................... 7-45
DIA: Display Character String ........................................................ 7-46
DIS: Memory Disassembler ............................................................. 7-48
DM: Display Memory Map ............................................................. 7-49
DME: Enable Data (8018x and 80C18x only).................................. 7-50
DNL: Download File ....................................................................... 7-51
DNV: Verify Download Data (80C18x and 80C18xEB only) ........ 7-52
DR: Display/Load Microprocessor Registers .................................. 7-53
DRT: Display Raw Trace Bus Cycles ............................................. 7-55
DT: Disassemble Trace Memory ..................................................... 7-59
DTB, DTF: Disassemble Trace Page ............................................... 7-61
FIL: Fill Operator ............................................................................. 7-62
FIN: Find Pattern In Memory .......................................................... 7-63
FSI: Force Special Interrupt ........................................................... 7-64
FSX: FSI On Instruction Execution ................................................. 7-66
GD: General Purpose Data Registers ............................................... 7-67
GR: General Purpose Address Registers ......................................... 7-68
GRO: Change Event Groups ........................................................... 7-69
IDP: Interrupts During Pause (80C18x and 80C18xEB only) .......... 7-70
IHE: Ignore Halt Errors (80C18x and 80C18xEB only) ............... 7-72
IOP: I/O Mode Pointer ..................................................................... 7-73
LD: Load System Variables From EEPROM .................................. 7-74
LDV: Load Reset Vectors ................................................................ 7-75
LOV: Load Overlay Memory .......................................................... 7-76
M: Enter Memory Mode ................................................................. 7-77
MAC: Display Defined Macros ....................................................... 7-79
MAP: Set Memory Map .................................................................. 7-80
MIO: Enter I/O Mode ........................................................................ 7-83
MMP: Memory Mode Pointer ......................................................... 7-85
ON/OFF: Switch Setting ................................................................. 7-87
OVE: Overlay Memory Enable ....................................................... 7-92
OVS: Overlay Memory Speed (80C18x and 80C18xEB only) ....... 7-93
PCB: Display PCB Registers ......................................................... 7-95
PCS: Enable Chip Selects (80C18x and 80C18xEB only) ............. 7-98
PPT: Trace Peaks and Pokes (80C18x and 80C18xEB only) ....... 7-99
Contents

PRE: DRAM Refresh During Pause
(80C18x and 80C18xEB only)............................................................. 7-100
PUR: Delete All Symbols And Sections............................................. 7-102
RBK: Run Target Program .............................................................. 7-103
RBV: Run Target Program .............................................................. 7-104
RCS: Read Chip Select 8018x and 80C18x only............................... 7-105
RCT: Reset Hardware Counter ....................................................... 7-106
RDY: Select Internal or External Ready Signal ................................ 7-107
RET: Display A Blank Line ............................................................. 7-108
REV: Display The Software Revision Dates .................................... 7-109
RNV: Run Target Program ............................................................. 7-110
RSS: Read Serial Status 80C18xEB only ......................................... 7-111
RST: Reset ..................................................................................... 7-112
RUN: Run Target Program ............................................................. 7-113
SAV: Save System Variables In EEPROM ....................................... 7-114
SEC: Display Section ...................................................................... 7-115
SET: Set Up Parameters .................................................................. 7-116
SF: Special Functions List .............................................................. 7-120
SF 0: Simple RAM Test, Single Pass .............................................. 7-121
SF 1: Complete RAM Test, Single Pass ......................................... 7-123
SF 2: Simple RAM Test, Looping .................................................. 7-124
SF 3: Complete RAM Test, Looping .............................................. 7-125
SF 4: Toggle Data At Address ....................................................... 7-126
SF 5: Peeks Into The Target System .............................................. 7-127
SF 6: Pokes Into The Target System .............................................. 7-128
SF 7: Write Alternate Patterns ..................................................... 7-129
SF 8: Write Pattern Then Rotate .................................................... 7-130
SF 9: Write Data Then Read ......................................................... 7-132
SF 10: Write Incrementing Value ................................................. 7-133
SF 11: Read Data Over An Entire Range ....................................... 7-134
SF 12: Cyclic Redundancy Check .................................................. 7-135
SF 13: Toggle Data At Address ..................................................... 7-136
SF 14: Peeks Into The Target System ............................................ 7-137
SF 15: Pokes Into The Target System ............................................ 7-138
SF 16: Write Alternate Patterns .................................................... 7-139
SF 17: Write Pattern Then Rotate .................................................. 7-140
SF 18: Write Data Then Read ....................................................... 7-142
SF 19: Write Incrementing Value .................................................. 7-143
SF 20: Read Data Over An Entire Range ....................................... 7-144
SF 21: Step Through Interrupts ..................................................... 7-145
SF 22: Stop And Step Target System ............................................. 7-146
SF 23: Display Symbols ................................................................. 7-147
SF 24: Dynamic Trace Capture Enable ......................................... 7-148
SF 25: Terminal Port Control ......................................................... 7-149
SF 26: Timers ............................................................................... 7-150
Section 8
ES LANGUAGE
  Structure of the ES Language ............................................................ 8-1
  Notes on ESL ..................................................................................... 8-5
  Help .................................................................................................. 8-16
  Log In Banner .................................................................................. 8-19
  Prompts ............................................................................................ 8-21
  Special Modes .................................................................................. 8-22
  Special Characters ............................................................................ 8-24
  Errors ............................................................................................... 8-25
  ES Language Error Messages .......................................................... 8-26

Appendix A
ERROR MESSAGES
  Target Hardware Error Messages .................................................... A-1
  Emulator Hardware Error Messages .............................................. A-4
  Target Software Error Messages .................................................... A-5

Appendix B
SERIAL DATA FORMATS
  MOS Technology Format ................................................................. B-2
  Motorola Exorcisor Format ............................................................ B-3
  Intel Intellec Format ........................................................................ B-4
  Signetics/Absolute Object File Format .......................................... B-5
  Tektronix Hexadecimal Format ....................................................... B-6
  Extended Tekhex Format ................................................................. B-7
Unpacking and Inspection

Your ES 1800 emulator has been inspected and tested for electrical and mechanical defects before shipping, then configured for the line voltage requested. Although the emulator was carefully packed, check it for possible transit damage and verify that the following components are present.

If you find any damage, file a claim with the carrier and notify Applied Microsystems Corporation. In the United States and Canada, call Customer Support at 800-ASK-4AMC (206-882-2000 in Washington). Outside the U.S. and Canada, please contact your local sales office or representative. Before turning on the emulator, please follow the instructions in Section 2, Getting Started.

Standard Equipment

1. Emulator chassis with power cord, includes two boards: main control board and trace and break board
2. Processor specific equipment: emulation board and either an 80186/188 pod, an 80C186/C188 pod, or an 80C186EB/C188EB pod.
3. ES 1800 Emulator User’s Manual for 8018X, 80C18X and 80C18XEB Microprocessors

Optional Equipment

1. Overlay memory board (choice of 128K, 256K, 512K, 1M or 2M)
2. Symbolic debug
3. Dynamic trace board
4. Time stamp module and manual addendum
5. Logic state analyzer pod
6. SCSI high speed communications: includes SCSI board, terminator resistor network, SCSI cable and manual. PC version includes a Future Domain card.
7. ES Driver emulator control software, ES Driver User’s Manual and cable
8. Software debugger with associated manuals and cables
9. Compiler, assembler and associated manuals
10. Carrying case
11. Additional processor support: additional control board and pod
Service

If the ES 1800 unit needs to be returned for repairs, please follow these instructions:

*In the United States and Canada* Call 800-ASK-4AMC (in Washington, 206–882–2000) and ask for Customer Support. They will give you a return authorization number and shipping information.

*Outside the U.S. and Canada* Please contact your local sales office or representative for repair procedures.

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

**Limited Hardware Warranty**

Applied Microsystems Corporation warrants that all Applied Microsystems manufactured products 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 cannot be transferred.

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.

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.
This section provides an overview of the manual, an introduction to the ES 1800, and a description of all the hardware and software features and options available with the ES 1800.

How to Use This Manual

The manual is organized as follows:

Section 1: Introduction introduces Applied Microsystems Corporation’s ES 1800 emulator for the 8018x, 80C18x and 80C18xEB microprocessors. It explains possible configurations, and provides an overview of how the ES 1800 is used in debugging. ES 1800 features and options which can be used at various stages of debugging are described.

Section 2: Getting Started provides a checklist for setting up the emulator and target system, starting and testing the ES 1800, and storing customized system variables in EEPROM.

Section 3: Hardware contains all the information on the ES 1800, the control boards, the rear panel, the pod, and the serial ports, as well as information on maintenance and troubleshooting.

Section 4: Preparing for Emulation explains the steps required to use the ES 1800 to debug a problem in software or hardware. It is organized sequentially, taking you through establishing communications, setting up your target environment, running your program, breaking emulation, examining the results and making modifications to your program.

Section 5: Bringing Up Hardware shows you how to use the ES 1800 when bringing up target hardware.

Section 6: Performance Analysis, explains how to decide where to optimize your code based on time stamp information.

Section 7: Alphabetical Command Reference provides an alphabetical reference to all emulation commands.

Section 8: ES Language is a reference for the structure of the language that controls the ES 1800, with explanations of the help menus, prompts, special modes and characters, and language related error messages.

Appendix A provides explanations of the hardware error messages and serial data formats.

Appendix B describes the object module formats available for uploading and downloading files.

Appendix C describes jumpers on the 8018x, 80C18x and 80C18xEB pods which can be used to control chip selects and clock circuitry.

Appendix D lists the available application notes.

Appendix E supplies diagrams for the communication interface connections.
Introduction to the ES 1800

The ES 1800 emulation system allows you to analyze and control a target environment, consisting of hardware or software, in real time. To use the ES 1800 with your target hardware, remove the target system's microprocessor and plug in the ES 1800 emulator. Your system uses the emulator in place of the microprocessor and behaves as if the target microprocessor were there. The ES 1800 emulator also allows you to debug software without being physically connected to the target system. In this configuration, the ES 1800 uses its own real-time clock feature combined with overlay memory capabilities.

During the integration and debugging process you can read and write to the microprocessor registers or memory locations and execute programs resident in the target system or overlay memory. A program will run until you manually stop it, until it encounters a user-defined stop condition, or until it encounters an error condition. A predefined condition can be in the form of single-step operation statements or more complex statements.

Information in this manual applies to the Intel 8018x, 80C18x, and 80C18xEB microprocessors only. For more complete information on these chips, refer to the Intel hardware reference manuals: *iAPX 86/88, 186/188 User's Manual*, *186EB/188EB User's Manual*, and *16/32-Bit Embedded Processors*, published by Intel Corporation.

System Configuration

The ES 1800 can be used to help integrate and debug software and hardware. There are several configurations depending on what stage of integration you are at, and what debugging software you are using.

In each configuration, 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 ES 1800 emulator consists of a chassis which houses the control boards and an ES 1800 pod which houses the emulating microprocessor. The emulator can be controlled from a dumb terminal or a host computer, or you can use a software package on the host computer to control the emulator. These two basic environments are described below.

*ESL Control*

In this environment (refer to diagram in Figure 1-1), you use the ESL language to control the emulator. Access to the emulator is either via a dumb terminal, or via a terminal emulation program on your host computer, such as kermit, tip or cu. This environment requires an ES 1800 and a either a dumb terminal or a host computer connected to the ES 1800 terminal port.

When used with a dumb terminal, this configuration is useful for debugging target systems with software already installed or short, hand-entered routines.
When used with a host computer, you can load data from the host computer’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.

Figure 1-1: ES 1800 Controlled via ESL

Host Computer Software Package Control

The ES 1800 can also be totally controlled by a host system. This hosted software environment requires special host resident software: either the ES Driver emulator control software, or a high level language debugger.

ES Driver emulator control software provides symbolic debugging, and a convenient menu-driven interface to the ES 1800. The various high level language debuggers have been integrated with the ES 1800, providing a flexible integration environment that provides high and low level language control and debugging, and still allows direct access to the ES 1800 via ESL.
ES Driver control software and high level language debuggers are available from Applied Microsystems for most languages and host systems. For a complete list of software products that work with the 8018x, 80C18x and 80C18xEB processors, see the "Software Options" information at the end of this section.

Figure 1-2: ES 1800 Run Via Host Computer Software
System Overview

The ES 1800 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.

Emulation, or run mode, means that the microprocessor in the ES 1800 pod is running a program in the target system. Emulation stops when (1) you stop it, (2) user-defined breakpoints are enabled and occur, (3) you reset the emulator with <ctrl-z>, or (4) errors occur in the target system. During run mode, you have access to commands which let you view the target system.

When you manually stop emulation or a breakpoint is reached, you enter pause mode. In pause mode, all commands for viewing the target system are available, including commands to view the trace history of performance of the microprocessor. A command language allows you to start emulation and leave emulation when the desired combination of events are detected in the target.

ES Language

The ES 1800 uses its own command language called Emulator Satellite Language (ESL). To take full advantage of the ES 1800, you must understand the general concepts of the ESL language.

The ES 1800 operates in response to command statements composed of command mnemonics and, for some commands, arguments. The command statements form a control language, similar to high-level computer languages.

An argument to a command is an additional value entered as part of the command sequence, such as an address range or data value. Arguments can consist of single values, expressions, or lists. 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 you 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) ES 1800 hardware, (3) target software, and (4) ESL command language syntax.

Real Time Operation

Since the pod processor is identical to the target microprocessor, the target system runs in real time. No wait states are inserted by the ES 1800 emulator during run mode while accessing code, memory or I/O in the target. Note that one wait state is inserted if running above 12 MHz in standard (not high speed) overlay memory. See the discussion of the OVS command in Section 7 for wait states as they apply to overlay memory.
Steps for Using an ES 1800 Emulator

This section explains the process of using an emulator, and describes the main features and optional accessories used at each step. Detailed explanations of each step are provided in Section 4, Preparing for Emulation. Since debugging is an iterative process, these steps are meant only as a rough sequence of typical tasks, rather than a step-by-step guide.

In order to provide a complete embedded system development environment, Applied Microsystems Corporation regularly adds new software and hardware options for the ES 1800, so this list may not be comprehensive. Please contact your local sales office or representative if you are interested in extending the capabilities of the ES 1800 in ways not listed here. Phone numbers of all our offices are on the last page of this manual.

1. Establishing communications with the emulator.
   - **Features:**
     - Two convenient setup menus
     - Communications setup can be saved between sessions
     - Variety of configurations supported
     - Two serial ports
   - **Options:**
     - SCSI high speed communications

2. Setting up the target environment.
   - **Features:**
     - Built in download commands
     - Convenient commands for manipulating information in memory and I/O space
     - Convenient access to registers, including PCB registers
   - **Options:**
     - Overlay memory
     - ES Driver control software

3. Running your program from overlay or target memory.
   - **Features:**
     - Clock choices
     - Choice of run commands
     - Force special interrupt to enable safe shutdown of equipment

Introduction
4. Setting up breakpoints or tracing conditions.
   - Features:
     Event Monitor System
   - Options:
     Logic State Analyzer Pod

5. Isolating a problem by examining the trace memory, checking registers or single stepping.
   - Features:
     Trace memory
     Registers
   - Options:
     Dynamic trace
     Time Stamp Module

6. Modifying your program, either in the target or overlay memory.
   - Features:
     Built-in single line assembler
     Disassembler for trace and memory
     Single address and block memory manipulation commands

7. Using shortcuts.
   - Features:
     Repeat commands, macros, general purpose registers
     Saving setups between sessions for multiple users
   - Options:
     Symbolic debugging

8. Bringing up prototype hardware.
   - Features:
     Special functions (RAM tests, scope loops ...)

   - Options:
     Time stamp module
Steps for Using an ES1800 Emulator

Establishing Communications

How you establish communications depends on the configuration of your debugging environment: whether you are using the ES 1800 from a dumb terminal, from a host computer without a software debugger, or controlled by a software debugger on the host computer, and whether you are using serial or SCSI communications between your host computer and the ES 1800.

System setup is accomplished from two menus which contain all external communication variables and the control switches for emulation. Both setups can be saved to EEPROM and automatically loaded at power-up.

SCSI High Speed Communications (Optional)

Standard communications is via an RS-232 serial port, at speeds up to 19,200 baud. SCSI communications provides faster download speeds. Data can be transferred at rates of up to 1.5MB/second.

Setting Up the Target Environment

This step includes downloading your code to either target memory or overlay memory, verifying that the program is where you want it and making sure that everything is set up correctly to begin emulating.

The ES 1800 provides convenient commands for all these tasks, including:

- soft switches to control using the emulator with target hardware
- overlay memory, so that you can run code before hardware is available or use a combination of existing hardware and new code
- memory commands to examine and compare memory regions in overlay and target memory

Overlay Memory (Optional)

Overlay memory is ES 1800 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. Overlay memory comes on a separate board that is inserted into the ES 1800 chassis. You have a choice of a 128K, 256K, 512K, 1M or 2M Overlay Memory board. Overlay is mappable in 2KB segments. Each segment can be assigned one of four attributes: target, read/write, read-only, or illegal.
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 at speeds up to 16 MHz with the high-speed overlay memory board, and 12 MHz with the standard board. 0-15 wait states can be optionally inserted for overlay access.

**ES Driver Control Software (Optional)**

ES Driver software provides a simple, menu-driven interface to the ES 1800. ES Driver provides convenient menu access for common tasks such as configuration, uploading and downloading files and diagnostics, and allows transparent access to the full range of ESL commands. It also includes on-line help for each function, simplifying operation for new users.

**Run Program**

You can run your program using either target memory, overlay memory, or a combination of both. If you are not using a target, the ES 1800 provides an internal clock. There are a variety of run commands which you can use, depending on what information you are looking for.

**Internal Clock**

When there is no target system, you may select the internal clock feature, which places the ES 1800 in null target mode. Overlay memory can then be used to develop code as if a target system were attached.
**Steps for Using an ES1800 Emulator**

**Break Emulation**

Emulation can be halted in three ways: by you, by the Event Monitor System, or by a program error. You can enter a command to stop emulation at any time the emulator is running. You can set up the Event Monitor System to break emulation at a particular program state. If your target program commits an access or write violation in overlay memory, emulation breaks automatically. The force special interrupt command offers a way to safely stop equipment that requires a special shut-down routine.

**Setting Up Breakpoints and Conditional Tracing**

The primary way you determine where to break emulation is by setting up the Event Monitor System to detect a particular program state, and then perform a specific action.

**Event Monitor System**

The Event Monitor System is structured in three basic units:

- **Events**
  
  Events identify specific target conditions. When these conditions are encountered, actions can be performed.

- **Actions**
  
  Actions are what the emulator does when an event is detected. There are many actions that the event system can take, including standard features such as forcing a special interrupt to jump to a soft shutdown routine before stopping the target program, sophisticated trace control and breaking emulation.

- **WHEN/THEN Statements**
  
  Statements coordinate the events and actions.

You define statements that specify single or multiple events that are logical combinations of address, data, status, counter, and optional logic field states. When those events are encountered in the target system program, the ES 1800 can break emulation, trace specific sequences, count events and trigger outputs, allowing you to analyze the cause-effect relationship established by the event/action sequences defined.

There are four event groups which provide the logical structure necessary for tracking deeply nested bugs. This structure lets you debug any problem you can imagine, using a combination of events and actions.

Figure 1-3 shows the structure of the Event Monitor system.
Steps for Using an ES1800 Emulator

Figure 1-3: Event Monitor System Structure

Logic State Analyzer LSA (Optional)

The optional logic state analyzer pod (LSA) allows tracing of additional signals in the target system. It provides 16 additional input lines, giving access to signals other than the normal address, data, and control signals of the microprocessor. It also provides one trigger output line, which can be used with an oscilloscope or with another emulator for multiprocessor development.

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. This is useful when monitoring (1) buffers suspected of failure, (2) decode logic, (3) memory management circuit translations, and (4) asynchronous external events.

Isolating the Problem

Breakpoints are used to stop program execution at specific times in order to track down a hardware or software problem. 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.

Trace Memory

Trace memory contains a history of the target system program’s execution. This memory can record 2046 bus cycles and can be displayed in raw bus cycle data or disassembled into instructions. 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
Steps for Using an ES1800 Emulator

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.

The Dynamic Trace feature of the ES 1800 allows you to read trace while the target is running. Dynamic Trace is a standard feature in performance packages, and is optional otherwise. With Dynamic Trace, you can trace in target systems which require the program to remain running, such as control systems. With targets using multiple multiprocessors, dynamic trace lets you examine trace from one processor without shutting down all processors.

If you have the Dynamic Trace feature, you can view trace without stopping emulation. Without the Dynamic Trace feature, you can stop the program to read trace with either an asynchronous stop or by using the Event Monitor System to stop at the exact program state you are interested in.

Registers

The registers can be logically divided into four groups:

1. Microprocessor registers
2. General ES 1800 registers
3. Target Peripheral Control Block (PCB) registers, including registers used only in iRMX (slave) mode and registers used in non-iRMX (master) mode
4. Event Monitor System registers

These registers can be viewed and modified using the ES 1800. Each register accepts either integer values or a choice of integer, range and don’t care values. Registers can be displayed in your choice of base, and can be saved between emulation sessions.

Modifying Your Program

Once you have run your program, stopped at a particular place, and isolated the problem by looking at trace memory, the next step is to design and test possible solutions to the problem. The ES 1800 emulator lets you easily modify memory in either your target or the emulator overlay memory to make changes to your program or data.

Using Shortcuts

There are many shortcuts to shorten your setup time and reduce the number of keystrokes you must use.

Symbolic debugging

The symbolic debug option allows you to assign frequently used values to symbol names. These can either be the same symbol names you use in your program, or an easy-to-remember name to use while debugging. Symbols can be used as arguments to all commands.
Repeat commands let you repeat a command line a specified number of times or indefinitely.

Up to 10 macros can be set up for lists of commonly used commands or expressions.

You can set these registers to commonly used addresses or expressions, and then use them as arguments to commands.

Emulation setups for two users can be saved between sessions. There are six categories of information which may be saved separately: the setup menu, emulator registers, Event Monitor System WHEN/THEN statements, overlay map, software switch settings and macros.

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

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

The ES 1800 includes a set of commands specifically used for bringing up target hardware, called the diagnostic functions.

Diagnostics available in the ES 1800 emulator include RAM/ROM tests and scope loops. RAM test routines verify that RAM is operating properly. They can be run on the target or ES 1800 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 ES 1800 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.

**Time Stamp Module (Optional)**

The Time Stamp Module adds performance analysis capabilities to the ES 1800. This module is standard with performance packages, and optional otherwise. With it, you can measure the elapsed time your program spends in a module, outside of a module or between modules for up to 4 modules at once. This helps provide a picture of where your program spends the most time, so you can choose the areas which benefit most from optimization.

The Time Stamp Module also allows you to count the number of times a module or address range is accessed in order to troubleshoot iteration problems and help with optimization decisions.

You can measure the time from a hardware interrupt to a software service routine. A direct electrical connection between the interrupt line on your target processor and the Time Stamp Module lets you avoid delay in measuring interrupts.
Software Options

You have a choice of software options, including emulator control software, symbolic debuggers, high level language debuggers and a wide range of compilers and assemblers. Applied Microsystems Corporation’s goal is to provide you with a complete microprocessor development environment for both software and hardware design and debugging.

Figure 1-4: Microprocessor Development Environment

The key to this development environment is the shared information provided in the object module format. Applied Microsystems products use a variety of object module formats, including most popular standards. However, to choose a complete development environment, your compiler and assembler must produce an object module format that the assembly level or high-level language debugger and ES 1800 emulator can use.

Software options for Intel 16-bit microprocessors include:
- ES Driver Emulator Control Software
- High level language debuggers
  - GeneProbe Symbolic Debugging
  - VALIDATE/Soft-Scope Debugger
  - VALIDATE/Soft-Scope 286 Debugger
  - XDB
Software Options

- Compilers and Assemblers

ES Driver Emulator Control Software

ES Driver software provides a simple, menu-driven interface to the ES 1800, with convenient access for common tasks such as configuration, uploading and downloading files and diagnostics, and allows transparent access to the full range of ESL commands. It also includes on-line help for each function, simplifying operation for new users.


| Microprocessors supported: | Intel: 808x, 80C8x, 8018x, 80C18x, 80C18xEB, 80286  
Motorola: 68000/08, 68010, 68020  
Zilog: Z8001/Z8002 |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hosts supported:</td>
<td>PC, Sun</td>
</tr>
</tbody>
</table>
| Object module formats supported: | Extended Tekhex, Intel OMF, Intel Hex,  
Motorola S-Records, Microtec |

High Level Language Debuggers

GeneProbe Symbolic Debugging (Intel processors only)

The GeneProbe debugger provides debug support for source level problems involving CPU registers and memory or I/O ports. It has been integrated with the ES 1800 emulator, in order to provide access and control of your target. The debugger displays trace history and memory disassembly on a split screen. You can use high-level language symbols, line numbers, procedure names, code labels and variable names in place of absolute addresses.

GeneProbe executes on the IBM PC and compatibles to debug programs written in C, PL/M, FORTRAN and assembly language.

<table>
<thead>
<tr>
<th>Microprocessors supported:</th>
<th>Intel: 808x, 80C8x, 8018x, 80C18x, 80C18xEB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hosts supported:</td>
<td>PC</td>
</tr>
<tr>
<td>Object module formats supported:</td>
<td>Intel OMF</td>
</tr>
</tbody>
</table>

Introduction
VALIDATE/Soft-Scope Debugger (Intel only)

The VALIDATE/Soft-Scope debugger includes two versions: an integrated tool designed to work with the ES 1800, and a simulator version. Both versions provide high level language, assembly level and symbolic debugging. The program allows easy access to high level language data, such as structures, arrays and dynamic variables.

The trace display is available in many forms, including display of source lines only, source lines with disassembled instructions and source lines with all associated machine cycles.

A format converter called MSOMF is included with VALIDATE/Soft-Scope, so that you can use the Microsoft C compiler. Other linkers are also available which let you use a variety of other popular C compilers.

<table>
<thead>
<tr>
<th>Microprocessors supported:</th>
<th>Intel: 808x, 80C8x, 8018x, 80C18x, 80C18xEB, 80286 (real mode only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hosts supported:</td>
<td>PC</td>
</tr>
<tr>
<td>Object module formats supported:</td>
<td>OMF 86</td>
</tr>
</tbody>
</table>

VALIDATE/Soft-Scope 286 Debugger (80286 only)

The VALIDATE/Soft-Scope 286 debugger provides protected-mode support for the 80286, with all the same features as VALIDATE/Soft-Scope.

<table>
<thead>
<tr>
<th>Microprocessors supported:</th>
<th>Intel: 80286 (protect mode only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hosts supported:</td>
<td>PC</td>
</tr>
<tr>
<td>Object module formats supported:</td>
<td>OMF 286</td>
</tr>
</tbody>
</table>
Software Options

XDB Source Level Debugger

The XDB debugger provides high-level language support for C and Pascal for both Motorola and Intel microprocessors. You can step over functions, and set breakpoints on line numbers or procedures. When a breakpoint is reached, you can use the emulator's trace memory to analyze exactly what led up to the breakpoint. XDB includes a powerful assertion feature to specify conditions to be tested after execution of each high level language statement.

<table>
<thead>
<tr>
<th>Microprocessors supported:</th>
<th>Intel: 808x, 80C8x, 8018x, 80C18x 80286 (real mode only) Motorola: 68000/08, 68010, 68020</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hosts supported:</td>
<td>PC, Sun, VAX (UNIX and VMS)</td>
</tr>
<tr>
<td>Object module formats supported:</td>
<td>Intermetrics</td>
</tr>
</tbody>
</table>

Compilers and Assemblers

A wide range of compilers and assemblers are available through Applied Microsystems. Please consult a current price list, or contact your sales office or representative for information.
GETTING STARTED

Introduction

This section provides a step-by-step guide for setting up the ES 1800 and target system, starting and testing the ES 1800 and storing customized system variables in EEPROM. You should bring up the ES 1800 in stand-alone mode, using RS-232 communications to verify that it is working before trying to set it up to work with a software debugger or with SCSI communications.

For specific getting started information on using the ES 1800 controlled from a host computer via ES Driver or a software debugger, please see your appropriate software manual.

Detailed information on the hardware referred to in this section can be found in Section 3, and complete descriptions of the steps can be found in Section 4.

For a complete description of commands referenced, see Section 7.

The instructions provided in this section apply to ES 1800 emulators purchased in 1988 or later. If your ES 1800 was purchased before 1988, and has not been brought up to the current revision, there will be minor variations. Please follow the instructions provided at the time of purchase.

NOTE

If you are using the 80C186/C188 pod or the 80C186EB/C188EB pod, you may need to reconfigure the pod with several jumpers before attempting operation. Failure to properly set the jumpers results in inability to correctly run the target. See Appendix C for instructions. There are also five jumpers in the 80186/188 pod. See the 80186/188 Pod Jumpers portion of this section for more information on these jumpers.
No Target System

1. Verify that the pod is connected to the ES 1800.
2. Be sure there is nothing in contact with the probe tip.
3. Power-up the ES 1800.
4. The power-up banner should be displayed. Select the internal clock source by typing Y. If a “NO TARGET POWER” error message appears, then type <ctrl-z> to reset the emulator. The power up banner will be redisplayed. Type Y again and the emulator prompt (>) will appear.

When you power-up the ES 1800, all registers, maps, event clauses, and system variables are either cleared or set to default values. Examine the SET and ON menus (see Section 7) and configure the system to your liking. Your special setup can then be stored in EEPROM (see the SAV command in Section 7). By setting the thumbwheel switch on the MCB controller board to the proper position, your set-up can be automatically loaded on power-up, (see page 3-4), or you can load it manually with the LD command.

The ES 1800 emulator system is now running and ready to accept ESL commands.

Getting Started with ESL

ESL is extremely easy to use. The rest of this section shows you exactly which ESL commands to type as you use your ES 1800 for the first time.

If the ESL command interpreter detects an illegal statement, it beeps and places a question mark under the command line at the position the error was detected. Entering a ? following an error will cause the appropriate error message to be displayed.

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 pages 8-18 and 8-19.

Information on switch settings, configuration settings, and special functions is available without using the ? help menus.

Software Switches

Enter either ON or OFF to display the current settings and definitions of all software switches. (See ON in Section 7.)

Communications Set-up

Enter SET to display the current configuration settings and possible values. (See SET in Section 7.)

Special Diagnostic Functions

Enter SF to display a list of the available special functions (RAM/ROM tests, scope loops, etc.) (See SF in Section 7.)

For complete information on ESL syntax, see Section 8.

Getting Started
Emulator Setup

1. Refer to page 3-1 and verify that proper grounding and power requirements have been met.

2. Verify that the emulator has been configured for the correct voltage by checking the fuse on the back of the ES 1800. Pull out the fuse holder: you’ll see one functional fuse and one spare fuse. The functional fuse should be 3 amps for 115 volt, and 1.5 amp for 220 volt. Replace the fuse holder with the correct fuse in place.

3. Remove the front cover of the ES 1800 by turning the two release screws counterclockwise. The pod and LSA pod may need to be unplugged in order to do this.

4. If you are not using SCSI communications, verify that the MCB controller board is in the top slot of the ES 1800 chassis. (See page 3-1 and page 3-3 for descriptions of each board and board positions). Verify also that the shorting jumper block is installed across J1.

   If you are using SCSI communications, the SCSI board should be in the top slot, and the MCB controller board should be in the second slot. Make sure the shorting jumper block is not installed across J1.

5. Verify that the trace/break board is in the third bus slot of the ES 1800 chassis.

6. If you are using overlay memory, verify that the RAM overlay board is inserted under the trace and break board. Note that the 2MB overlay board requires a slave board.

7. Verify that the correct ES 1800 board for your target microprocessor is in the bottom slot.

8. Verify that all boards are firmly seated.

9. Set the thumbwheel switch on the MCB controller board for your particular system variables. See Table 3-1 on page 3-4 for switch settings.

   System default variables in switch position 0 are:

   - 9600 baud
   - One stop bit
   - Full duplex
   - Terminal control

   - 8-bit word length
   - No parity
   - No echo
   - XON and XOFF are recognized

   - 8th data bit set to 0 (space)

10. Verify that the three-position toggle switch on the MCB controller board is in the center position.
Target System Setup

11. Set the pod jumpers as appropriate for your target. (Skip this step if you are running the emulator with no target system.) The jumpers are located in the pod, and you can get to them by removing the four screws on the bottom of the pod to open the cover. See Appendix C to determine if you need to change any of the jumpers from their factory-configured positions.

12. Replace the front panel and attach the pod for the microprocessor you are emulating. The pod must be connected to the ES 1800 even if you are not connecting it to a target system.

13. Check that the pod cable is securely connected.

14. OPTIONAL: Connect optional accessories such as the Logic State Analyzer pod or Time Stamp module. (see Section 3 for details)

15. Connect the RS-232 cable to the TERMINAL port and to your terminal. For other setups, please see Section 4, Serial Communications.

16. Verify that the RS-232 baud rates and data requirements are set the same on both the ES 1800 and the terminal. See page 3-4 for thumbwheel switch settings.

17. 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:

   \[ \text{Figure 2-1. Null Modem Cable Wiring Diagram} \]

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.

Getting Started

Target System Setup

1. Check that the target has a 68 contact leadless chip carrier socket. An adapter, Part No. 210-00023-00, is available for plastic leaded chip carriers. For the 80C18xEB microprocessor, make sure the target has an 84-pin PLCC socket.

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 ES 1800 power.

6. Plug in the probe tip. (See Section 3 for probe tip precautions.)
**Power-Up Sequence**

**Emulating in Targets with Attached CPUs (80C18x)**

When your target CPU is soldered directly to the PCB, it is necessary to place the attached CPU in ONCE mode before emulating. The ONCE mode on the 80C18x processor causes all CPU output lines to be tristated. You can enter ONCE mode by pulling the LCS- and UCS- signals low during a reset.

To do this with the 80C18x emulator, follow these steps:

1. Power off the target and emulator.
2. Attach the 80C18x emulator pod to the target CPU with the special adaptor.
3. Jump the LCS- and UCS- lines from the target CPU to target ground.
4. Apply target power. The target-mounted CPU will come up in ONCE mode.
5. Apply emulator power and wait for the normal prompt.
6. Remove the jumpers from the target UCS- and LCS- pins.

**NOTE**

1. The procedure above assumes your target asserts a power-on reset to the 80C18x.
2. Any emulator operations which cause a target reset, such as ON CK, OFF CK, or RST will cause the target to exit from ONCE mode. If you want to perform such operations and remain in ONCE mode, set the LCS- and UCS- jumpers as described above, set the PCS softswitch to OFF, and enter PAUSE mode to perform the reset operations. **Never leave the LCS- and UCS- jumpers attached during run mode.**

A target system generated RESET during RUN mode will bring the CPU out of ONCE mode and into immediate contention with the emulator, causing unpredictable results.

**Power-Up Sequence**

**Target System Present**

1. Turn on the target system.
2. Turn on the ES 1800.
3. Reset the target system. (<ctrl-z> default)

**NOTE**

When you turn off the emulator, you should also turn off power to your target. The target VCC is fed to the pod and emulator, and can cause heat problems in the emulator if the target is left on.
**Test Run of System**

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

A system test run consists of the following 10 steps:

1. Initialize ES 1800.
2. Map overlay memory.
3. Test overlay memory.
4. Enter a program.
5. Verify a program.
6. Run the ES 1800.
7. Stop the program.
8. Display the trace buffer.
9. Set a breakpoint.
10. Initialize PCB registers.

This test requires an optional overlay memory board, but does not require a target system.

If you suspect trouble with the ES 1800 hardware, call Applied Microsystems Corporation Customer Service at 800-ASK-4AMC for assistance.

**1. Initialize The ES 1800**

You should initialize the ES 1800 the first time it is powered up, or after changing any part of the system.

Enter the following to initialize the ES 1800 for two users.

>SAV
>SET 1,1
>SAV
>SET 1,0

Save setup for user 0.
The following commands apply to user 1.
Save setup for user 1.
The following commands apply to user 0.

This will ensure that all necessary emulator firmware parameters have been loaded into the EEPROM on the MCB controller board. These parameters will be used anytime the MCB rotary switch is positioned to select EEPROM control. This EEPROM initialization should be done whenever:

1. The emulator board is changed to a different ESL revision or processor family.
2. At initial power-up of a newly purchased or rented emulator.
3. If the emulator experiences communication anomalies with the host computer.

The EEPROM initialization must be done with the MCB controller board switch in one of the factory default positions (e.g., 0 for 9600 baud, or B for 19.2 Kbaud)

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 ES 1800.

>MAP 0 to XXXX  

XXXX is the ending address (in hex) of the amount of overlay memory installed.

The following table provides a quick reference for hex values corresponding to overlay memory sizes:

<table>
<thead>
<tr>
<th>Hex Value</th>
<th>Overlay Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1FFFFF</td>
<td>128K</td>
</tr>
<tr>
<td>3FFFFF</td>
<td>256K</td>
</tr>
<tr>
<td>7FFFFF</td>
<td>512K</td>
</tr>
<tr>
<td>0FFFFFF</td>
<td>1M</td>
</tr>
<tr>
<td>1FFFFFF</td>
<td>2M</td>
</tr>
</tbody>
</table>

For example, to map 128K, enter:

>MAP 0 to 1FFFFF  

1FFFFF is 128K in hex.

3. Test RAM

Test all overlay memory installed by entering:

>SF 1,0 to XXXX  

XXXX is the ending address (in hex) of the amount of overlay memory installed.

E.g., SF1,0 to 1FFF (for 128K)

If there is a failure, repeat mapping and testing.

4. Enter Program

Enter a short program by invoking the line assembler and entering 8018x op codes. See the ASM command in Section 7 for more information.

>ASM 1000  

Enter line assembler at address 1000.

**** 8086/88/186/188 LINE ASSEMBLER VX.XLA ****

CSEG = 0000  

Set code byte segment window.

1000> NOP  

Enter NOP instruction.

1001/>  

Repeat previous NOP

1002/>  

"  

1003/>  

"  

1004/>  

"  

1005> JMP 1000H  

Enter jump instruction.

1007>X  

Exit line assembler.
**Test Run of System**

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

5. **Verify The Program**

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

\[ \text{>CS = 0} \quad \text{Set the CS register to 0.} \]
\[ \text{>IP = 1000} \quad \text{Set the IP to 1000.} \]
\[ \text{>STP;DT} \quad \text{Single step, and display trace.} \]
\[ \text{/>} \quad \text{Repeat previous command.} \]
\[ \text{/>} \quad \text{"} \]
\[ \text{/>} \quad \text{"} \]
\[ \text{/>} \quad \text{"} \]
\[ \text{/>.....} \quad \text{"} \]

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

6. **Run The ES 1800**

Enter RUN.

\[ \text{>RUN} \quad \text{Begin running the emulator.} \]
\[ \text{R>} \quad \text{The prompt will change to indicate run mode.} \]

The R> prompt should be displayed with no error messages. This indicates the ES 1800 is running in real time, executing the program.

7. **Stop The Program**

Enter STP to stop.

\[ \text{R>STP} \quad \text{The STP command from run mode stops emulation.} \]

The ES 1800 should stop running and display the CS:IP register value and Group 1. The CS:IP value should not exceed 0:1005.
8. Display The Trace Buffer

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

> DRT  
Display raw trace. The display should show sequence numbers between 0 and 20, and address values between 1000 and 1007.

> 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 ES 1800 executes 100 (hex) bus cycles, then breaks.

> DCl = OXXXX  
Set up data comparator 1 to be OXXXX.

> CTL = 100  
Set up the counter limit to be 100.

> WHEN DCl THEN CNT  
Start counting at data bus value 0XXXX.

WHEN CTL THEN BRK  
When count limit is reached, break emulation.

> RBK  
Run until a breakpoint is reached.

R>

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

If a break does not occur:

1. Set CS to 0 and IP to 1000.
2. Enter **DES 1** and verify that you have entered the WHEN/THEN statement and comparator values as shown above.
3. Type **RBK** again.

If no break occurs call Applied Microsystems Applications Engineering at 800–ASK–4AMC for assistance.

10. Initialize Peripheral Control Registers

8018x, 80C18x only

Go to page 3-14 for instructions on initializing registers for the 80C18xEB processor.

The ES 1800 emulator enables you to modify PCB register values by ESL commands (e.g., LMCS = 1FFF). If your PCB initialization code is already resident in your target, you *do not* need to set it up manually with ESL commands: you can skip this section and just run your code.

If your PCB code is not resident in the target and you need to access target memory to download your code, then you must manually set up the PCB using ESL commands, and execute at least one **STP** to load the emulator’s copy of the PCB into the pod CPU.
Test Run of System

Tables listing all PCB registers are located in Section 4.

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.
   
   >REL = <register value> Set the REL register

   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 read-chip-select switch (RCS). All the chip selects will be read and activated when RCS is turned on. If you do not set up all the chip selects, this switch should remain OFF. If you do not use on-chip chip selects, then go to step 3.
   
   >ON RCS Enables the display of the PCB chip select register values.

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

   Pause-to-run transitions will write the ES 1800 chip select PCB values into the target PCB.

   Run-to-pause transitions will read the ES 1800 chip select PCB values from the target PCB.

   >UMCS = <register value> Set UMCS register.
   >LMCS = <register value> Set LMCS register.
   >MPCS = <register value> Set MPCS register.
   >MMCS = <register value> Set MMCS register.
   >PACS = <register value> Set PACS register.

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

3. Set up the on-chip DMA peripheral. If on-chip DMA circuitry is not used, then go on to step 4.
   
   >USRIO = <register value>
   >SRIO = <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.

   >ON DME Set DMA controllers active during pause mode.
4. Set up the on-chip timer peripheral. If on-chip timer circuitry is not used, then go on to step 5.

   >TC0 = <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.

   >INT0 = <register value>
   >INT1 = <register value>
   >INT3 = <register value>
   >EOI = <register value>
   >POL = <register value>
   >POS = <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>
Test Run of System

>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         Display PCB registers.
   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.
   >MAP $FF800;DM This maps in overlay from $FF800 to $FFFFF and displays the memory map.
   >ON RDY This ensures that reads and writes to overlay memory use the ES 1800's internal ready signal.
   >ASM This invokes the single-line assembler to enter a sequence of NOP instructions.
   >CSEG = 0FFFF This sets the assembler to an absolute address of $FFFF0.
   >NOP This throw-away program initializes the on-chip peripheral circuitry.
   >NOP
   >NOP
   >NOP
   >X Exit the line 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 ES 1800's ON and OFF software switches have been properly set up.
   • A program resides at the start up location ($FFFF0).
   >AC1 = <stopping point>
   Set address comparator 1 to the end of the program. This 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 ES 1800'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 ACl, as either:

\[
ACl = $FFFF2 \quad \text{If manually initializing and using NOP program in step 7,}
\]

\[
\text{or}
\]

\[
ACl = \text{<stopping point>} \quad \text{if using your own PCB initializing program.}
\]

\[
\text{WHEN ACl THEN BRK} \quad \text{Set up WHEN/THEN statement. This allows a breakpoint when ACl is recognized during emulation.}
\]

\[
>RST;RBV \quad \text{RST sends a reset signal to the target system via the RESET OUT line. RBV sets CS:IP registers to the absolute address of $FFFF0, activates the Event Monitor System, and initiates a real-time run.}
\]
11. Initialize Peripheral Control Registers

80C18xEB only

The ES 1800 emulator enables you to modify PCB register values by ESL commands (e.g., LCST = 1FF). If your PCB initialization code is already resident in your target, you do not need to set it up manually with ESL commands: you can skip this section and just run your code.

If your PCB code is not resident in the target and you need to access target memory to download your code, then you must manually set up the PCB using ESL commands, and execute at least one STP to load the emulator’s copy of the PCB into the pod CPU.

Tables listing all PCB registers are located in section 4.

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.
   
   \[ \text{REL} = \text{<register value>} \]  
   Set the REL register

   Refer to the Intel 80C186EB/80C188EB 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.
   
   \[ \text{UCT} = \text{<register value>} \]  
   Set UCT register.
   
   \[ \text{UCP} = \text{<register value>} \]  
   Set UCP register.
   
   \[ \text{LCT} = \text{<register value>} \]  
   Set LCT register.
   
   \[ \text{LCP} = \text{<register value>} \]  
   Set LCP register.
   
   \[ \text{STR0} = \text{<register value>} \]  
   Set STR0 register.
   
   \[ \text{STP0} = \text{<register value>} \]  
   Set STP0 register.
   
   \[ \text{STR1} = \text{<register value>} \]  
   Set STR1 register.
   
   \[ \text{STP1} = \text{<register value>} \]  
   Set STP1 register.
   
   \[ \text{STR2} = \text{<register value>} \]  
   Set STR2 register.
   
   \[ \text{STP2} = \text{<register value>} \]  
   Set STP2 register.
   
   \[ \text{STR3} = \text{<register value>} \]  
   Set STR3 register.
   
   \[ \text{STP3} = \text{<register value>} \]  
   Set STP3 register.
   
   \[ \text{STR4} = \text{<register value>} \]  
   Set STR4 register.
   
   \[ \text{STP4} = \text{<register value>} \]  
   Set STP4 register.
   
   \[ \text{STR5} = \text{<register value>} \]  
   Set STR5 register.
   
   \[ \text{STP5} = \text{<register value>} \]  
   Set STP5 register.
   
   \[ \text{STR6} = \text{<register value>} \]  
   Set STR6 register.
   
   \[ \text{STP6} = \text{<register value>} \]  
   Set STP6 register.
   
   \[ \text{STR7} = \text{<register value>} \]  
   Set STR7 register.
   
   \[ \text{STP7} = \text{<register value>} \]  
   Set STP7 register.
Refer to the Intel 80C186EB/80C188EB User's Manual for the proper way to set up the registers.

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

> TC0 = <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 80C186EB/80C188EB 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 TEO
> ON TE1
> ON TE2

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

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

> INT0 = <register value>
> INT1 = <register value>
> INT2 = <register value>
> INT3 = <register value>
> INT4 = <register value>
> BOI = <register value>
> POL = <register value>
> POS = <register value>
> MSR = <register value>
> PLM = <register value>
> ISV = <register value>
> IRQ = <register value>
Test Run of System

>\text{IST} = \text{<register value>}
>\text{TCR} = \text{<register value>}
>\text{SCR} = \text{<register value>}

Refer to the Intel 80C186EB/80C188EB User's Manual for the proper setup.

5. Set up the refresh registers. Refer to the Intel 80C186EB/80C188EB User's Manual for the proper setup.

>\text{RFBS} = \text{<register value>} \quad \text{Set the refresh base register}
>\text{RFTM} = \text{<register value>} \quad \text{Set the refresh time register}
>\text{RFCH} = \text{<register value>} \quad \text{Set the refresh control register}
>\text{RFAD} = \text{<register value>} \quad \text{Set the refresh address register}

6. Set up the \text{I/O} port and serial port registers. Refer to the Intel 80C186EB/80C188EB User's Manual for the proper setup.

>\text{PDR1} = \text{<register value>}
>\text{PPN1} = \text{<register value>}
>\text{PCN1} = \text{<register value>}
>\text{PLT1} = \text{<register value>}
>\text{PDR2} = \text{<register value>}
>\text{PPN2} = \text{<register value>}
>\text{PCN2} = \text{<register value>}
>\text{PLT2} = \text{<register value>}
>\text{SBD0} = \text{<register value>}
>\text{SCT0} = \text{<register value>}
>\text{SCN0} = \text{<register value>}
>\text{SRB0} = \text{<register value>}
>\text{SBD1} = \text{<register value>}
>\text{SCT1} = \text{<register value>}
>\text{SCN1} = \text{<register value>}
>\text{SRB1} = \text{<register value>}

7. Set up the power management register. Refer to the Intel 80C186EB/80C188EB User's Manual for the proper setup.

>\text{PMC} = \text{<register value>}

\textit{Getting Started}
8. Display the status of the PCB registers.

   >PCB Display PCB registers.

   The screen displays the current contents of the PCB registers.

9. 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 10.

   >MAP $FF800;DM This maps in overlay from $FF800 to $FFFFF and displays the memory map.

   >ON RDY This ensures that reads and writes to overlay memory use the ES 1800’s internal ready signal.

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

   >CSEG = OFFFF This sets the assembler to an absolute address of $FFFF0.

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

   >NOP
   >NOP
   >NOP

   >X Exit the line assembler.

10. 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 ES 1800’s ON and OFF software switches have been properly set up.

   • A program resides at the start up location ($FFFF0).

   >AC1 = <stopping point> Set address comparator 1 to the end of the program. This should follow the initialize section.
Test Run of System

The on-chip peripherals are activated by either a read from, or write to appropriate registers. The setting of the ES 1800’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:

\[
\text{AC1} = \text{FFFF2} \quad \text{If manually initializing and using NOP program in step 6,}
\]

or

\[
\text{AC1} = \text{<stopping point>} \quad \text{if using your own PCB initializing program.}
\]

\[
\text{WHEN AC1 THEN BRK} \quad \text{Set up WHEN/THEN statement. This allows a breakpoint when AC1 is recognized during emulation.}
\]

\[
\text{>RST;RBV} \quad \text{RST sends a reset signal to the target system via the RESET OUT line. RBV sets CS:IP registers to the absolute address of $FFFF0, activates the Event Monitor System, and initiates a real-time run.}
\]
This section describes the emulator chassis, control boards, pod, optional hardware (Time Stamp Module and Logic State Analyzer pod), ports, maintenance, troubleshooting and emulator specifications.

## Emulator Chassis

The ES 1800 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 Figure 3–1. The ES 1800 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 ES 1800 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 ES 1800 to overheat.

* Always turn off target power when the emulator is off, and vice versa. Damage can occur to the pod by leaving target power on when the emulator is turned off.

## System Grounds

The ES 1800 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 ES 1800 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 ES 1800.

**WARNING**

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

## Emulator Control Boards

Removing the front panel of the ES 1800 chassis exposes the chassis card cage as shown in Figure 3-1. Follow these steps to open the ES 1800 chassis:

1. Turn off the emulator.
2. Disconnect it from the power source.
3. Remove the front panel. Depending on the version of your emulator, you have one of two types of front panel.

If you have the molded-plastic front panel, the release tabs are located at the bottom left and right sides. Press the left release tab, while pulling the left side of the panel slightly outward. Then press the right release tab, and pull outward until the bottom of the panel is completely free. Slide the panel down to remove it. Finally, disconnect the SCSI cable (if applicable).

If you have the metal front panel, disconnect the cables from the front of the emulator (if applicable). Then loosen the thumbscrews in the upper corners of the front panel and remove it.

Verify that all boards are seated properly before turning on power to the emulator.

**SCSI Board**

The SCSI board is required in order to use SCSI communications between the ES 1800 and host computer. If present, it should be in the top slot in the chassis. The SCSI port is discussed in detail under Ports, later in this section and in the *ES Driver User's Manual*.

**NOTE**

When you set up the emulator for SCSI communications, the shorting jumper block J1 on the MCB controller board must be removed or set in the storage position.

**MCB Controller Board**

The MCB controller board holds the controlling 6809 CPU for the ES 1800, the EEPROM, two serial ports, RAM, the memory management logic and optional symbolic memory.

The 16-position thumbwheel switch on this board determines the system variables and serial line baud rates for autoloading on power-up. Refer to Table 3-1 on page 3-4 for switch position setup. Switch position 0 automatically loads default system variables.

The three-position toggle switch must be in the center position. If the toggle switch is in either of the other two positions, the ES 1800 will not work properly.
If there is no SCSI board, this board should be in the top slot in the chassis. When the MCB controller board is in the top slot, make sure the J1 shorting jumper block is in the *active* position.

**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 is optional and can hold 128K, 256K, 512K, 1M or 2M of memory. 2M of memory requires a slave board.

**Emulation Board**

The emulation board depends on the target microprocessor you are using. It contains the target processor specific logic.

---

*Figure 3-1: Control Boards*
Table 3-1: MCB Controller Board
Thumbwheel Switch Settings

<table>
<thead>
<tr>
<th>POSITION</th>
<th>PARAMETERS</th>
<th>BAUD RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Factory Default*</td>
<td>9,600</td>
</tr>
<tr>
<td>1</td>
<td>User &quot;0&quot; defined</td>
<td>User defined Terminal control</td>
</tr>
<tr>
<td>2</td>
<td>User &quot;1&quot; defined</td>
<td>User defined Terminal control</td>
</tr>
<tr>
<td>3</td>
<td>User &quot;0&quot; defined</td>
<td>User defined Computer control</td>
</tr>
<tr>
<td>4</td>
<td>User &quot;1&quot; defined</td>
<td>User defined Computer control</td>
</tr>
<tr>
<td>5</td>
<td>Factory Default*</td>
<td>110</td>
</tr>
<tr>
<td>6</td>
<td>Factory Default*</td>
<td>300</td>
</tr>
<tr>
<td>7</td>
<td>Factory Default*</td>
<td>1,200</td>
</tr>
<tr>
<td>8</td>
<td>Factory Default*</td>
<td>2,400</td>
</tr>
<tr>
<td>9</td>
<td>Factory Default*</td>
<td>4,800</td>
</tr>
<tr>
<td>A</td>
<td>Factory Default*</td>
<td>7,200</td>
</tr>
<tr>
<td>B</td>
<td>Factory Default*</td>
<td>19,200</td>
</tr>
<tr>
<td>C,D,E,F</td>
<td>Reserved for factory use</td>
<td></td>
</tr>
</tbody>
</table>

*Factory Default Parameters
8-bit word length one stop bit
no parity full duplex
Terminal control XON and XOFF are recognized
no echo baud rate the same for both terminals
8th data bit set to 0 or a space
ES 1800 Chassis Front Panel

The front panel of the ES 1800 is shown in Figure 3-1.

*Release screws* Unscrewing these two screws makes it possible to remove the front panel of the ES 1800 to get access to the control boards. If you have a molded plastic front panel, there are no release screws. See page 3-1 for instructions on removing the front panel.

*LSA port* The LSA port is used for either the Logic State Analyzer pod or the Time Stamp module.

*SCSI port* The SCSI port is used only if you are using SCSI communications.

*Pod connection* The pod is attached here.

ES 1800 Chassis Rear Panel

The rear panel of the ES 1800 is shown in Figure 3-2.

*Serial Ports* The two serial ports are RS-232C ports labeled **TERMINAL** and **COMPUTER**. Serial ports are discussed in detail under "Ports" later in this section.

*Trigger Output* The ES 1800 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. See Figure 3-8 for timing information on the trigger output, and refer to Section 4 for information on Event Monitor System actions.

The trigger can be used for such things as:

1. Synchronizing an oscilloscope to the execution of an I/O routine.
2. Measuring the duration of a routine by asserting the trigger for its duration and using a timer-counter.
3. Cross-coupling two or more ES 1800s so that an event in one can control events in the others.
Before powering up, two items should be checked:

1. Proper grounding of power cable (see page 3-1).
2. Proper power-up sequence of ES 1800, target system, and/or peripheral equipment. (See Power-Up Sequence, page 2-4.)

A 3 amp slo-blo fuse for 110V operation or a 1.5 amp slo-blo fuse for 220V operations. Remove the fuse by turning the fuse holder counterclockwise.
Pod

The pod is the link between the ES 1800 emulator and the target system. A 40-inch ribbon cable connects the pod to the ES 1800 board. An 11-inch ribbon cable ends in a probe tip that is normally inserted into the microprocessor socket in the target system.

The proper pod is determined by the microprocessor being emulated. Three pods are available from Applied Microsystems Corporation: one for the 80186 and 80188, one for the 80C186 and 80C188, and one for the 80C186/88 EB.

The 80186 and 80188 microprocessors can be emulated with the same pod, but with different microprocessors in the pod. The pod should have been shipped from the factory with the correct microprocessor installed.

- 80186
- 80188

The 80C186 and 80C188 can be emulated with the same pod, but with different microprocessors in the pod.

- 80C186
- 80C188

The 80C186EB and 80C188 EB can be emulated with the same pod and same microprocessor in the pod. A jumper setting determines which microprocessor is being emulated.

To install the probe tip into your target system:

- 80186/188, 80C186/C188
  
  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.

- 80C186EB/C188EB
  
  Remove the microprocessor chip from your target and plug the probe tip into the PLCC socket. Always check that pin 1 is aligned correctly.

For the 8018x/80C18x, check that the target has a 68 contact leadless chip carrier (LCC) socket. An adapter, Part No. 210-00023-00, is available for plastic leaded chip carriers (PLCC). For the 80C18xEB, check that the target has an 84-pin PLCC socket.
Figure 3-3: 8018x, 80C18x and 80C18xEB Pod Assemblies
Saving Desk Space

To save limited desk or table space, the 80C186/C88 pods can be supported from walls, an overhead hook, or other non-horizontal surfaces either by velcro tape or by a hanging strap.

Velcro Tape

To support the pod using velcro tape, you must first attach the 5" long bracket to the bottom sheet metal of the pod (you may need to bend the bracket slightly). Figure 3-4 shows bracket placement. When the bracket is in place, simply peel off the adhesive backing on the velcro tape strip and firmly press the tape onto the bracket as shown in Figure 3-4. You can now attach the 80C186/C188 pod to any surface that adheres to velcro, such as many types of office partitions.

Figure 3-4: Velcro Tape Support

Hanging Strap

The hanging strap can be threaded through either set of eyelets on the bottom sheet metal of the pod. The 5" long bracket is not needed when using the hanging strap. Figure 3-5 shows both of these configurations. After threading the strap through the eyelet, bend the strap back on itself and fasten it with the enclosed fasteners. Make sure the fasteners on both sides are firmly closed before hanging the pod from the strap.
Figure 3-5: Hanging Strap Support

Pod

Bottom view of pod

Eyelets

Fastener

Hanging strap
Time Stamp Module

An optional feature, the Time Stamp Module, adds performance analysis capabilities to the ES 1800. This module allows you to measure the elapsed time your program spends in a module, outside of a module or between modules for up to 4 modules at once. This can provide a picture of where your program spends the most time, so you can choose the areas which benefit most from optimization.

The Time Stamp module also allows you to count the number of times a module or address range is accessed in order to troubleshoot iteration problems and help with optimization decisions. The time from a hardware interrupt to a software service routine can be measured. A direct electrical connection between the interrupt line on your target processor and the Time Stamp Module lets you avoid delay in processing interrupts.

The time stamp module connects directly above the ES 1800 pod to the connector labelled LSA Pod. You cannot use both the LSA pod and time stamp module at the same time.

For complete information on setting up and using your Time Stamp Module, see Section 6.

Figure 3-6: Time Stamp Module
Logic State Analyzer (LSA)

An optional feature, the logic state analyzer (LSA) pod, connects directly above the ES 1800 pod. The LSA 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 ES 1800 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.

To use the pod, you plug it in to the port on the front of the ES 1800 labeled "LSA." The 16 input clips can be attached anywhere in your target. Then you use the LSA comparators in the Event Monitor System to monitor the input pulses from the Logic State Analyzer.

Figure 3-7: Logic State Analyzer Pod

LSA Timing Strobe

The ES 1800 uses a bus request signal, shown in Figure 3-8, to generate a trigger which is sent to the LSA pod and to the BNC connector on the rear panel. The trigger is a low-going-high signal for approximately one bus cycle, and is generated approximately 70 ns after an event.
Figure 3-8: LSA and Trigger Timing
Ports

There are two serial ports and one optional SCSI port on the ES 1800. The SCSI option requires installation at both the host computer and emulator sides. On the host side, the installation depends on which host computer you are using. For PCs and compatibles, a SCSI board such as the Future Domain TMC 800 line must be installed in order to add a SCSI port to the PC. For Sun workstations, a new SCSI device driver must be installed. The SCSI option is not available on VAX and Apollo computers. On the emulator side, the SCSI board must be installed in the ES 1800. A special SCSI cable is also provided with the option to connect the ES 1800 to the host computer.

For information on the SCSI port, see either your *SCSI Addendum for ES 1800 Emulators* or your *ES Driver/Sun* user’s manual.

Serial Ports

Both the terminal port and the computer port end in standard RS-232C 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 in Section 7 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. For complete information, see *Serial Communications* in Section 5.

**Upload/Download**

The ES 1800 accepts commands to begin uploading/downloading from either port. However, the ES 1800 uploads/downloads hex format data files only through the computer port.

Serial Port Pin Configurations

The pin configuration of your equipment (terminal, PC or host) may not match that of the ES 1800. It is important to be familiar with the pin configurations of all peripheral equipment you intend to use with the ES 1800 emulator.

The ES 1800 emulator is configured as *Data Terminal Equipment* (DTE). Before powering up, make sure the ES 1800 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 ES 1800 use these pins.
Both ES 1800 serial ports use the same pin assignment. All pin assignments and voltage levels conform to Electronics Industries Association (EIA) RS-232C standards. The following chart lists the signals present on each pin.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Protective Ground</td>
<td>Connected in the ES 1800 emulator to logic ground.</td>
</tr>
<tr>
<td>2</td>
<td>Serial Data Out</td>
<td>This signal is driven to nominal 12 voltage levels by an RS-232C compatible driver.</td>
</tr>
<tr>
<td>3</td>
<td>Serial Data In</td>
<td>Data is accepted on this pin if the voltage levels (12V) are as specified by RS-232C specifications.</td>
</tr>
<tr>
<td>4</td>
<td>Request to Send (Output)</td>
<td>This signal is driven to nominal 12V levels by an RS-232C compatible driver. It signals other equipment that the ES 1800 emulator is ready to accept data at this port.</td>
</tr>
<tr>
<td>5</td>
<td>Clear to Send (Input)</td>
<td>An input signal to the ES 1800 emulator indicates another piece of equipment in the system is ready to accept data. This signal is terminated so the ES 1800 emulator operates with the signal disconnected.</td>
</tr>
<tr>
<td>6</td>
<td>Not Used</td>
<td>Connected in the ES 1800 emulator to the system logic ground.</td>
</tr>
<tr>
<td>7</td>
<td>Signal Ground</td>
<td>These pins are not used by the ES 1800 emulator but may be required by your peripheral hardware.</td>
</tr>
<tr>
<td>8-25</td>
<td>Not Used</td>
<td></td>
</tr>
</tbody>
</table>

**Data Requirements**

The data requirements are set in the SET menu. See Section 7 for details on using the SET menu.

**Stop Bits**

The ES 1800 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. (Section 7).

**Parity**

The ES 1800 sends and checks parity according to system SET parameter #12 for the terminal port and #22 for the computer port.
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 ES 1800 is ready to receive data, it asserts the Request To Send line (pin #4). When a receive buffer is nearly full, the ES 1800 deasserts the Request To Send line.

When the ES 1800 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 ES 1800 uses normal flow control codes to control software handshaking. The default values are XON (DC1) and XOFF (DC3).

The ES 1800 serial I/O 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 ES 1800.

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 ES 1800 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, probe tip and cleaning the fan filter.

Cables

The 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. By swapping the cables in question with a known good set of cables, you can easily isolate the faulty cable.

Probe tip

The probe tip for the 8018x and 80C18x consists of a ceramic leadless chip, four ribbon cables and an adapter board. The probe tip for the 80C18xEB consists of a 114-pin grid array with an 84-pin PLCC adapter. The adapter board is inside the pod case. When the ES 1800 is not in use, the protective cover should be installed over the probe tip 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. Check that the probe tip is clean and well seated.

Cleaning the Fan Filter

NOTE

Units with part numbers in the range 750-00500-xx do not have fan filters.

If you have an emulator with a fan filter, it 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 release screws.)
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.
Figure 3-9: ES 1800 Fan Mounting
5. Tilt the fan towards the boards in the chassis.

Figure 3-10: 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
- Short cable set
- Long cable set
Troubleshooting

Check that the cables are installed properly, that the probe tip is plugged into a compatible target system, with power applied to both the target system and the ES 1800 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` and record the serial number located on the back of the chassis. You will be asked for the revision number and serial number when you call.

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

<table>
<thead>
<tr>
<th>SYMPTOM</th>
<th>POSSIBLE CAUSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target system runs erratically</td>
<td>1. Faulty cables.</td>
</tr>
<tr>
<td></td>
<td>2. Broken pin on adapter.</td>
</tr>
<tr>
<td></td>
<td>3. ES 1800 emulator and target system not compatible.</td>
</tr>
<tr>
<td></td>
<td>4. LDV not executed before RUN (vector not loaded).</td>
</tr>
<tr>
<td></td>
<td>5. The probe tip is not making good contact.</td>
</tr>
<tr>
<td>Emulator will not communicate over RS-232</td>
<td>1. Baud rate set incorrectly.</td>
</tr>
<tr>
<td></td>
<td>2. Target system requires &quot;null&quot; modem cable (pin 2 and pin 3 of RS-232 connector reversed).</td>
</tr>
<tr>
<td></td>
<td>3. For terminal operation, thumbwheel switch located on the top card is not in the &quot;0&quot; position or the cable is not properly attached to the terminal port in the back of the ES 1800.</td>
</tr>
<tr>
<td></td>
<td>4. Cable not going to correct port of the terminal or PC.</td>
</tr>
<tr>
<td></td>
<td>5. Toggle switch located on the second card from the top in the ES 1800 not in the middle position.</td>
</tr>
<tr>
<td></td>
<td>6. Power is not on, or a fuse has blown.</td>
</tr>
<tr>
<td></td>
<td>7. Control boards not seated properly.</td>
</tr>
</tbody>
</table>
ES 1800 Emulator Specifications

Input Power

<table>
<thead>
<tr>
<th></th>
<th>Standard</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>90 to 130 VAC, 47 to 60 Hz consumption less than 130W</td>
<td>180 to 260 VAC, 47 to 50 Hz consumption less than 130W</td>
</tr>
</tbody>
</table>

Environmental

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating Temperature</td>
<td>0°C to 40°C (32°F to 104°F)</td>
</tr>
<tr>
<td>Storage Temperature</td>
<td>-40°C to 70°C (-40°F to 158°F)</td>
</tr>
<tr>
<td>Humidity</td>
<td>5% to 95% relative humidity, noncondensing</td>
</tr>
</tbody>
</table>

Physical

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mainframe</td>
<td>13.2 cm x 43.18 cm x 34.29 cm.</td>
</tr>
<tr>
<td></td>
<td>(6.2 in. x 17 in. x 13.5 in.)</td>
</tr>
<tr>
<td>80186/188 Pod</td>
<td>22.6 cm x 12.9 cm x 4.1 cm.</td>
</tr>
<tr>
<td></td>
<td>(8.9 in. x 5.1 in. x 1.6 in.)</td>
</tr>
<tr>
<td>80C186/C188 Pod</td>
<td>21.6 cm x 27.9 cm x 2.2 cm.</td>
</tr>
<tr>
<td></td>
<td>(8.5 in. x 11.0 in. x 0.85 in.)</td>
</tr>
<tr>
<td>80C186EB/C188EB Pod</td>
<td>21.6 cm x 29.7 cm x 5.1 cm.</td>
</tr>
<tr>
<td></td>
<td>(8.5 in. x 11.7 in. x 2 in.)</td>
</tr>
<tr>
<td>Target System Connection</td>
<td>1.5 m (total length including pod)</td>
</tr>
<tr>
<td>LSA Pod</td>
<td>12.4 cm x 7.9 cm x 2.3 cm.</td>
</tr>
<tr>
<td></td>
<td>(4.9 in. x 3.1 in. x .9 in.)</td>
</tr>
</tbody>
</table>
This section guides you through the steps required to use the ES 1800 emulator to debug hardware and software problems. The general steps are:

- establishing communication with the emulator
- setting up your target environment by mapping overlay memory, checking registers, setting up soft switches, and downloading program
- running your program
- breaking emulation
- isolating a problem by examining the trace memory, checking registers or single stepping
- modifying your program, either in the target or overlay memory
- using shortcuts, such as symbols, repeat commands, macros, saving setup between sessions, maintaining different setups for multiple users and clear commands

Each step includes a summary of the commands used during that step and examples of using groups of commands to do useful tasks.

Section 7 provides a detailed alphabetical reference for all the commands mentioned in this section.
Terms

Before using this section, you should be familiar with the following terms:

**target**
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.

**run mode**
Indicates that emulation has begun. The microprocessor in the pod is running a program in the target. The run mode prompt is R>.

**pause mode**
Indicates that emulation is not taking place. The pause mode prompt is >. Many commands can only be used in pause mode.

**transparent mode**
Transparent mode is used to communicate with a host computer or any other peripheral you attach to a serial port on the ES 1800. In transparent mode, the two ES 1800 serial ports (TERMINAL and COMPUTER) are connected.

**peek/poke**
Peeks and pokes are single bus cycle reads and writes to target or overlay memory. When a peek/poke is requested during run mode, we break emulation (you don’t see this) and do a single target bus cycle, then go back into emulation.
Establish Communication with the Emulator

How you establish communication depends on the configuration of your debugging environment and whether you are using serial or SCSI communication between your host computer and the ES 1800.

This section describes establishing communication when you are using the emulator with a dumb terminal or with a terminal and a host computer. For information on establishing communication from ES Driver or one of the VALIDATE software debuggers, please use the appropriate software manual.

Note that Section 2 of this manual provides quick instructions to get you started, whereas this section provides a more complete explanation of the process.

### Commands Used to Establish and Verify Communication

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCT</td>
<td>Control emulator from COMPUTER port</td>
</tr>
<tr>
<td>SET</td>
<td>Set up port parameters</td>
</tr>
<tr>
<td>TCT</td>
<td>Control emulator from TERMINAL port</td>
</tr>
<tr>
<td>TRA</td>
<td>Enter transparent mode</td>
</tr>
</tbody>
</table>

### Serial Communication

The ES 1800 can communicate through both DB-25 connectors on the chassis rear panel using standard RS-232C serial protocol. The ports can be independently configured for baud rate, data length, and number of stop bits.

### From a Terminal or Host Computer

When using a dumb terminal to control the ES 1800, you connect a terminal to the TERMINAL port on the back of the ES 1800 using an RS-232 cable. When the ES 1800 is shipped, it is configured for TERMINAL port control.

One common development configuration is with a terminal connected to the TERMINAL port of the ES 1800 and a host development system connected to the COMPUTER port. The ES 1800 provides a transparent mode that essentially connects your terminal to the computer. The ES 1800 also has a special download command to load modules from the host system and commands to upload data and symbols to the host system.

In configurations where the ES 1800 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 ES 1800 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 ES 1800 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 or systems using networks 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.

The XON/XOFF problem described in the above paragraph can also happen in the reverse direction. If the ES 1800 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 ES 1800 will quickly send an XOFF to the host while continuing to send normal upload characters. The host system will then probably send an XOFF to the ES 1800 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 department at 800-ASK-4AMC for help.

XON and XOFF characters can be used to control either output port on the ES 1800. These characters can be redefined using the SET command.

Communication with the Host Computer

While in transparent mode, the ES 1800 passes characters between the computer and TERMINAL ports. There is a user definable two-character escape sequence to exit transparent mode, set with the SET command (<esc><esc> default). If the first character of the escape sequence arrives at either port, the ES 1800 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.

While in transparent mode, the only characters that are meaningful to the ES 1800 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. You should define the reset character (<ctrl-z> default) using the SET command to be a character that will not normally be used by the host system or an editor.
Establish Communication: Setup Command

Controlled by Host Computer

In this configuration, a software package on the host computer actually controls the ES 1800. Please see your ES Driver or VALIDATE debugger manual for information on setting up communication.

Setup Commands

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 set the serial communication parameters and save them to EEPROM without affecting the parameters currently in use.

The three categories of parameters are summarized in the following table:

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>User number, reset character, XON/XOFF characters, LSA display</td>
</tr>
<tr>
<td>Terminal port</td>
<td>Baud rate, stop bits, parity, screen display length, transparent mode escape sequence</td>
</tr>
<tr>
<td>Computer port</td>
<td>Baud rate, stop bits, parity, transparent mode escape sequence, command terminator sequence, record length, download/upload data format, acknowledge character</td>
</tr>
</tbody>
</table>

Port Dependent Commands

The ‘controlling’ port is determined at power-up by the setting of the thumbwheel switch on the controller board (see Section 3). 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. All commands except UPL, DNL and UPS respond in the same manner if entered from either the computer port or the TERMINAL port.

Transparent Mode

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.

SCSI Communication

For information on the SCSI port, see either your SCSI Addendum for ES 1800 Emulators or your ES Driver/Sun user’s manual.

Preparing for Emulation 4-5
Set Up Target Environment

After you have established communication with the emulator, you must download your code to either target or overlay memory. Once the code is downloaded, you will want to verify that the program is where you want it, and that everything is set up correctly to begin emulating.

The ES 1800 provides convenient commands for all these tasks, including:

- overlay memory commands, so that you can run code before hardware is available or use a combination of existing hardware and new code
- download commands to load code into target or overlay memory
- memory commands to examine and compare memory regions in overlay and target memory
- register commands to examine and modify registers
- soft switches to control using the emulator with target hardware

**Commands Used to Set Up Target Environment**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overlay Memory Commands</strong></td>
<td></td>
</tr>
<tr>
<td>CLM</td>
<td>Clear memory map</td>
</tr>
<tr>
<td>DM</td>
<td>Display memory map</td>
</tr>
<tr>
<td>LOV</td>
<td>Load overlay memory from target</td>
</tr>
<tr>
<td>MAP</td>
<td>Set memory map</td>
</tr>
<tr>
<td>OVE</td>
<td>Enable overlay memory</td>
</tr>
<tr>
<td>OVS</td>
<td>Overlay memory speed</td>
</tr>
<tr>
<td>VFO</td>
<td>Verify overlay memory</td>
</tr>
<tr>
<td><strong>Clock Commands</strong></td>
<td></td>
</tr>
<tr>
<td>CK</td>
<td>Choose target clock</td>
</tr>
<tr>
<td>CLK</td>
<td>Read target clock frequency</td>
</tr>
<tr>
<td><strong>Download Commands</strong></td>
<td></td>
</tr>
<tr>
<td>DNL</td>
<td>Download file to target or overlay</td>
</tr>
<tr>
<td>SET</td>
<td>Set up communication parameters</td>
</tr>
<tr>
<td>TRA</td>
<td>Enter transparent mode</td>
</tr>
<tr>
<td>VFY</td>
<td>Verify serial download data</td>
</tr>
</tbody>
</table>
Map 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 128K to 2M and 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.

You can always modify overlay memory mapped as read-only. 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.
Overlay memory is mapped with the MAP command, and the map is displayed with the DM command. Once you have memory mapped, you can move a program from target memory to overlay with the LOV command. The VFO command lets you compare a range of memory in your target to the same range in the 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 switch (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 at speeds up to 16 MHz with the high-speed overlay memory board, and 12.5MHz with the standard board. To operate at speeds greater than 12.5MHz using the standard board, you will need to add wait states using the OVS command. The OVS command requires the RDY switch to be set: this switch selects an internally generated ready signal to complete memory accesses. OVS is automatically set to 1 if CLK (clock frequency) is greater than 12.5 MHz and you are using a standard overlay memory board. If you are using a fast overlay memory board, OVS is set to zero.

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.

The following examples show using overlay memory to patch a program.

```
>CLM
>MAP 1000 to 7FFF:RO
>LOV 1000 to 7FFF
>ASM 2000
(Assembler commands)
>RNV
>STP;MAP 1000 TO 7FFF:TGT;RUN
>STP;MAP 1000 to 7FFF:RO;RNV
```

Clear any previous mapping.
Map ROM over existing target program.
Copy target program into overlay memory.
Use line assembler to make a patch.
Run patched version.
Stop, remove map, run normal version.
Stop, restore map, run patched version.
Set Up Target Environment: Download Files

Download Files

You can enter the download command from either the TERMINAL port or COMPUTER port, but download data is always received by the emulator through the computer port. The data will be written to the target system memory, or to overlay memory if it is mapped.

Before downloading, you should verify the following:

- Overlay is mapped to the appropriate address range.
- The start address of the file is the address to which you expect to download (see TRA in Section 7).
- The data format of the host system matches that used by the ES 1800 emulator (see SET parameter #26 and TRA in Section 7).

Download from Terminal Port

When you type DNL from the TERMINAL port, the ES 1800 automatically enters transparent mode. Because the ES 1800 expects data records to arrive at the COMPUTER port, transparent mode allows you to send commands to the host system using the TERMINAL port.

When you are ready to download a file, enter the proper command for your system which causes the host system to display a file to the terminal (commands such as copy, type, or cat). Terminate the command with the transparent mode escape sequence (<esc><esc> is the default), not a <return>. The command terminator sequence, which is user definable, is sent to the host system (<return>,null,null is the default).

The ES 1800 is now ready to read the data records the host system will be sending. Data records are displayed as they are received by the ES 1800. Each data byte is verified with a 'read after write' cycle. If an error is detected, the download is aborted. 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 be written to memory. 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.

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.

Download from Computer Port

If the download command is entered from the COMPUTER port, the process is different. In this case, the ES 1800 does not enter transparent mode. The DNL command can be immediately followed by data records.

Preparing for Emulation
Set Up Target Environment: Download Files

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

Each data record is acknowledged with an ACK (6) character if its checksum is correct and correctly written into target memory (verified with read-after-write cycles). The EOF record is also acknowledged if valid. If an error occurs during a download, the first character sent back to the host will be the BEL (7) code. Programs written on the host system can use these two characters to handshake the data records in an automatic download routine.

There are some differences between COMPUTER port control and TERMINAL port control during the downloading process. Under COMPUTER port control:

1. All good records are acknowledged with an ACK $6.
2. All error messages from bad records are received on the COMPUTER port; therefore the host program that is controlling the ES 1800 will need to be able to interpret error messages.
3. Records are not echoed.

Return Control to ES 1800

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 ES 1800, 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 ES 1800 reset is executed (default is <ctrl-z>).
3. An error is detected.

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 ES 1800.

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.
Set Up Target Environment: Check Registers

<table>
<thead>
<tr>
<th>Problem</th>
<th>What to Check</th>
</tr>
</thead>
</table>
| Emulator does not return a prompt            | 1. Serial data format - SET menu.  
2. No end of file (EOF) record.  
3. You entered a <return> instead of the transparent mode escape sequence after entering the host copy command. |
| Read-after-write verify error                 | 1. Target hardware problem.  
2. Overlay memory not mapped in download range. Address is indicated by message. |
| Checksum error                               | 1. Improperly formatted record sent by host.  
2. Noisy serial data lines.  
3. Host computer is not responding to XON/XOFF protocol. |
| Display of data does not begin after entering transparent mode escape sequence | 1. Host not responding to user defined command terminator sequence - see SET menu. |

If the ES 1800 does not return a prompt, you will need to reset the system (default is <ctrl-z>) in order to enter any other ES 1800 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.

Symbolic Download

The download command accepts symbolic definition records as well as data records when the symbolic debug option is used and the ES 1800 download format variable is set to 5 (Extended Tekhex). (See SET parameter #26).

Symbols can be verified with memory using the VFY command. VFY will verify that the symbols in the ES memory match the symbols in the download file.

Check Registers

Before going into run mode, you will want to be sure that the code segment and instruction pointer (CS:IP) contain the correct values. You may also want to set a valid stack pointer, initialize the CPU status register (FLX) or some of the PCB registers.
You can either set registers by hand or use the **LDV** command to load them with their power-up values.

This section includes information on using the registers and a complete list of all the registers in the ES 1800.

The registers can be logically divided into four groups:

1. microprocessor registers
2. general ES 1800 registers
3. Peripheral Control Block (PCB) registers, those used only in iRMX mode and those used in non-iRMX mode
4. Event Monitor System registers

Each ES 1800 or Event Monitor System register accepts one or two of three value types: integer values, range values or don’t care values. The value of any register can be displayed by entering its name on the command line. Register values can be modified using the syntax `register = value`.

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. Display bases are often changed for registers such as the Event Monitor LSA comparators, which you might like to see in binary, and the count limit (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.

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.

**Registers In Run Mode**

Setting and displaying the microprocessor registers during run mode can lead to unexpected results because the ES 1800 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 ES 1800’s image of the registers. The ES 1800 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.
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 ES 1800 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:

```plaintext
>MCWO=$1234
```

or

```plaintext
>IST=$5678
```

Relocation of the PCB

The PCB is completely relocatable in memory or I/O. It contains an interrupt controller, two timers, three counters, two DMA channels (8018x and 80C18x) and chip select circuitry for decoding memory and I/O space. For the 80C18x and 80C18xEB, the PCB also contains a dynamic RAM refresh controller and a power save mode controller. The 80C186EB/C188EB also contains two serial communications channels and two general purpose I/O ports. 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 a 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 ES 1800 board and then passes a copy of the registers to ESL. The copy that is sent to ESL is what is shown to you. 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 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 a pause-to-run transition, the registers are passed from ESL to the firmware. 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 ES 1800 into either run mode or execute a single step command (STP).

On a run-to-pause transition, the firmware locates the PCB and moves it back to the power-up location of $FF00 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.
Set Up Target Environment: Check Registers

If you use the MIO command to write to the PCB and change the contents of the registers, the following situations may cause confusion:

<table>
<thead>
<tr>
<th>Situation</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. You can’t find the PCB at the location you expect it.</td>
<td>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 a pause-to-run transition. Look for it at 0FF00 in I/O space.</td>
</tr>
<tr>
<td>2. If you modify a PCB register directly, using the MIO command, and then look at the PCB registers through the ESL command (PCB) you will find that the register you changed in the PCB was not changed in the ESL RAM table.</td>
<td>The values in the ESL RAM table are only loaded from the PCB on a run-to-pause transition. Also, the values loaded back into the PCB on a pause-to-run transition are from the ESL RAM table and therefore write over the top of anything that you put into the PCB. To avoid this problem, change the PCB registers using the ESL command format register=value.</td>
</tr>
<tr>
<td>3. If you modify a PCB register directly, by using the MIO command, and then go into run mode, you will find that the CPU did not use the value you changed in the PCB.</td>
<td>Commands do not modify the current contents of the physical PCB until the next pause-to-run transition.</td>
</tr>
</tbody>
</table>

When the ES 1800 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 ES 1800 enters the target system.

Using Peripherals During Pause

The ES 1800 may be configured to allow some or all of the integrated peripherals controlled by the PCB to continue operating during pause mode. See the ON/OFF menu.

The dynamic RAM refresh registers are controlled by the PRE switch, and can be used to enable continuous refresh of target RAM during pause mode.

Timers

The ON/OFF TE switches are used to enable/disable the integrated timers during pause mode.
If the switch is set to ON, on a run-to-pause transition, the timer registers are handled as described in the General PCB Handling section. On a 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 timer 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 a pause-to-run transition, 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 (not applicable to the 80C18xEB)**

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

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

Upon a 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 (8018x and 80C18x only)**

The ON/OFF RCS switch controls the emulator's reading of the LMCS, MMCS, MPCS, and PACS registers upon a 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 (e.g., LMCS=1234). This is necessary because reading of these chip select registers enables them to drive the 80186/188/C186/C188's chip select lines.

Upon a pause-to-run transition, only the registers that have been modified during pause mode 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.
Set Up Target Environment: Check Registers

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 using ESL and then executing an STP to load them into the CPU.

The LMCS register is especially critical to emulator operation because the NMI vector is located in the LMCS memory area. When making a run-to-pause transition, whether from a run or step command, the CPU picks up its NMI vector from the emulator’s internal memory space, but it uses the target’s RDY line to complete the bus cycle. If LMCS is not setup when you enter a step command or go into run mode with a breakpoint set, the emulator may hang up waiting for a target RDY signal.

When reading the contents of the CS registers the value returned is often different from the value written into the register. This is because the CS registers have some read-only bits.

- 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.

Chip Select Registers (80C18xEB only)

Upon a pause-to-run transition, all registers are reloaded to the physical 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 using ESL and then executing an STP to load them into the CPU.

The LCS registers are especially critical to emulator operation because the NMI vector is located in the LCS memory area. When making a run-to-pause transition, whether from a run or step command, the CPU picks up its NMI vector from the emulator’s internal memory space, but it uses the target’s RDY line to complete the bus cycle. If LCS is not set up when you enter a step command or go into run mode with a breakpoint set, the emulator may hang up waiting for a target RDY signal.

When reading the contents of the CS registers the value returned is often different from the value written into the register. This is because bits 4 and 5 of the chip select registers are undefined when read, and must be written as 0.

Interrupt Controller Registers

Upon a run-to-pause transition, the poll status register (POS) is read and its value stored both to its own RAM table entry, and to the polling register (POL) table entry. The emulator does not read the poll register as this would cause any pending interrupt to be treated as if it had been serviced. When you enter the PCB command, POL and POS will contain the same value.
Because POL and POS are read-only registers, they are not reloaded to the physical PCB upon a pause-to-run transition.

For the 8018x processors, on a run-to-pause transition all interrupts are disabled because there is no way for the ES 1800 to handle interrupts during pause. This means that both externally generated and chip generated interrupts are ignored during pause mode.

For the 80C18x and 80C18xEB processors, on a run-to-pause transition all interrupts are disabled unless the IDP switch is set to ON. This is true for all interrupts except INT4 on the 80C18xEB processors. INT4 cannot be used during pause mode.

Interrupts are restored to their previous condition upon a pause-to-run transition. If interrupts occur during pause and are still pending upon a pause-to-run transition, they are serviced at that time.

**Serial Controller Registers (80C18xEB only)**

Upon a run-to-pause transition, the serial registers, except for the status registers, are read and their values stored in the RAM table. Any access to the status registers, whether read or write, causes the registers to be cleared. The status registers are only read if the RSS softswitch is set to ON.

Upon a pause-to-run transition, the status registers are not written to the physical PCB. The transmit buffer registers are not written back to the physical PCB because this would initiate a transmission if the serial port were configured.

Since the serial communications are interrupt driven, the IDP switch can be used to run the serial unit during pause mode.
Register Lists

This section lists all the registers:

- Microprocessor Registers
- Target Peripheral Control Block (PCB) Registers
- PCB Registers Used Only in iRMX (slave) Mode
- PCB Registers Used in Non-iRMX (master) Mode
- PCB Registers Used in Enhanced Mode (80C18X Only)
- Event Monitor System Registers
- General ES 1800 Registers
Set Up Target Environment: Check Registers

Table 4-1: Microprocessor Registers
8018x, 80C18x and 80C18xEB

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Length (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX, AL, AH</td>
<td>accumulator (low and high)</td>
<td>Integer</td>
<td>16,8,8</td>
</tr>
<tr>
<td>BP</td>
<td>base pointer</td>
<td>Integer</td>
<td>16</td>
</tr>
<tr>
<td>BX, BL, BH</td>
<td>base (low and high)</td>
<td>Integer</td>
<td>16,8,8</td>
</tr>
<tr>
<td>CS</td>
<td>code segment</td>
<td>Integer</td>
<td>16</td>
</tr>
<tr>
<td>CX, CL, CH</td>
<td>count (low and high)</td>
<td>Integer</td>
<td>16,8,8</td>
</tr>
<tr>
<td>DI</td>
<td>destination index</td>
<td>Integer</td>
<td>16</td>
</tr>
<tr>
<td>DS</td>
<td>data segment</td>
<td>Integer</td>
<td>16</td>
</tr>
<tr>
<td>DX, DL, DH</td>
<td>data (low and high)</td>
<td>Integer</td>
<td>16,8,8</td>
</tr>
<tr>
<td>ES</td>
<td>extra segment</td>
<td>Integer</td>
<td>16</td>
</tr>
<tr>
<td>FLX, FLL, FLH</td>
<td>flags (low and high)</td>
<td>Integer</td>
<td>16,8,8</td>
</tr>
<tr>
<td>IP</td>
<td>instruction pointer</td>
<td>Integer</td>
<td>16</td>
</tr>
<tr>
<td>SI</td>
<td>source index</td>
<td>Integer</td>
<td>16</td>
</tr>
<tr>
<td>SP</td>
<td>stack pointer</td>
<td>Integer</td>
<td>16</td>
</tr>
<tr>
<td>SS</td>
<td>stack segment</td>
<td>Integer</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 4-2: Target Peripheral Control Block (PCB) Registers
8018x and 80C18x only

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REL</td>
<td>relocation register</td>
</tr>
<tr>
<td>UMCS</td>
<td>upper memory chip select control</td>
</tr>
<tr>
<td>LMCS</td>
<td>lower memory chip select control</td>
</tr>
<tr>
<td>MMCS</td>
<td>mid-range memory chip select control (base address)</td>
</tr>
<tr>
<td>MPCS</td>
<td>mid-range memory chip select control (block size)</td>
</tr>
<tr>
<td>PACS</td>
<td>peripheral chip select control</td>
</tr>
</tbody>
</table>
### Table 4-2: Target Peripheral Control Block (PCB) Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC0</td>
<td>timer #0 count register</td>
</tr>
<tr>
<td>TC1</td>
<td>timer #1 count register</td>
</tr>
<tr>
<td>TC2</td>
<td>timer #2 count register</td>
</tr>
<tr>
<td>MA0</td>
<td>timer #0 max count A register</td>
</tr>
<tr>
<td>MA1</td>
<td>timer #1 max count A register</td>
</tr>
<tr>
<td>MA2</td>
<td>timer #2 max count A register</td>
</tr>
<tr>
<td>MB0</td>
<td>timer #0 max count B register</td>
</tr>
<tr>
<td>MB1</td>
<td>timer #1 max count B register</td>
</tr>
<tr>
<td>MCW0</td>
<td>timer #0 mode control word register</td>
</tr>
<tr>
<td>MCW1</td>
<td>timer #1 mode control word register</td>
</tr>
<tr>
<td>MCW2</td>
<td>timer #2 mode control word register</td>
</tr>
<tr>
<td>USRC0</td>
<td>dma #0 upper 4 bits of source address</td>
</tr>
<tr>
<td>USRC1</td>
<td>dma #1 upper 4 bits of source address</td>
</tr>
<tr>
<td>SCR0</td>
<td>dma #0 lower 16 bits of source address</td>
</tr>
<tr>
<td>SCR1</td>
<td>dma #1 lower 16 bits of source address</td>
</tr>
<tr>
<td>U_DST0</td>
<td>dma #0 upper 4 bits of destination address</td>
</tr>
<tr>
<td>U_DST1</td>
<td>dma #1 upper 4 bits of destination address</td>
</tr>
<tr>
<td>DST0</td>
<td>dma #0 lower 16 bits of destination address</td>
</tr>
<tr>
<td>DST1</td>
<td>dma #1 lower 16 bits of destination address</td>
</tr>
<tr>
<td>XC0</td>
<td>dma #0 transfer count</td>
</tr>
<tr>
<td>XC1</td>
<td>dma #1 transfer count</td>
</tr>
<tr>
<td>CW0</td>
<td>dma #0 control word</td>
</tr>
<tr>
<td>CW1</td>
<td>dma #1 control word</td>
</tr>
</tbody>
</table>
### Table 4-3: Target Peripheral Control Block (PCB) Registers

**80C18xEB only**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interrupt registers</strong></td>
<td></td>
</tr>
<tr>
<td>EOI</td>
<td>End of Interrupt</td>
</tr>
<tr>
<td>POL</td>
<td>Poll</td>
</tr>
<tr>
<td>POS</td>
<td>Poll Status</td>
</tr>
<tr>
<td>MSK</td>
<td>Interrupt Mask</td>
</tr>
<tr>
<td>PLM</td>
<td>Priority Mask</td>
</tr>
<tr>
<td>ISV</td>
<td>In-Service</td>
</tr>
<tr>
<td>IRQ</td>
<td>Interrupt Request</td>
</tr>
<tr>
<td>IST</td>
<td>Interrupt Status</td>
</tr>
<tr>
<td>TCR</td>
<td>Timer Control</td>
</tr>
<tr>
<td>SCR</td>
<td>Serial Control</td>
</tr>
<tr>
<td>INT4</td>
<td>INT4 control</td>
</tr>
<tr>
<td>INT0</td>
<td>INT0 Control</td>
</tr>
<tr>
<td>INT1</td>
<td>INT1 Control</td>
</tr>
<tr>
<td>INT2</td>
<td>INT2 Control</td>
</tr>
<tr>
<td>INT3</td>
<td>INT3 Control</td>
</tr>
<tr>
<td><strong>Timer registers</strong></td>
<td></td>
</tr>
<tr>
<td>(Common to all: 8018x, 80C18x and 80C18xEB)</td>
<td></td>
</tr>
<tr>
<td>TC0</td>
<td>timer #0 count register</td>
</tr>
<tr>
<td>TC1</td>
<td>timer #1 count register</td>
</tr>
<tr>
<td>TC2</td>
<td>timer #2 count register</td>
</tr>
<tr>
<td>MA0</td>
<td>timer #0 max count A register</td>
</tr>
<tr>
<td>MA1</td>
<td>timer #1 max count A register</td>
</tr>
<tr>
<td>MA2</td>
<td>timer #2 max count A register</td>
</tr>
<tr>
<td>MB0</td>
<td>timer #0 max count B register</td>
</tr>
</tbody>
</table>
Table 4-3: Target Peripheral Control Block (PCB) Registers
80C18xEB only

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MB1</td>
<td>timer #1 max count B register</td>
</tr>
<tr>
<td>MCW0</td>
<td>timer #0 mode control word register</td>
</tr>
<tr>
<td>MCW1</td>
<td>timer #1 mode control word register</td>
</tr>
<tr>
<td>MCW2</td>
<td>timer #2 mode control word register</td>
</tr>
<tr>
<td></td>
<td><strong>Port control registers</strong></td>
</tr>
<tr>
<td>PDR1</td>
<td>port 1 direction</td>
</tr>
<tr>
<td>PPN1</td>
<td>port 1 pin</td>
</tr>
<tr>
<td>PCN1</td>
<td>port 1 control</td>
</tr>
<tr>
<td>PLT1</td>
<td>port 1 latch</td>
</tr>
<tr>
<td>PDR2</td>
<td>port 2 direction</td>
</tr>
<tr>
<td>PPN2</td>
<td>port 2 pin</td>
</tr>
<tr>
<td>PCN2</td>
<td>port 2 control</td>
</tr>
<tr>
<td>PLT2</td>
<td>port 2 latch</td>
</tr>
<tr>
<td>SBD0</td>
<td>serial 0 baud</td>
</tr>
<tr>
<td>SCT0</td>
<td>serial 0 count</td>
</tr>
<tr>
<td>SCN0</td>
<td>serial 0 control</td>
</tr>
<tr>
<td>SST0</td>
<td>serial 0 status</td>
</tr>
<tr>
<td>SRB0</td>
<td>serial 0 RBUF</td>
</tr>
<tr>
<td>STB0</td>
<td>serial 0 TBUF</td>
</tr>
<tr>
<td>SBD1</td>
<td>serial 1 baud</td>
</tr>
<tr>
<td>SCT1</td>
<td>serial 1 count</td>
</tr>
<tr>
<td>SCN1</td>
<td>serial 1 control</td>
</tr>
<tr>
<td>SST1</td>
<td>serial 1 status</td>
</tr>
<tr>
<td>SRB1</td>
<td>serial 1 RBUF</td>
</tr>
<tr>
<td>STB1</td>
<td>serial 1 TBUF</td>
</tr>
</tbody>
</table>

Preparing for Emulation
Set Up Target Environment: Check Registers

Table 4-3: Target Peripheral Control Block (PCB) Registers 80C18xEB only

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Chip Select Registers</td>
</tr>
<tr>
<td>STR0</td>
<td>GSC0 start</td>
</tr>
<tr>
<td>STP0</td>
<td>GSC0 stop</td>
</tr>
<tr>
<td>STR1</td>
<td>GSC1 start</td>
</tr>
<tr>
<td>STP1</td>
<td>GSC1 stop</td>
</tr>
<tr>
<td>STR2</td>
<td>GSC2 start</td>
</tr>
<tr>
<td>STP2</td>
<td>GSC2 stop</td>
</tr>
<tr>
<td>STR3</td>
<td>GSC3 start</td>
</tr>
<tr>
<td>STP3</td>
<td>GSC3 stop</td>
</tr>
<tr>
<td>STR4</td>
<td>GSC4 start</td>
</tr>
<tr>
<td>STP4</td>
<td>GSC4 stop</td>
</tr>
<tr>
<td>STR5</td>
<td>GSC5 start</td>
</tr>
<tr>
<td>STP5</td>
<td>GSC5 stop</td>
</tr>
<tr>
<td>STR6</td>
<td>GSC6 start</td>
</tr>
<tr>
<td>STP6</td>
<td>GSC6 stop</td>
</tr>
<tr>
<td>STR7</td>
<td>GSC7 start</td>
</tr>
<tr>
<td>STP7</td>
<td>GSC7 stop</td>
</tr>
<tr>
<td>LCT</td>
<td>LCS start</td>
</tr>
<tr>
<td>LCP</td>
<td>LCS stop</td>
</tr>
<tr>
<td>UCT</td>
<td>UCS start</td>
</tr>
<tr>
<td>UCP</td>
<td>UCS stop</td>
</tr>
<tr>
<td></td>
<td>Refresh and power registers</td>
</tr>
<tr>
<td>REL</td>
<td>relocation</td>
</tr>
<tr>
<td>RFBS</td>
<td>refresh base</td>
</tr>
<tr>
<td>RFTM</td>
<td>refresh time</td>
</tr>
<tr>
<td>RFLN</td>
<td>refresh control</td>
</tr>
</tbody>
</table>
**Set Up Target Environment: Check Registers**

Table 4-3: Target Peripheral Control Block (PCB) Registers  
80C18xEB only

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFAD</td>
<td>refresh address</td>
</tr>
<tr>
<td>PMC</td>
<td>power management control</td>
</tr>
</tbody>
</table>

Table 4-4: PCB Registers Used Only in iRMX (slave) Mode (8018x and 80C18x only)

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

Table 4-5: PCB Registers Used in Non-iRMX (master) Mode (8018x and 80C18x only)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POL</td>
<td>poll register</td>
</tr>
<tr>
<td>POS</td>
<td>poll status register</td>
</tr>
<tr>
<td>MSK</td>
<td>mask register</td>
</tr>
<tr>
<td>PLM</td>
<td>priority level mask register</td>
</tr>
<tr>
<td>ISV</td>
<td>in service register</td>
</tr>
</tbody>
</table>
### Table 4-5: PCB Registers Used in Non-iRMX (master) Mode (8018x and 80C18x only)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRQ</td>
<td>interrupt request register</td>
</tr>
<tr>
<td>IST</td>
<td>interrupt status register</td>
</tr>
<tr>
<td>EOI</td>
<td>end of interrupt register</td>
</tr>
<tr>
<td>TCR</td>
<td>timer interrupt control register</td>
</tr>
<tr>
<td>DMA0</td>
<td>dma #0 interrupt control register</td>
</tr>
<tr>
<td>DMA1</td>
<td>dma #1 interrupt control register</td>
</tr>
<tr>
<td>INT0</td>
<td>interrupt control register #0</td>
</tr>
<tr>
<td>INT1</td>
<td>interrupt control register #1</td>
</tr>
<tr>
<td>INT2</td>
<td>interrupt control register #2</td>
</tr>
<tr>
<td>INT3</td>
<td>interrupt control register #3</td>
</tr>
</tbody>
</table>

### Table 4-6: PCB Registers Used in Enhanced Mode (80C18x Only)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDR</td>
<td>DRAM memory partition register</td>
</tr>
<tr>
<td>CDR</td>
<td>DRAM clock pre-scalar register</td>
</tr>
<tr>
<td>EDR</td>
<td>DRAM enable RCU register</td>
</tr>
<tr>
<td>PDC</td>
<td>Power save control register</td>
</tr>
</tbody>
</table>

### Table 4-7: Event Monitor System Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Length (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC1.1-AC1.4</td>
<td>address comparator</td>
<td>Range</td>
<td>24</td>
</tr>
<tr>
<td>AC2.1-AC2.4</td>
<td>address comparator</td>
<td>Range</td>
<td>24</td>
</tr>
<tr>
<td>CTL.1-CTL.4</td>
<td>count limit comparator</td>
<td>Integer</td>
<td>16</td>
</tr>
<tr>
<td>DC1.1-DC1.4</td>
<td>data comparator</td>
<td>Don’t care</td>
<td>16</td>
</tr>
<tr>
<td>DC2.1-DC2.4</td>
<td>data comparator</td>
<td>Don’t care</td>
<td>16</td>
</tr>
<tr>
<td>LSA.1-LSA.4</td>
<td>logic state comparator</td>
<td>Don’t care</td>
<td>16</td>
</tr>
<tr>
<td>S1.1-S1.4</td>
<td>status comparator</td>
<td>Don’t care</td>
<td>16</td>
</tr>
</tbody>
</table>
### Table 4-7: Event Monitor System Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Length (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2.1-S2.4</td>
<td>status comparator</td>
<td>Don’t care</td>
<td>16</td>
</tr>
<tr>
<td>SIA</td>
<td>special interrupt address</td>
<td>Integer</td>
<td>32</td>
</tr>
</tbody>
</table>

### Table 4-8: General ES 1800 Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Length (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTO</td>
<td>ms to wait before NO BUS CYCLES error</td>
<td>Integer</td>
<td>8</td>
</tr>
<tr>
<td>DFB</td>
<td>default base</td>
<td>Integer</td>
<td>8</td>
</tr>
<tr>
<td>GD0-GD7</td>
<td>general purpose data</td>
<td>Don’t care</td>
<td>32</td>
</tr>
<tr>
<td>GR0-GR7</td>
<td>general purpose range</td>
<td>Range</td>
<td>32</td>
</tr>
<tr>
<td>IDX</td>
<td>repeat index register</td>
<td>Integer</td>
<td>32</td>
</tr>
<tr>
<td>IOP</td>
<td>I/O mode pointer</td>
<td>Integer</td>
<td>16</td>
</tr>
<tr>
<td>LIM</td>
<td>repeat limit register</td>
<td>Integer</td>
<td>32</td>
</tr>
<tr>
<td>MMP</td>
<td>memory mode pointer</td>
<td>Integer</td>
<td>32</td>
</tr>
<tr>
<td>OVE</td>
<td>overlay enable</td>
<td>Don’t care</td>
<td>8</td>
</tr>
<tr>
<td>TST</td>
<td>terminator for repeats</td>
<td>Integer</td>
<td>32</td>
</tr>
</tbody>
</table>
Set Up Soft Switches

If you have target hardware, the ON/OFF menu contains switches which allow you to configure the emulation environment to your liking. For example, you can run the ES 1800 without a target system by using the ES 1800-supplied clock signal, an emulator-generated ready signal and overlay memory. The copy switch (CPY) copies data to both serial ports for obtaining hard copy of your emulation session.

The ON/OFF menu can be saved to EEPROM with the SAV 4 command. These values may then be automatically loaded into the ES 1800 on power-up by setting the thumbwheel switch to the appropriate value, or manually by typing the load command (LD 4) to the ES 1800 after power-up.

The following chart summarizes the switches. More information can be found in Section 7 under each switch name.

Table 4-9: Soft Switches

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>8018x</th>
<th>80C18x</th>
<th>80C18xEB</th>
</tr>
</thead>
<tbody>
<tr>
<td>BKX</td>
<td>Break on instruction execution (not prefetch)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>BTE</td>
<td>BUS(RDY) timeout enable</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CDH</td>
<td>Clear DHLT bit in IST register on a pause-to-run</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CK</td>
<td>Select internal clock</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CPY</td>
<td>Copy data to TERMINAL &amp; COMPUTER ports</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DME</td>
<td>Enable DMA during pause</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>FSX</td>
<td>FSI on instruction execution (not prefetch)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IDP</td>
<td>Enable interrupts during pause</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>IHE</td>
<td>Ignore halt errors</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>PRE</td>
<td>Refresh enable during pause</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PPT</td>
<td>Enable peek/poke trace</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>RCS</td>
<td>Enable chip select registers display</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>RDX</td>
<td>Select internal ready when accessing overlay</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
### Table 4-9: Soft Switches

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>8018x</th>
<th>80C18x</th>
<th>80C18xEB</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSS</td>
<td>Enable reading of serial status</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>STI</td>
<td>Enable step through interrupts</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TCE</td>
<td>Enable trace memory during run</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TE0</td>
<td>Enable timer 0 during pause</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TE1</td>
<td>Enable timer 1 during pause</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>TE2</td>
<td>Enable timer 2 during pause</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Run Your Program

This section explains how to run and stop your program.

To run your program, you must put the emulator into run mode. You can enter run mode by executing any of four run commands. You can also single step your program using the STP command. The STI switch controls whether the emulator should recognize or ignore interrupts while single stepping.

Emulation can be halted in one of four ways, single stepping, manual reset, reaching an error or reaching a breakpoint preset with the Event Monitor System. Before running your program, you should choose a method for stopping emulation. The method you choose depends on what data you want to look at when emulation stops.

Event monitor system breakpoints may be enabled or disabled during run mode. Even when breakpoints are disabled, all other Event Monitor System functions are active.

### Commands Used to Start and Stop Emulation

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Start Emulation</strong></td>
<td></td>
</tr>
<tr>
<td>LDV</td>
<td>Load reset vectors</td>
</tr>
<tr>
<td>RBK</td>
<td>Run with breakpoints enabled</td>
</tr>
<tr>
<td>RBV</td>
<td>Run, load reset vectors, breakpoints enabled</td>
</tr>
<tr>
<td>RNV</td>
<td>Run, load reset vectors, breakpoints disabled</td>
</tr>
<tr>
<td>RUN</td>
<td>Run with breakpoints disabled</td>
</tr>
<tr>
<td>STI</td>
<td>Step through interrupts</td>
</tr>
<tr>
<td>STP</td>
<td>Step through target system</td>
</tr>
<tr>
<td><strong>Stop Emulation</strong></td>
<td></td>
</tr>
<tr>
<td>BKX</td>
<td>Break on instruction execution or address</td>
</tr>
<tr>
<td>BRK</td>
<td>Break emulation</td>
</tr>
<tr>
<td>FSI</td>
<td>Force special interrupt</td>
</tr>
<tr>
<td>FSX</td>
<td>FSI on instruction execution</td>
</tr>
<tr>
<td>RST</td>
<td>Reset pod microprocessor, load reset vectors</td>
</tr>
<tr>
<td>SET #2</td>
<td>Set reset character</td>
</tr>
<tr>
<td>WHEN</td>
<td>Enter when/then statement</td>
</tr>
</tbody>
</table>

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. The reset vectors are defined by Intel as:

\[
\begin{align*}
CS &= FFFFH \\
IP &= 0 \\
FLX &= 002H
\end{align*}
\]

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 chart is a quick reference to the RUN commands.
Run Program: Break Emulation

**Commands Used to Start Emulation**

<table>
<thead>
<tr>
<th>Command</th>
<th>Load Reset</th>
<th>Breakpoints</th>
<th>Valid in Run mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>RNV</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>RBK</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>RBV</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

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. 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.

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. Some error conditions may require a reset to bring the system back into command entry mode.

**Break Emulation**

Emulation can be halted in one of four ways. Before running your program, you should choose a method for stopping emulation. The method you choose depends on what data you want to look at when emulation stops.

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 ES 1800 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 ES 1800 memory, and the CS:IP and event monitor group number are displayed.
3. Issuing the reset character (\texttt{<ctrl-z>} default) stops emulation. After the reset character is issued, the ES 1800 registers have the same value they had before emulation began. You should check those values or load the reset vectors (\textbf{LDV}) before restarting emulation.

4. Emulation breaks automatically if the target program commits an access or write violation in overlay memory. The condition that caused the error is displayed. Breaking can also be qualified by the softswitch \textbf{BKX}. This soft switch determines if breaks will occur only on instruction execution, or on any access to an address, including prefetches.
Set Up Breakpoints

Once you have run your program, and discover a problem, the next step is typically to decide where to break so that you can find the problem. This section describes using the Event Monitor System to break emulation and to perform other actions. It begins with an overview, and then describes each unit of the Event Monitor System in detail. The end of the section includes a variety of useful examples.

**Commands Used to Decide Where to Break Emulation**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup/Display/Clear Advanced Event System</td>
<td>CES [1-4] Clear event monitor system setup</td>
</tr>
<tr>
<td>DES [1-4] Display event monitor system setup</td>
<td>WHEN Enter when/then statement</td>
</tr>
</tbody>
</table>

**Advanced Event System Actions**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRK</td>
<td>Break emulation</td>
</tr>
<tr>
<td>CNT</td>
<td>Count bus cycle</td>
</tr>
<tr>
<td>FSI</td>
<td>Force special interrupt</td>
</tr>
<tr>
<td>GRO n</td>
<td>Change event group</td>
</tr>
<tr>
<td>RCT</td>
<td>Reset count value</td>
</tr>
<tr>
<td>TGR</td>
<td>Output trigger signal</td>
</tr>
<tr>
<td>TOC</td>
<td>Toggle count state</td>
</tr>
<tr>
<td>TOT</td>
<td>Toggle trace state</td>
</tr>
<tr>
<td>TRC</td>
<td>Trace bus cycle</td>
</tr>
</tbody>
</table>

**Set Up the Event Monitor System**

The ES 1800’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 Analyzer pod inputs. When an event is detected, the ES 1800 can break emulation, trace specific sequences, count events, execute user supplied target routines, and trigger TTL outputs.

The Event Monitor System monitors target information at the bus cycle level, including every read or write cycle that the microprocessor executes. The Event Monitor system ‘sees’ every signal that can affect the target system. It can also monitor inputs from the logic state analyzer probe.
The Intel 8018x/C18x/C18xEB microprocessors multiplex address and data lines. The ES 1800 demultiplexes those signals so that the Event Monitor System sees all signals at the same time. The Event Monitor system essentially takes a picture of the microprocessor's signals at the beginning of every T4 state (refer to the Intel manuals iAPX 86/88, 186/188 Users Manual, iAPX C86/C88, C186/C188 Users Manual and the 80C186EB/C188EB Users Manual). The information that is recorded into trace memory is the same information that the Event Monitor system is monitoring.

The address comparators in the 80186/188, C186/C188 and C186EB/C188EB may need to be specially set up. These are 16-bit chips, with a prefetch QUE and byte based instructions. This causes problems when breaking on instructions that occur on odd boundaries.

You can enter Event Monitor System WHEN/THEN 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.*

NOTE: Simultaneous use of the Dynamic Trace feature and the Event monitor system is not possible. (See TCE in Section 7).

**Structure**

The Event Monitor System is structured in three basic units:

- **Events**
  - Events identify specific target conditions. When these conditions are encountered, actions can be performed.

- **Actions**
  - Actions are what the emulator does when an event is detected. There are many actions that the event system can take, including standard features such as forcing a special interrupt to jump to a soft shutdown routine before stopping the target program, sophisticated trace control and breaking emulation.

- **WHEN/THEN Statements**
  - Statements coordinate the events and actions.

You define statements that specify single or multiple events that are logical combinations of address, data, status, counter, and optional logic field states. When those events are encountered in the target system program, the ES 1800 can break emulation, trace specific sequences, count events and trigger outputs, allowing you to analyze the cause-effect relationship established by the event/action sequences defined.

There are four event groups which provide the logical structure necessary for tracking deeply nested bugs. This structure lets you debug any problem you can imagine, using a combination of events and actions. Figure 4-1 shows the structure of the Event Monitor system.
There can be several actions for any event. There can be many WHEN/THEN statements in effect at any time.

The basic Event Monitor System WHEN/THEN statement is of the form:

\[ [Group] \text{WHEN} [event] \text{THEN} [action] \]

The system only recognizes the first three letters of any word in a control statement (e.g., WHEN=WHE; THEN=THE).

**Define Events**

You can define an event to be some combination of address, data, status, count, and Logic State Analyzer pod conditions. Numerous Event Monitor System WHEN/THEN 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 (AND and OR).

\[ \text{WHEN AC1 AND NOT DC1 THEN BRK} \]

means break whenever any data pattern other than that in DC1 is read from or written to an address in AC1.
AND and OR can be used to form more restrictive event definitions. AND terms have higher precedence than OR terms. For example:

```
WHEN AC1 AND DC1 OR DC2 THEN BRK
```

is the same as

```
WHEN AC1 AND DC1 THEN BRK
WHEN DC2 THEN BRK
```

If you are looking for two different data values at an address, you would use

```
WHEN AC1 AND DC1 OR AC1 AND DC2 THEN BRK
```

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 WHEN/THEN statements for clarity.

```
WHEN AC1 AND S1 OR AC2 AND S2 THEN BRK
```

is the same as

```
WHEN AC1 AND S1 THEN BRK
WHEN AC2 AND S2 THEN BRK
```

There are eight comparator registers for each of the four event groups. These event registers are listed in the following table.

<table>
<thead>
<tr>
<th>Comparator Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>address comparators</td>
<td>Used to detect discrete addresses or addresses inside or outside a specified range.</td>
</tr>
<tr>
<td>data comparators</td>
<td>Used to detect specific data patterns (can ignore specified bit positions)</td>
</tr>
<tr>
<td>status comparators</td>
<td>Monitor all of the status signals from the microprocessor as well as some generated by the ES 1800. The status comparators can also ignore bit positions.</td>
</tr>
<tr>
<td>count limit</td>
<td>Used to detect when an event has occurred more than a specified number of times.</td>
</tr>
<tr>
<td>LSA registers</td>
<td>Detect bit patterns in the inputs from the logic state probe. Specified bit positions can be ignored.</td>
</tr>
</tbody>
</table>
Set Up Breakpoints: Events

The following table describes the available event comparator registers:

<table>
<thead>
<tr>
<th>Register Description</th>
<th>Type</th>
<th>Size (bits)</th>
<th>Name by Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address 1</td>
<td>Range, Int</td>
<td>24</td>
<td>AC1 or AC1.1</td>
</tr>
<tr>
<td>Address 2</td>
<td>Range, Int</td>
<td>24</td>
<td>AC2 or AC2.1</td>
</tr>
<tr>
<td>Data 1</td>
<td>Don’t Care, Int</td>
<td>16</td>
<td>DC1 or DC1.1</td>
</tr>
<tr>
<td>Data 2</td>
<td>Don’t Care, Int</td>
<td>16</td>
<td>DC2 or DC2.1</td>
</tr>
<tr>
<td>Status 1</td>
<td>Don’t Care, Int</td>
<td>16</td>
<td>S1 or S1.1</td>
</tr>
<tr>
<td>Status 2</td>
<td>Don’t Care, Int</td>
<td>16</td>
<td>S2 or S2.1</td>
</tr>
<tr>
<td>LSA</td>
<td>Don’t Care, Int</td>
<td>16</td>
<td>LSA or LSA.1</td>
</tr>
<tr>
<td>Count</td>
<td>Int</td>
<td>16</td>
<td>CTL or CTL.1</td>
</tr>
</tbody>
</table>

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.

```plaintext
>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 Address Boundaries

The address comparators in the 80186 family processors may need to be specially set up. These are 16-bit chips, with a prefetch QUE and byte based instructions. This causes problems when breaking on instructions that occur on odd boundaries.

This section describes three distinct conditions, and suggestions for resolving them. Any remarks about the 8018x apply to the 80C18x and 80C18xEB as well.

1. **8018x prefetches an instruction.**

   When the 8018x 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. *8018x jumps to an odd address.*

When the 8018x 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 $4001. This byte could be accessed by the address $4001 or $4000.

- If the address $4001 is on the bus, then the byte is accessed.
- If the address $4000 is on the bus, and the bus cycle is a 16-bit cycle, then the byte is accessed.
- If the address $4000 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:

> \textsc{AC1}=4000  
> \textsc{AC2}=4001  
> \textsc{S1}=WRD  
> \text{WHEN AC1 AND S1 OR AC2 THEN BRK}

\textsc{AC1} contains the even address. \textsc{S1} is the word bus cycle condition. If both are true, the high or odd byte has been accessed. \textsc{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 signals from the Logic State Analyzer pod.

Data and LSA comparators may be assigned integer values or don’t care values. Don’t care values may be assigned in two ways.

1. The first is to specify the value followed by the don’t care mask
2. The second is to specify the value using \texttt{X} in the don’t care positions.

The following are examples of valid data and LSA comparator assignments.

> \texttt{DC1}=237F  
> \texttt{LSA}=5300 DC $FF
Set Up Breakpoints: Events

>LSA.3 = 53XX
>LSA = %110101 DC $FF00
>DC2.2 = 42 DC %101
>DC2 = GDO + $F
>DC1.4 = @'data_table + 56

The following example shows turning on trace when an activity occurs and turning off the trace when the activity finishes. Note the use of two event groups to specify the on/off conditions. This setup waits for the logic state analyzer bit 0 to go low, and then uses the toggle trace command (TOT) to turn on trace memory, and GRO 2 to switch groups. In group 2, all bus cycles are traced until LSA pod bit 0 goes high. Then emulation is broken.

>WHEN LSA THEN TOT, GRO 2
>2 WHEN LSA THEN BRK
>LSA = 0 DC $FFFE
>LSA.2 = 1 DC $FFFE

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 status lines can be used in event specifications.

STATUS MNEMONICS

<table>
<thead>
<tr>
<th>ALT</th>
<th>Alternate Data Access</th>
<th>QD1-6</th>
<th>Queue Depth (1-6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYT</td>
<td>Byte Access</td>
<td>QF</td>
<td>Queue Flush Cycle</td>
</tr>
<tr>
<td>COD</td>
<td>Code Access</td>
<td>RD</td>
<td>Read</td>
</tr>
<tr>
<td>DAT</td>
<td>Data access</td>
<td>RIO</td>
<td>Read IO Status</td>
</tr>
<tr>
<td>HLT</td>
<td>Halt Status</td>
<td>RM</td>
<td>Read Memory Status</td>
</tr>
<tr>
<td>IAK</td>
<td>Interrupt Acknowledge Status</td>
<td>STA</td>
<td>Stack Access</td>
</tr>
<tr>
<td>IF</td>
<td>Instruction Fetch Status</td>
<td>TAR</td>
<td>Target Access</td>
</tr>
<tr>
<td>IOA</td>
<td>IO Access</td>
<td>WIO</td>
<td>Write IO Status</td>
</tr>
<tr>
<td>MEM</td>
<td>Memory Access</td>
<td>WM</td>
<td>Write Memory Status</td>
</tr>
<tr>
<td>NMI</td>
<td>NMI Cycle</td>
<td>WR</td>
<td>Write</td>
</tr>
<tr>
<td>OVL</td>
<td>Overlay Access</td>
<td>WRD</td>
<td>Word Access</td>
</tr>
<tr>
<td>DMA</td>
<td>DMA Cycle (8018x and 80C18x only)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 (+) as a Boolean AND.
Set Up Breakpoints: Events

STATUS MNEMONIC TABLE

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>TAR</td>
<td>RD</td>
<td>BYT</td>
<td>MEM</td>
<td>ALT</td>
<td>HLT</td>
<td>QD1</td>
<td>QF</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>OVL</td>
<td>WR</td>
<td>WRD</td>
<td>IOA</td>
<td>COD</td>
<td>IAK</td>
<td>QD2</td>
<td>NMI</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DAT</td>
<td>RIO</td>
<td>QD3</td>
<td>DMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STA</td>
<td>RM</td>
<td>QD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WIO</td>
<td>QD5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WM</td>
<td>QD6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Some examples of status comparator assignments:

\[ S1 = \text{BYT} \]
\[ S2 = \text{OVL} + \text{RD} + \text{DAT} \]
\[ S1.3 = \text{WR} + \text{IOA} \]
\[ S2.4 = \text{RIO} \]
\[ S1.2 = \text{QF} \]

Figure 4-2: Status Translation Table

<table>
<thead>
<tr>
<th>SEGMENT</th>
<th>CPU STATUS</th>
<th>QUE DEPTH</th>
<th>EMULATOR STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMI=0</td>
<td>ALT=0</td>
<td>QD1=1</td>
<td>MEM=1 TAR=1 RIO=0 WR=0</td>
</tr>
<tr>
<td>STA=1</td>
<td>IAK=0</td>
<td>QD2=2</td>
<td>RD=1 BYT=1</td>
</tr>
<tr>
<td>COD=2</td>
<td>RIO=1</td>
<td>QD3=3</td>
<td>QF=0 IOA=0 OVL=0</td>
</tr>
<tr>
<td>DAT=3</td>
<td>WIO=2</td>
<td>QD4=4</td>
<td>WR=0 WRD=0</td>
</tr>
<tr>
<td></td>
<td>HLT=3</td>
<td>QD5=5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IF=4</td>
<td>QD6=6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RM=5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WM=6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When you display the value of the status comparators, you will see a 32-bit don’t care value rather than the mnemonics you originally assigned them. The Status Translation Table is provided to aid you in decoding the numbers back into the mnemonics.

The don’t care mask is the value to the right of the DC. A ’0’ in a mask bit position enables the status bit in the same position on the left side of the DC, and a ’1’ in a mask bit position masks or disables the corresponding bit on the left side of the DC.

Determine which bit positions are unmasked (those containing 0’s in the mask value). It may be easier to do this by setting the status comparator’s display base to binary ( BAS S1 = 2 ). Then refer to the translation table and find the unmasked bit positions. Look at the value contained on the left side of the DC and match it with the corresponding value shown underneath the bit position in the table.

\[ S1 = 00000504 \text{ DC } 0000B8F8 \]
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:

\[ S_1 = \text{NMI} + \text{RM} + \text{TAR} \]

**NOTE**

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 reset (<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 actions.

**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 Define Action Lists). 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.

**Define WHEN/THEN Statements**

The syntax of WHEN/THEN statements is:

\[ [\text{group}] \text{WHE} \text{<events> THE <action>, <action>...} \]

This will cause the emulator to take the specified actions when the events are reached.

The Event Monitor System is arranged in four independent groups. Each WHEN/THEN statement is associated with one of the four groups. If no group numbers are mentioned in the WHEN/THEN statement, the statement is assigned to group 1. There are two ways to override this default selection of group 1. You can begin the WHEN/THEN 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 WHEN/THEN statement.

**Define Action Lists**

The action list in a WHEN/THEN statement defines what the ES 1800 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.
The following table lists all possible actions. Each action is described in detail in Section 7: “Alphabetical Command Reference.”

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRK</td>
<td>Break emulation</td>
</tr>
<tr>
<td>CNT</td>
<td>Count bus cycle</td>
</tr>
<tr>
<td>FSI</td>
<td>Force special interrupt</td>
</tr>
<tr>
<td>GROn</td>
<td>Change event group</td>
</tr>
<tr>
<td>RCT</td>
<td>Reset count value</td>
</tr>
<tr>
<td>TGR</td>
<td>Output trigger signal</td>
</tr>
<tr>
<td>TOC</td>
<td>Toggle count state</td>
</tr>
<tr>
<td>TOT</td>
<td>Toggle trace state</td>
</tr>
<tr>
<td>TRC</td>
<td>Trace bus cycle</td>
</tr>
</tbody>
</table>

For details on the actions, see Section 7, Alphabetical Command Reference.

The Event Monitor System resolves conflicting WHEN/THEN statements. For example, the TOC action in the first statement is ignored.

>`WHEN ACl THEN TOC
>`WHEN ACl THEN CNT

### Event Monitor System Examples

There are three examples shown on the following pages:

1. Using the trigger out action to display the duration of a software routine on an oscilloscope.
2. Using the force special interrupt action to safely stop a mechanical system.
3. Debugging a suspected problem in a belt jam routine that uses reentrant code.

#### Example 1

The trigger out action (TGR) can be used to trigger a logic analyzer, oscilloscope or counter-timer. In this example, it is used to display the duration of a software routine on an oscilloscope.

Three actions are done at the same time in this example. When the routine starts, trace is turned on (TRC), the trigger out is started (TGR), and we switch to event group 2 (GRO 2). Note the use of symbols: the symbols `'sub_start` and `'sub_end`.

>`AC1 = 'sub_start
>`AC1.2 = 'sub_end

---

Preparing for Emulation
Set Up Breakpoints: Examples

>DC1.2 = 0XXXX
Set a data comparator (DC1.2) to don't cares (XXXX) to keep the trigger high.

>WHEN AC1 THEN TRC, TGR, GRO 2
In group 1, at the beginning of the subroutine, start the trace (TRC), set the trigger high (TGR) and switch to group 2 (GRO 2).

> 2 WHEN DC1 THEN TRC, TGR
In group 2, use DC1 as a dummy value, used to keep the trace on and the trigger high during the subroutine.

> 2 WHEN AC1 THEN GRO 1
At the subroutine end (AC1.2), return to group 1 and stop the trace and trigger pulse.

Figure 4-3: Display the Duration of a Software Routine on An Oscilloscope Using the Trigger Out

Example 2
The problem with debugging a mechanical system like a robot arm is that any interruption to the controlling software may cause the system to crash. The Event Monitor System provides a special interrupt system so that when a specified breakpoint is reached, a soft shutdown routine can safely stop the mechanical system, and only then is the program stopped to locate the problem.

>SIA = 'shut_down
Set the special interrupt address (SIA) to the address of the soft shutdown routine, specified by the symbol 'shut_down.

>AC1 = $7F4B2
Set the first address comparator (AC1) to the address of the suspected problem where you want to break emulation.

>AC2 = 'shut_down + 4E
Set the second address comparator (AC2) to the end of the soft shutdown routine.
>WHEN AC1 THEN FSI

When you get to the address where you want to break, first execute the forced special interrupt (FSI).

>WHEN AC2 THEN BRK

When you get to the end of the 'shut_down routine, break emulation (BRK).

>RBK

Run to the breakpoint.

Figure 4-4: Safely Debug a Problem with a Robot Arm by Jumping to a Specified Address and Executing a Soft Shutdown

Example 3

In this example, debugging a suspected problem in a belt jam routine requires debugging reentrant code. The state diagram identifies the route of suspected trouble: the problem occurs only after initialization, when the specified belt is stuck (belt C on conveyor 2), and the jam routine is called with a particular value.

Note that the program continues to execute in real-time while several events isolate the problem. The breakpoint is set only after the exact program state is identified.
Set Up Breakpoints: Examples

Figure 4-5: Debugging a Problem in a Belt Jam Routine

AC1 = 'end_init

Group 1 is used to step over the initialization routine.

WHE AC1 THE GRO 2

This is done to make sure that initialization is complete.

AC1.2 = 'conveyor2
2 WHE AC1 THE GRO 3

Group 2 is used to specify that you are only interested in when conveyor#2 calls the routine that checks the belts.

AC1.3='checkbelts

Group 3 is used to specify that the checkbelt routine has identified that belt C is the one with the problem. This is specified in your code by bit 3 at the address 'checkbelts.

DC1.3 = 0004 DC 0FF7

Use the data comparator (DC1.3) to specify the value read at the address AC1.3. 0004 DC 0FF7 means to check bit 3 of the data word (0004), and ignore the other bits (DC 0FF7).

S1.3 = RD

Use the status comparator (S1.3) to qualify only reads from address AC1.3.

3 WHE AC1 AND DC1 AND S1 THE RCT, GRO 4

When all these conditions are met, it is time to go to group 4 (GRO 4) and to reset the
counter (RCT) so you can use it in group 4. Group 4 is used to identify the portion of the beltjam routine which you suspect contains the problem.

AC1.4 = 'beltjam LEN 400

Set the address comparator in group 4 (AC1.4) to a range which starts at the beginning of the beltjam routine.

S1.4 = IF

Use the status comparator (S1.4) to monitor for an instruction fetch (IF) from the range AC1.4.

CTL.4 = #100

Set the count limit to 100, so that you can break after the first 100 instruction in the routine. This assumes that you suspect the problem is in these instructions.

4 WHE AC1 AND S1 THE CNT

When you’re in the beltjam routine, increment the counter at every instruction fetch.

4 WHE CTL THE BRK

When the count limit is reached, then break.

RBK

Run to the breakpoint. The events leading up to the breakpoint are checked while the software is running in real time.

Using Software Debuggers

There are some constraints and differences in operation when using the Event Monitor system with some software debuggers.

ES Driver

The Target Emulation menu allows transparent access to setting up the event monitor system: in transparent mode, you enter ESL commands just as you would when using the ES 1800 without a host computer.

The Event Monitor System menu provides a convenient display of the set up. For 68020 processors, the Event Monitor System menu also provides a convenient way to set up the Event Monitor System without typing in ESL commands.
VALIDATE/XEL

When you use VALIDATE/XEL, you must enter ICE mode in order to access the Event Monitor System and ESL. Once in ICE mode, you enter ESL commands just as you would when using the ES 1800 without a host computer. To return to VALIDATE/XEL, type NOICE.

VALIDATE/Soft-Scope

When you use VALIDATE/Soft-Scope or VALIDATE/Soft-Scope 286, you must use the CONSOLE command in order to access to the Event Monitor System and ESL. Once connected, you enter ESL commands just as you would when using the ES 1800 without a host computer. Use a Q to return to VALIDATE/Soft-Scope.

XDB

When you use XDB, you must use the "Interactive Transparency Mode" in order to access the Event Monitor System and ESL. The o command enters interactive transparency mode. Once connected, you enter ESL commands just as you would when using the ES 1800 without a host computer. Use a <ctrl-d> to return to XDB.

GeneProbe

When using ESL with GeneProbe, you can suppress GeneProbe’s command processing by prefixing the line with a semicolon (;). This allows you to use ESL expressions if you need to use them. For example:

;AC1=$FDE02
Isolate a Problem

There are two parts to isolating a problem:

1. If you can’t make your target program run, you can often use overlay to determine if the problem is in software or hardware.

2. Once you have an idea of where a problem is occurring, you can use the breakpoints of the Event Monitor System to stop program execution at specific times and then disassemble the trace memory, look at the LSA bits in the raw trace, check the CPU register values, or begin stepping through your code.

This section describes the commands used to examine trace memory, registers and other status information.

### Commands Used to Isolate a Problem

<table>
<thead>
<tr>
<th><strong>Command</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Run Program from Overlay Commands</strong></td>
<td></td>
</tr>
<tr>
<td>LOV</td>
<td>Load overlay from target memory</td>
</tr>
<tr>
<td>MAP</td>
<td>Map overlay memory</td>
</tr>
<tr>
<td><strong>Trace Commands</strong></td>
<td></td>
</tr>
<tr>
<td>DRT</td>
<td>Display raw trace bus cycles</td>
</tr>
<tr>
<td>DT</td>
<td>Disassemble trace memory</td>
</tr>
<tr>
<td>DTB</td>
<td>Disassemble previous page of trace memory</td>
</tr>
<tr>
<td>DTF</td>
<td>Disassemble next page of trace memory</td>
</tr>
<tr>
<td>TCE</td>
<td>Trace capture enable</td>
</tr>
<tr>
<td>TRC</td>
<td>Trace events</td>
</tr>
<tr>
<td><strong>Register Commands</strong></td>
<td></td>
</tr>
<tr>
<td>BAS</td>
<td>Change default register display base</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear CPU registers</td>
</tr>
<tr>
<td>DR</td>
<td>Display registers</td>
</tr>
<tr>
<td>LD 1</td>
<td>Load register set from EEPROM</td>
</tr>
<tr>
<td>LDV</td>
<td>Load reset vectors</td>
</tr>
<tr>
<td>ON/OFF</td>
<td>Control various registers</td>
</tr>
<tr>
<td>PCB</td>
<td>Display PCB registers</td>
</tr>
<tr>
<td>SAV 1</td>
<td>Save register set into EEPROM</td>
</tr>
<tr>
<td><strong>Single Step Commands</strong></td>
<td></td>
</tr>
<tr>
<td>STI</td>
<td>Single step through interrupts</td>
</tr>
<tr>
<td>STP</td>
<td>Single step through program</td>
</tr>
</tbody>
</table>
Isolate Problem: Run Program from Overlay

**Commands Used to Isolate a Problem (cont)**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Miscellaneous Useful Problem Isolation Commands</strong></td>
<td></td>
</tr>
<tr>
<td>BUS</td>
<td>Display status of bus status lines</td>
</tr>
<tr>
<td>COM</td>
<td>Communication with target programs</td>
</tr>
<tr>
<td>CPY</td>
<td>Copy data to both ports</td>
</tr>
<tr>
<td>DIA</td>
<td>Display character string</td>
</tr>
<tr>
<td>RET</td>
<td>Insert a blank line in display</td>
</tr>
<tr>
<td>TGR</td>
<td>Send trigger signal</td>
</tr>
<tr>
<td>WAI</td>
<td>Wait until emulation break</td>
</tr>
<tr>
<td><strong>Uploading Data to Host Computer Commands</strong></td>
<td></td>
</tr>
<tr>
<td>UPL</td>
<td>Upload data to host</td>
</tr>
<tr>
<td>UPS</td>
<td>Upload symbol table to host</td>
</tr>
</tbody>
</table>

**Run Program from Overlay**

If your program doesn’t seem to run correctly in your target system, you can try running it from overlay instead. Map the appropriate address range using the MAP command, and load the program from your target memory using LOV.

This can help isolate target hardware problems such as addresses not being decoded properly, timing problems, or memory accesses not being terminated properly.

**Examine the Trace Memory**

Trace is your window to the activity of the microprocessor. You can disassemble the trace buffer to see assembly instructions or you can look at raw trace to see the status of the CPU during each bus cycle. You will probably need to use both of these commands to get enough information to solve a problem.

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.

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.
There are several commands available to display trace in different formats: **DRT** for raw trace, and **DT** for disassembled trace. You can scroll the trace buffer with the **DTB** and **DTF** commands. The **WAI** command is used to wait until execution stops to execute a particular command.

The **DIA** command can be used to check the contents of any null terminated string in target memory. One common use is 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 ES 1800 to break emulation. The **DIA** command can then be used to display the error message.

You cannot access trace memory during emulation unless you have the Dynamic Trace feature. Therefore, you must stop program execution before reading the trace. You can stop the program either manually or by using the Event Monitor System to stop at the exact program state you are interested in. After program execution is stopped, you may review the address, data and control signals of the most recently traced cycles.

### Dynamic Trace (Optional)

The Dynamic Trace feature of the ES 1800 allows you to read trace while the target is running. You can trace in target systems which require the program to remain running, such as control systems. With targets using multiple multiprocessors, dynamic trace lets you examine trace from one processor without shutting down all processors.

Simultaneous use of the Dynamic Trace feature and the Event Monitor System is not possible. Refer to the Dynamic Trace Capture Enable command (**TCE**) in Section 7 for more information.

### Check CPU Registers

Before going into run mode, you will want to be sure that the code segment and instruction pointer (**CS:IP**) contain the correct value. You may also want to set a valid stack pointer. initialize the CPU status register (**FLX**) or some of the PCB registers.

You can either set registers by hand or use the **LDV** command to set them to the values defined by Intel at power-up.

Each register has a separate display base. The display base is viewed and changed with the **BAS** command. 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.

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.
Isolate Problem: Single Step

Single Step Through Program

From pause mode, the STP command executes one instruction. To receive visual feedback, combine this command with a trace display command such as STP;DT.

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

Miscellaneous Useful Commands

The COM command establishes a 'transparent communication mode' between the running target program and the controlling port of the ES 1800. An address is specified from which ASCII characters can be passed from the user to the target program and from the target program to the user. For example,

- The target program can ask the user a question, and the user can type an answer at the terminal.
- You can simulate I/O before hardware is read
- You can use COM in test situations

The BUS displays the status of several bus lines: NMI, ARDY, SRDY, INT0, INT1, INT2/INTA0, INT3/INTA1, and TEST. This command may be entered in run mode.

The ON CPY soft switch provides a way to make a hard copy of emulation data. It is also useful for monitoring computer control commands.
Modify Your Program

Once you have run your program, stopped at in a particular place, and isolated the problem, the next step is to design and test possible solutions to the problem. The ES 1800 emulator lets you easily modify memory either in your target or in the emulator overlay memory to make changes to your program.

This section includes information on memory commands, memory mode and I/O mode. The term 'memory' is used here to describe memory in the target system or the ES 1800's overlay memory.

Memory commands allow you to modify and display memory in five different ways.

1. Copy blocks of memory, fill blocks with a constant data pattern, search for a pattern or a particular block, and load or verify memory using memory commands.
2. Directly modify single lines in memory using the line assembler.
3. View data from memory using the memory disassembler.
4. View and modify memory using a simple scrolling scheme using memory mode.
5. View and modify I/O address space data using I/O mode.
Modify Program: Memory Commands

Commands Used to Modify the Emulation Environment

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>Read/write memory</td>
</tr>
<tr>
<td>BMO</td>
<td>Move memory block to new address</td>
</tr>
<tr>
<td>BYM</td>
<td>Set default data length to byte</td>
</tr>
<tr>
<td>DB</td>
<td>Display memory block</td>
</tr>
<tr>
<td>DIS</td>
<td>Disassemble memory</td>
</tr>
<tr>
<td>FIL</td>
<td>Fill memory with constant</td>
</tr>
<tr>
<td>FIN</td>
<td>Find pattern in memory</td>
</tr>
<tr>
<td>LOV</td>
<td>Load overlay memory from target</td>
</tr>
<tr>
<td>VBL</td>
<td>Verify pattern in memory</td>
</tr>
<tr>
<td>VBM</td>
<td>Verify block move</td>
</tr>
<tr>
<td>WDM</td>
<td>Set default data length to word</td>
</tr>
</tbody>
</table>

Line Assembler Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASM</td>
<td>Line assembler</td>
</tr>
<tr>
<td>END</td>
<td>Exit line assembler</td>
</tr>
<tr>
<td>X</td>
<td>Exit line assembler</td>
</tr>
</tbody>
</table>

Memory Mode Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>Enter memory mode</td>
</tr>
<tr>
<td>MMP</td>
<td>Display/set memory mode pointer</td>
</tr>
<tr>
<td>X</td>
<td>Exit memory mode</td>
</tr>
</tbody>
</table>

I/O Mode commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOP</td>
<td>Display I/O mode pointer</td>
</tr>
<tr>
<td>MIO</td>
<td>Enter I/O mode</td>
</tr>
<tr>
<td>X</td>
<td>Exit I/O mode</td>
</tr>
</tbody>
</table>

Memory Commands

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 for details on mapping overlay memory.

The default data length affects most memory commands. There are two data lengths to choose from: byte mode (BYM) and word mode (WDM). Commands that accept data parameters truncate the data entered to the current default data length. If you enter FIN 0 LEN 20, 23F6 and the default data length is byte mode, the find command truncates the data field to F6 and searches 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.

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.

<table>
<thead>
<tr>
<th>Command</th>
<th>Legal in Run Mode?</th>
<th>Uses Default Data Length?</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>FIN</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>FIL</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>BMO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>VBL</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>LOV</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>VFO</td>
<td>YES</td>
<td>N/A</td>
</tr>
<tr>
<td>ASM</td>
<td>YES</td>
<td>N/A</td>
</tr>
<tr>
<td>DIS</td>
<td>YES</td>
<td>N/A</td>
</tr>
<tr>
<td>M</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>MIO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>@</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

**Line Assembler**

The line assembler is used to make small modifications to your program. For example, if you wanted to branch when a variable was equal to 0, and you realize your code inadvertently checked to see if the variable was not equal to 0.

All instructions can be entered from line assembly mode. The instructions are converted to machine code and are loaded into memory at the address specified in the prompt. See Section 7, ASM, for information on how to use the line assembler.
Modify Program: Memory Mode

The assembler directives are:

- `symbol'`: Print value of symbol
- `<return>`: Disassemble one instruction
- `\`: Display current assembler offset address
- `CSEG`: Set 64K byte code segment.
- `DB`: Define constant byte data
- `DW`: Define constant word data
- `END`: Exit line assembler
- `EQU`: Define local symbol
- `FAR`: Outside current line assembly segment
- `L0-L9`: Print value of local symbol
- `NEAR`: Within current line assembly segment
- `ORG`: Set 64K byte offset into code segment window.
- `PRE`: Toggle preview mode
- `X`: Exit line assembler

Memory Mode

If you need to modify data space, memory mode is convenient. It allows you to view and modify memory using a simple scrolling scheme. Enter memory mode by executing the M command. The current address and associated data are displayed. If the first character entered on a memory mode command line is a `<return>`, the next address and its data are displayed. If a value is entered before the `<return>`, that value is written to the current address before displaying the next address. A list of up to nine values separated by commas may be entered after a memory mode prompt. This data is stored to consecutive addresses.

The scroll direction is determined by two commands, `NXT` and `LST`. `NXT` (next) increments the address and `LST` (last) decrements the address. Entering either of these commands during run or pause mode sets the scroll direction 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 (.) and comma (,) keys. A period increments the address; a comma decrements it.

The MMP register (Memory Mode Pointer) is always set to the current address being accessed. If memory mode is entered without specifying an address, the value in this register specifies the starting address. On power-up, MMP is set to zero.

The @ command is a shorthand command for reading and writing to memory. It uses the default data length. See page 7-2 for a description of the @ command.
I/O Mode

I/O mode allows viewing and modification of the data in I/O address space. I/O mode is entered with the MIO command. Data is not automatically read from an I/O address on entry to I/O mode. Many I/O ports are write only ports, and trying to read from them may cause hardware problems. In order to read data from an I/O port, you must enter a <return> as the only character on the line. The data is displayed, but the address is not automatically incremented. You must manually change the address while in I/O mode using the period and comma keys. A period (.) increments the address and a comma (,) 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.

IOP

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. (See IOP in Section 7).
Shortcuts

There are many shortcuts to shorten your setup time and reduce the number of keystrokes you must use. They include:

- Using symbols rather than hex addresses.
- Repeating a command indefinitely or a specified number of times.
- Creating and storing macros to use for common command sequences.
- Using general purpose emulator registers for common addresses or data values.
- Saving setup information to ES 1800 EEPROM and reloading it later for one or two users.
- Using clear commands for registers, memory maps, macros and symbols.

Commands Used in Shortcuts

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>'</code></td>
<td>Define symbol or section</td>
</tr>
<tr>
<td><code>DEL</code></td>
<td>Delete symbol or section</td>
</tr>
<tr>
<td><code>PUR</code></td>
<td>Clear all symbols and sections</td>
</tr>
<tr>
<td><code>SEC</code></td>
<td>Display all sections</td>
</tr>
<tr>
<td><code>SYM</code></td>
<td>Display all symbols</td>
</tr>
<tr>
<td><code>/</code></td>
<td>Repeat last command line (no &lt;return&gt;)</td>
</tr>
<tr>
<td><code>*</code></td>
<td>Repeat operator</td>
</tr>
<tr>
<td><code>&lt;ctrl-z&gt;</code></td>
<td>Reset emulator (terminates repeat)</td>
</tr>
<tr>
<td><code>IDX</code></td>
<td>Counter register (can be used to terminate repeat)</td>
</tr>
<tr>
<td><code>LIM</code></td>
<td>Limit register (can be used to terminate repeat)</td>
</tr>
<tr>
<td><code>TST</code></td>
<td>Test variable (can be used to terminate repeat)</td>
</tr>
<tr>
<td><code>CMC</code></td>
<td>Define macros</td>
</tr>
<tr>
<td><code>MAC</code></td>
<td>Clear macros</td>
</tr>
<tr>
<td><code>MAC</code></td>
<td>View macros</td>
</tr>
</tbody>
</table>
### Commands Used in Shortcuts (cont)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General Purpose Register Commands</strong></td>
<td></td>
</tr>
<tr>
<td>BAS</td>
<td>Set/display register default base</td>
</tr>
<tr>
<td>DFB</td>
<td>Display default base</td>
</tr>
<tr>
<td>GD0-7</td>
<td>General purpose data registers</td>
</tr>
<tr>
<td>GR0-7</td>
<td>General purpose address registers</td>
</tr>
<tr>
<td><strong>Saving and Loading Setup Commands</strong></td>
<td></td>
</tr>
<tr>
<td>LD</td>
<td>Load setup from EEPROM</td>
</tr>
<tr>
<td>SAV</td>
<td>Save setup to EEPROM</td>
</tr>
<tr>
<td>SET</td>
<td>Determine configuration for two users</td>
</tr>
<tr>
<td><strong>Clear Commands</strong></td>
<td></td>
</tr>
<tr>
<td>CES</td>
<td>Clear When/Then statements</td>
</tr>
<tr>
<td>CLM</td>
<td>Clear memory map</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear CPU registers</td>
</tr>
<tr>
<td>CMC</td>
<td>Clear macros</td>
</tr>
<tr>
<td>DEL</td>
<td>Delete section or symbol</td>
</tr>
<tr>
<td>OFF-1</td>
<td>Set all on/off switches to off</td>
</tr>
<tr>
<td>PUR</td>
<td>Delete all symbols and sections</td>
</tr>
<tr>
<td><strong>Miscellaneous Useful Commands</strong></td>
<td></td>
</tr>
<tr>
<td>REV</td>
<td>Display revision level</td>
</tr>
</tbody>
</table>

### Use Symbols Rather than Addresses

Symbol definitions allow you to refer to addresses or data values using names rather than numbers. Section definitions allow you to refer to a range of addresses and data values using names rather than addresses. Symbols and sections are sometimes collectively referred to as symbols.

Symbols are 32-bit integer values and sections are 32-bit ranges. To determine approximately how many symbols you can define, use this formula:

\[
\frac{(64 \times 1024)}{(\text{average symbol name length} + 6)}
\]

Symbols are not typed within the ES 1800, so all symbols are global. This implies that a symbol and a section may not be defined using the same name. Duplicate symbol names are not allowed. 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, you can download the symbol table using the **DNL** command. If your linker produces another type of object module format, you must either use a format converter to convert to Extended Tekhex, or use ES Driver. ES Driver accepts a variety of object module formats. See Appendix B.

If you are going to download sections that have already been defined (perhaps from a previous download of the same file), purge all symbols or delete the section definitions from memory before downloading. If you do not, an error occurs when you attempt to redefine the value of a section, and the download aborts.

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

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

First, define the symbols and sections:

```
>SYM $00000480 csr
$00000486 sh_csr
$00001000 CMND
$00001022 Tauc
$00000004 busy
$00000002 got_it
$00000080 action
$00004020 es10
>SEC
$00001000 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  C60680402 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
```

Preparing for Emulation
The following example shows trace disassembly with symbol definitions.

```plaintext
>DTB
>PARTIAL T.M. MAP: PASS 1 PASS 2
FULL T.M. MAP: PASS 1 PASS 2
```

<table>
<thead>
<tr>
<th>SEC</th>
<th>ADDR</th>
<th>OPCODE</th>
<th>MNEMONIC</th>
<th>OPERAND</th>
<th>FIELDS</th>
<th>BUS</th>
<th>CYCLE</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>monitor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0038 +CMND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0038+0000</td>
<td>F70680048000</td>
<td>TEST</td>
<td>WORD PTR csr,0080</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0034+0006</td>
<td>74F8</td>
<td>JE</td>
<td>SHORT CMD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0033+0008</td>
<td>C606800402</td>
<td>MOV</td>
<td>BYTE PTR csr,02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0031+000D</td>
<td>C606860402</td>
<td>MOV</td>
<td>BYTE PTR sh_csr,02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0027+0012</td>
<td>A02040</td>
<td>MOV</td>
<td>AL,BYTE PTR es10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0026+0015</td>
<td>800E860404</td>
<td>OR</td>
<td>BYTE PTR sh_csr,04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0021+001A</td>
<td>8A268604</td>
<td>MOV</td>
<td>AH,BYTE PTR sh_csr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0018+001E</td>
<td>88268004</td>
<td>MOV</td>
<td>BYTE PTR csr,AH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0014+Tauc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>014+0022</td>
<td>F70680048000</td>
<td>TEST</td>
<td>WORD PTR csr,0080</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010+0028</td>
<td>75F8</td>
<td>JNE</td>
<td>SHORT Tauc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0008+002A</td>
<td>EBD4</td>
<td>JMP</td>
<td>SHORT CMD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0005+CMND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0005+0000</td>
<td>F706</td>
<td>TEST</td>
<td>WORD PTR 0000,06F7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following example shows trace disassembly without section definitions.

```plaintext
>DEL 'monitor;DTB
FULL T.M. MAP: PASS 1 PASS 2
```

<table>
<thead>
<tr>
<th>SEC</th>
<th>ADDR</th>
<th>OPCODE</th>
<th>MNEMONIC</th>
<th>OPERAND</th>
<th>FIELDS</th>
<th>BUS</th>
<th>CYCLE</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0038</td>
<td>10000</td>
<td>F70680048000</td>
<td>TEST</td>
<td>WORD PTR csr,0080</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0034</td>
<td>1006</td>
<td>74F8</td>
<td>JE</td>
<td>SHORT CMD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0033</td>
<td>1008</td>
<td>C606800402</td>
<td>MOV</td>
<td>BYTE PTR csr,02</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0031</td>
<td>100D</td>
<td>C606860402</td>
<td>MOV</td>
<td>BYTE PTR sh_csr,02</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0027</td>
<td>1012</td>
<td>A02040</td>
<td>MOV</td>
<td>AL,BYTE PTR es10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0026</td>
<td>1015</td>
<td>800E860404</td>
<td>OR</td>
<td>BYTE PTR sh_csr,04</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0021</td>
<td>101A</td>
<td>8A268604</td>
<td>MOV</td>
<td>AH,BYTE PTR sh_csr</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0018</td>
<td>101E</td>
<td>88268004</td>
<td>MOV</td>
<td>BYTE PTR csr,AH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0014</td>
<td>Tauc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0014</td>
<td>1022</td>
<td>F70680048000</td>
<td>TEST</td>
<td>WORD PTR csr,0080</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Preparing for Emulation

4-59
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
SE# ADDR OPCODE MNEMONIC OPERAND FIELDS BUS CYCLE DATA

0038 1000 F70680048000 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
```
Repeat Operators

The command repeat feature provides a way to repeat a command line a specified number of times or indefinitely.

/ Repeat the last command one time. No <return> is necessary.

* [n] Repeat the last command n times. If no number is specified, repeat command indefinitely. If n=0, * does not cause the command to be repeated.

In these three equivalent examples, the STP;DT command is repeated five times.

> *5 STP;DT
> *5 STP;DT
> * 5 STP;DT

If the slash key is typed after one of the above examples is input, the entire line is repeated, causing five more STP;DT commands to be executed.

There are four rules for using the repeat operators:

1. Repeat commands must be the first character on a line.
2. The repeat argument must be entered as a number. The number will be interpreted as a decimal value. Do not enter a base prefix before entering the repeat value.
   When no repeat argument is specified, it is assumed to be 4,294,967,295(2^32 - 1).
3. You cannot use a register, variable or symbol as the repeat argument.
4. There must be a space following the repeat count if the next character is a decimal digit.

You can always use the system reset character to stop the repeat if the specified test conditions are never reached. However, this will also abort emulation, if it is in progress, without saving the state of the CPU.

The TST variable terminates a repeat when it becomes zero. It is used in an expression on the command line. It is tested just before the command line is executed and if it has become zero, the command buffer is not executed and the repeat halts.

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

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

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

> *STP;DT; TST=087020

The TST variable is set to all 1’s 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
```

<ctrl-z> stops the repeat early.

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

```
> BYM; M $1000
  $001000 $34  >*4 LIM-IDX-1

Old data in memory.
  $001001 $C0
  $001002 $BF
  $001003 $00
  $001004 $21  >M MMP-4
  $001000 $03  >*4
  $001001 $02  New data written to
  $001002 $01  memory with repeat
  $001003 $00  command
  $001004 $21  >
```

**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 using the underscore (_I). Macros are referred to by the decimal numbers #0-9.

Macros can be saved in the ES 1800 EEPROM with the SAV 5 command, and reloaded using the LD 5 command.

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.

Once the buffer is full, attempting to add a macro with a higher numbers will result in those macros remaining null. For example, 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.
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.

**WARNING**

*There are no warnings when truncation or nullification of a macro occurs.*

**General Purpose Registers**

There are two sets of general purpose registers: 8 data registers and 8 general purpose registers. These registers can be used as integer or range arguments to commands to save keystrokes when using values repeatedly. They can also be used to save space in macro definitions.

**Save Setup to EEPROM**

The SET menu, registers, Event Monitor System setup, overlay map, ON/OFF switches and macros can be saved to EEPROM. them with the SAV command. These values may then be automatically loaded into the ES 1800 on power-up by setting the thumbwheel switch to the appropriate value, or loaded manually after power-up by typing a load command (LD).

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:

<table>
<thead>
<tr>
<th>Section #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SET menu</td>
</tr>
<tr>
<td>1</td>
<td>Registers</td>
</tr>
<tr>
<td>2</td>
<td>Event Monitor WHEN/THEN clauses</td>
</tr>
<tr>
<td>3</td>
<td>Overlay map</td>
</tr>
<tr>
<td>4</td>
<td>ON/OFF menu</td>
</tr>
<tr>
<td>5</td>
<td>Macros</td>
</tr>
</tbody>
</table>

**Configure System for Two Users**

In the SET menu, you can specify whether the setup you are doing is for user 0 or user 1. Any configuration changes you make to registers, Event Monitor System setup, overlay map, ON/OFF setup and macros will only apply to whichever user you have specified.
Shortcuts: Clear Commands

This allows you to create two completely different setups. These can be saved in EEPROM between emulation sessions using the SAV command, and reloaded with the LD command. The default is user 0. To save the configuration for user 1:

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

Change to second user
Save configuration
Change back to first user

Clear Commands

There are commands to clear WHEN/THEN statements, I/O map, memory map, CPU registers, macros, symbols and sections, and to set all ON/OFF soft switches to either ON or OFF. These are handy when you want to set your target environment to a known state.

The CES command clears only the WHEN/THEN statements, and leaves the comparators unchanged.

The I/O and memory map clear commands assign all overlay memory the target attribute.

The CLR command clears the CPU registers AX, BX, CX and DX. The segment registers, flags, CS:IP and stack registers remain unchanged.
BRINGING UP HARDWARE

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 three categories:

1. RAM tests
2. Scope loops
3. Miscellaneous special functions

### Commands Used for Diagnostic Functions

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF</td>
<td>Display list of special functions</td>
</tr>
<tr>
<td>SF 0</td>
<td>Simple RAM test, single pass</td>
</tr>
<tr>
<td>SF 1</td>
<td>Complete RAM test, single pass</td>
</tr>
<tr>
<td>SF 2</td>
<td>Simple RAM test, looping</td>
</tr>
<tr>
<td>SF 3</td>
<td>Complete RAM test, looping</td>
</tr>
<tr>
<td>SF 4</td>
<td>Toggle data at address</td>
</tr>
<tr>
<td>SF 5</td>
<td>Peeks into the target system</td>
</tr>
<tr>
<td>SF 6</td>
<td>Pokes into the target system</td>
</tr>
<tr>
<td>SF 7</td>
<td>Write alternate patterns</td>
</tr>
<tr>
<td>SF 8</td>
<td>Write pattern then rotate</td>
</tr>
<tr>
<td>SF 9</td>
<td>Write data then read</td>
</tr>
<tr>
<td>SF 11</td>
<td>Write incrementing value</td>
</tr>
<tr>
<td>SF 12</td>
<td>Read data over an entire range</td>
</tr>
<tr>
<td>SF 13</td>
<td>Cyclic redundancy check</td>
</tr>
<tr>
<td>SF 24</td>
<td>Toggle data at address</td>
</tr>
<tr>
<td>SF 25</td>
<td>Peeks into the target system</td>
</tr>
<tr>
<td>SF 26</td>
<td>Pokes into the target system</td>
</tr>
<tr>
<td>SF 27</td>
<td>Write alternate patterns</td>
</tr>
<tr>
<td>SF 28</td>
<td>Write pattern then rotate</td>
</tr>
<tr>
<td>SF 29</td>
<td>Write data then read</td>
</tr>
<tr>
<td>SF 31</td>
<td>Write incrementing value</td>
</tr>
<tr>
<td>SF 32</td>
<td>Read data over an entire range</td>
</tr>
<tr>
<td>BUS</td>
<td>Display status of bus status lines</td>
</tr>
<tr>
<td>BYM</td>
<td>Set global data length to byte</td>
</tr>
<tr>
<td>CLK</td>
<td>Display target clock frequency</td>
</tr>
<tr>
<td>CRC</td>
<td>Calculate CRC of specified range</td>
</tr>
<tr>
<td>CRE/CRO</td>
<td>Calculate CRC of even/odd bytes only</td>
</tr>
<tr>
<td>WDM</td>
<td>Set global data length to word</td>
</tr>
</tbody>
</table>
Scope Loops

RAM Tests

The RAM tests (SF 0 to SF 3) 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. You must use the BWM or WDM commands to specify byte or word mode before initiating the SF test.

If you are going to test a large section 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 (ON CPY) you can let the test run while you do something else. The printer will record any errors that may occur in your absence.

SF 1 and 3 are modeled after a study by Abraham, Thatte, and Narir titled 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. Reprints are available from the Applied Microsystems Applications Engineering department.

Scope Loops

Scope loops are diagnostic routines for use when troubleshooting with an oscilloscope. Uses include locating stuck address data, status or control lines, and generating signatures using signature analysis equipment.

There are two types of scope loops: memory and I/O. Memory scope loops (SF 4-12) access the memory space defined by the current MMS (Memory Mode Status) register. I/O scope loops (24-32) access the target system's I/O space.

The scope loops are optimized so that they execute at maximum speed. This short cycle time allows you 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 (<ctrl-z> default). The scope loops can be executed in either byte or word mode.
Miscellaneous Special Functions

There are additional special functions for:

1. **CLK**: Reading the target system clock frequency.
2. **CRC, CRE/CRO**: Calculating a cyclic redundancy check on all, or just even or odd addresses in a range.
3. **BUS**: Displaying the status of bus status lines.
4. **BYM, WDM**: Set global data lengths to byte or word.
This section describes what the Time Stamp Module does, and how to install and use the module. Complete examples are provided for using the module to do each possible type of measurement.

The Time Stamp Module adds performance analysis to the ES 1800 Series emulators for 16 bit microprocessors. You can use this module when you use your ES 1800 from a dumb terminal or host computer, or from your host computer using ES Driver control software. Differences in operation for these two configurations are noted where appropriate.

There are two ways the module can be used:

1. To measure elapsed or absolute time.
2. To trigger the Event Monitor System to cause an action such as breaking emulation once a time stamp counter value is reached.

**Commands Used to Set Up Time Stamp**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET #9</td>
<td>Choose timestamp or LSA</td>
</tr>
<tr>
<td>CTS</td>
<td>Convert timestamp value</td>
</tr>
<tr>
<td>WHEN</td>
<td>Event monitor system statements</td>
</tr>
<tr>
<td>MAP</td>
<td>Set memory map</td>
</tr>
<tr>
<td>OVE</td>
<td>Enable overlay memory</td>
</tr>
<tr>
<td>OVS</td>
<td>Overlay memory speed</td>
</tr>
<tr>
<td>VFO</td>
<td>Verify overlay memory</td>
</tr>
</tbody>
</table>

**Possible Measurements**

There are eight distinct measurements that can be made using the Time Stamp Module:

**Elapsed Time Measurements**

- Measure time spent in a module
- Measure time spent between modules
- Measure duration of time when memory is accessed (opcode or data)
- Measure duration of time when code is accessed (opcode only)
- Measure interrupt response time directly
Measurements

Count Occurrences

- Count number of times address or range of memory is accessed (opcode or data)
- Count number of times code is accessed (opcode only)
- Count module linkage activity (the number of times one module calls another)

Each time measurement can be based on one of five scales: .1uS, 1uS, .01mS, .1mS or 1mS, so you can collect your data using the appropriate time scale. The maximum number of counts for any time base is 65,535 so you have a maximum period of 65 seconds without overflow.

Time can be measured on an absolute time frame, or on a relative time frame. When you use the absolute time frame, the measurement is from when the counter is reset. When you use the relative time frame, the measurement is from one traced cycle to the next traced cycle. For example, if you were measuring the elapsed time for entering and exiting a module, the time displays would show as follows:

<table>
<thead>
<tr>
<th></th>
<th>Absolute</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>enter</td>
<td>3000</td>
<td>3000 †</td>
</tr>
<tr>
<td>exit</td>
<td>3005</td>
<td>5</td>
</tr>
<tr>
<td>enter</td>
<td>3007</td>
<td>2</td>
</tr>
<tr>
<td>exit</td>
<td>3012</td>
<td>5</td>
</tr>
<tr>
<td>enter</td>
<td>3014</td>
<td>2</td>
</tr>
<tr>
<td>exit</td>
<td>3019</td>
<td>5</td>
</tr>
</tbody>
</table>

† The first line on the relative trace screen shows the absolute count.

Using the Time Stamp Counter Value as a Condition

The ES 1800 Event Monitor System lets you specify complex program states, using WHEN-THEN statements:

WHEN conditions THEN actions

You can use the absolute value of the time stamp counter as one condition. For more details on using CTS, see the example on page 6-28.
Installation

Hardware Installation

The Time Stamp Module consists of the module and the cable to connect it to the emulator. There are three steps to hardware installation:

1. Turn the emulator off.

CAUTION

*The ES 1800 emulator must be off before plugging in the Time Stamp Module, or the cable and module may be damaged. Do not plug in or unplug the Time Stamp Module with power turned on.*

2. Connect the module to the LSA port on the front of the ES 1800 emulator as shown in the following illustration. Note that you cannot use the Logic State Analysis pod and the Time Stamp Module at the same time.

*Figure 6-1: Connecting the Time Stamp Module to the ES 1800*
3. The Time Stamp Module requires a certain revision of ESL (the Emulator Standard Language). To check your revision:

**ESL command**
Type `REV` from the ES 1800 prompt.

**from ES Driver**
Enter the Target Emulation menu, and type `REV` from the ES 1800 prompt.

If you have an ESL equal to or greater than that shown in the chart below, you can use your Time Stamp Module as is. If your ESL is below the revision shown below, please contact your local sales office or representative, or call the Order Administration department at 800-ASK-4AMC for information on upgrading your ESL revision.

<table>
<thead>
<tr>
<th>Product</th>
<th>Minimum Revision Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>8018X</td>
<td>ESL 3.2</td>
</tr>
<tr>
<td>80C18X</td>
<td>ESL 1.0</td>
</tr>
</tbody>
</table>

**Software Installation**

No software changes are required to operate the Time Stamp Module for any of the following software packages available from Applied Microsystems Corporation.

- ES Driver
- VALIDATE/XEL
- VALIDATE/Soft-Scope
- GeneProbe
Using the Time Stamp Module

This section explains the meaning of the labels, buttons, switches and LEDs on the Time Stamp Module, and then provides complete information on how the unit works.

Figure 6-2: Time Stamp Module

Getting Started

Look at the end of your Time Stamp Module and identify the trigger inputs, reset button, switch and overflow indicator LED as shown in the following diagram.

Figure 6-3: End View of Time Stamp Module
Using the Time Stamp Module

TGR
The TGR input is used to measure interrupt latency directly. You connect the TGR input directly to the interrupt line in your target circuit, avoiding any logic delays due to use of the Event Monitor System. It is designed for processors that pull lines low for interrupts. (Motorola and Zilog processors) (see page 6-17)

TGR
The TGR input is used to measure interrupt latency directly. You connect the TGR input directly to the interrupt line in your target circuit, avoiding any logic delays due to use of the Event Monitor System. It is designed for processors that pull lines high for interrupts. (Intel processors) (see page 6-17)

RST
The reset button is used to reset the time stamp counter to 0.

Switch
The switch is used to determine the time base and the type of counting done. (see page 6-7)

O/F
The overflow LED is lit when the counter overflows the 65,535 limit.

The examples of each type of measurement give complete information on when to use the manual reset button, TGR and TGR, and how to use the switch to choose the time stamp mode and time base.

CAUTION
Do not plug in or unplug the Time Stamp Module when power is turned on to the emulator.

Steps for Using the Time Stamp Module

In order to make a measurement, there are seven steps you must follow:

1. Set the ESL SET flag 9 to the appropriate position for the measurement you want to make.
2. Choose a switch setting on the Time Stamp Module.
3. Set up your trigger inputs.
4. Set up the Event Monitor System to trigger the Time Stamp Module at the appropriate program states.
5. Run your program.
6. View the time stamp information.
7. Interpret the time stamp information.

Each step is described in detail below.
Using the Time Stamp Module

Step 1: Set ESL Flag 9

ESL flag 9 controls the LSA display of information coming in on the LSA port. Settings 1 and 2 are used with the Time Stamp Module. Setting 0 is used when you use the LSA pod.

0 Default: LSA value shown as 16 bits
1 Display the absolute time value
2 Display the relative time value

Absolute time values are used when you want to measure the total amount of time spent or the number of occurrences. Relative time values are used when you are interested in the time spent between points A and B in your code, but are not interested in how long it takes to get to point A.

To get to ESL flag 9:

ESL commands
Type SET 9, n, where n is 0, 1 or 2.

from ES Driver
Select Target Emulation mode, and type SET 9, n, where n is 0, 1 or 2.

Step 2. Set Time Stamp Module Switch

Choose a switch setting on your Time Stamp Module based on your measurement type and preferred time base. We recommend starting with the slowest time frame: 1 mS. The table below shows the maximum measurable time period for each switch setting.

<table>
<thead>
<tr>
<th>Time Base</th>
<th>Maximum Measurable Time Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1 uS</td>
<td>6.5 milliseconds</td>
</tr>
<tr>
<td>1.0 uS</td>
<td>65 milliseconds</td>
</tr>
<tr>
<td>.01 mS</td>
<td>.65 second</td>
</tr>
<tr>
<td>0.1 mS</td>
<td>6.5 seconds</td>
</tr>
<tr>
<td>1.0 mS</td>
<td>65 seconds</td>
</tr>
</tbody>
</table>

IMPORTANT

If the counter overflows, the yellow overflow LED will be lit. Check to see if you are using the correct time base for the duration of your measurements. When the counter overflows the 65,355 limit, it starts again at 0.

When the emulator is paused, no TGR is generated by the Event Monitor System in positions 0-4, so the counter is not reset and is likely to overflow. This is not a problem.
Using the Time Stamp Module

For example, the DRT display might be as follows. The highlighted counter value in the last line of the example shows the counter overflow.

<table>
<thead>
<tr>
<th>LINE</th>
<th>ADDRESS</th>
<th>DATA</th>
<th>R/W</th>
<th>M/IO</th>
<th>BCYC</th>
<th>QUE</th>
<th>ABS TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>#20</td>
<td>000344</td>
<td>&gt;</td>
<td>E2FD</td>
<td>R</td>
<td>TAR</td>
<td>M</td>
<td>IF 2     #63590</td>
</tr>
<tr>
<td>#19</td>
<td>000346</td>
<td>&gt;</td>
<td>80F9</td>
<td>R</td>
<td>TAR</td>
<td>M</td>
<td>IF 2     #64592</td>
</tr>
<tr>
<td>#18</td>
<td>000342</td>
<td>&gt;</td>
<td>754B</td>
<td>R</td>
<td>TAR</td>
<td>M</td>
<td>IF F 3   #65032</td>
</tr>
<tr>
<td>#17</td>
<td>000344</td>
<td>&gt;</td>
<td>E2FD</td>
<td>R</td>
<td>TAR</td>
<td>M</td>
<td>IF 2     #01222</td>
</tr>
</tbody>
</table>

The following table summarizes the switch positions.

The trigger to start and stop the counter in the Time Stamp Module is either the TGR signal from the Event Monitor System (Step 4), or the TGR or TGR direct input from your target interrupt line (Step 3).

<table>
<thead>
<tr>
<th>Position</th>
<th>Time Base</th>
<th>Effect of TGR on Time Stamp Counter</th>
<th>Useful Measurements</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.1 uS</td>
<td>Any TGR high causes the time stamp</td>
<td>Elapsed time</td>
</tr>
<tr>
<td>1</td>
<td>1 uS</td>
<td>counter to be reset to 0. No manual</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>.01 mS</td>
<td>reset is required in this mode for</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>either absolute or relative time</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>.1 mS</td>
<td>stamping.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1 mS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>.1 uS</td>
<td>While the TGR is held high by the</td>
<td>Elapsed time</td>
</tr>
<tr>
<td>6</td>
<td>1 uS</td>
<td>Event Monitor System, the time stamp</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>.01 mS</td>
<td>counter counts. Manual reset is</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>required in this mode for absolute</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>time stamping, but not for relative</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>.1 mS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1 mS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>.1 uS</td>
<td>In this mode, a long TGR signal(^1) from</td>
<td>Elapsed time</td>
</tr>
<tr>
<td>B</td>
<td>1 uS</td>
<td>the Event Monitor System resets the</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>.01 mS</td>
<td>counter. After that, successive short TGR</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>.1 mS</td>
<td>signals turn the counter on and off.</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>1 mS</td>
<td>Manual reset stops the counter and</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>n.a.</td>
<td>sets it to zero.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>This setting is used to count</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>occurrences. Each time the TGR signal</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>goes high, the time stamp counter</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>is incremented. Manual reset is</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>required.</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\) A long TGR is defined as being longer than 1.6 uS. This is the only mode where the length of the TGR matters. The following diagram shows what happens to the counter depending on the TGR signal.
Using the Time Stamp Module

Figure 6-4: Positions A-E: Effects of Multiple TGR Signals

Step 3. Set Up TGR Input

The counter in the Time Stamp Module can be controlled in one of three ways:

1. The Event Monitor System TGR action.
2. The TGR input.
3. The TGR input.

The default is the Event Monitor System trigger input. No additional wires are necessary.

To use the TGR and TGR lines to measure interrupt latency, you must connect one of these lines to an interrupt line on your target. Use of the TGR and TGR external inputs is described fully in the example on page 6-20.

Step 4. Set up the Event Monitor System

In this step, you set up the Event Monitor System to selectively trace the memory, program activity, or modules you are interested in time stamping. Setting up the Event Monitor System can be done through ESL or through the Target Emulation menu in ES Driver.

There are three steps to setting up the Event Monitor System:

1. Decide what condition you want to look at, and what actions to take when that condition is reached.
2. Set up the comparators to isolate that condition.
3. Set up WHEN/THEN statements using the appropriate conditions and actions.

For more information on using the Event Monitor System, please see Section 4 of this manual. The examples beginning on page 6-14 provide examples of using the Event Monitor System to specify conditions appropriate for time stamping.
Using the Time Stamp Module

Step 5. Run your Program

*ESL commands* Run the program using the **RUN** command, or run to a breakpoint using **RBK**.

*from ES Driver* Select the Target Emulation menu, and the Run or Run-to-Breakpoint command.

Step 6. View Time Stamp Information

There are several ways to display the time stamp information.

*ESL commands* The first step is to display the trace by either:
- stopping emulation with the **STP** command
- using the Event Monitor System to break emulation
- if you have Dynamic Trace available, you can use the **OFF TCE** command to view the trace while your program is still running

Then view the trace, using the **DRT** command. The last column shows the absolute or relative time stamp, depending on the position you specified with the **SET** command.

*from ES Driver* Enter the Target Emulation menu, and do the same commands as listed in stand-alone mode.

Step 7. Interpret Time Stamp Information

The time stamp information is always given as a number of units: the units are the ones you specify when you set the switch on the Time Stamp Module.

**IMPORTANT**

You must multiply this number by the time base you selected on the Time Stamp Module switch in order to determine the elapsed time in seconds.
Collecting Time Stamp Information in a File

After setting up your Event Monitor System and Time Stamp Module to provide just the information you need, you can use ES Driver to save the specific DRT displays to an ASCII file. Once the information is stored in the file, you can use a spreadsheet or data base management program to analyze the data.

While in Target Emulation mode,

1. Press <F3> to open a file to save the session record in. You will be prompted to enter a file name. The default extension for this file is .rec.
2. Run the DRT command to print the trace. It will appear on the screen, and also be stored in the file. Note the prompt on the bottom of the screen "SAVE file .rec <F8>=close."
3. Press <F8> to close the session record file.
Examples

There are two basic measurement modes: Elapsed Time and Counting Occurrences. The examples are organized as follows:

Measuring elapsed time
- measuring the time it takes to go from event A to event B
- measuring the time the program is in the specified range
- measuring the time between an interrupt and interrupt servicing

Counting occurrences
- counting the number of times the program transitions from event A to event B
- counting the number of accesses to a memory location or range

Measuring Elapsed Time

The elapsed time measurement can be used to measure in-module time, out-of-module time, inter-module time, and memory and program access time. These measurements use switch positions 0 to E.

Conceptually, there are three types of elapsed time measurements:
1. Measuring the time from event A to event B
   - used for measuring program time, out-of-module execution time, and inter-module execution time
2. Measuring the time spent in an address range
   - used for measuring memory time and program time (excluding calls to other modules)
3. Measuring the time between an interrupt and interrupt servicing
   - used for measuring interrupt latency

A to B Mode

To measure the time it takes a program to get from event A to event B, the easiest way is to set up the Event Monitor System so only event B appears in the trace display.

Step 1. Set LSA Display Type

SET 9, 1
Set display format to absolute time stamp

Step 2. Select Time Stamp Module Switch Setting

Use positions 0-4, depending on your preferred time base. In positions 0-4, the TGR from the Event Monitor System resets the time stamp counter to 0.
If you’re not sure which time base to use, use position 4 for the slowest. If the counter overflows, the yellow overflow LED will light. See page 6-9 for a chart of maximum time periods per setting.

Step 3. Set up the Trigger Input

To measure elapsed time, use the Event System Trigger input.

Step 4. Set up the Event Monitor System

\[ \text{AC1 = 'a} \quad \text{Specify address comparator 1 in group 1 to be event A} \]
\[ \text{AC2 = 'b} \quad \text{Specify address comparator 2 in group 1 to be event B} \]

\[ \text{WHEN AC1 THEN TGR} \quad \text{The TGR action resets the time stamp counter to 0 at event A} \]

\[ \text{WHEN AC2 THEN TRC} \quad \text{Trace event B} \]

Step 5. Run your Program

\[ \text{ESL commands} \quad \text{RUN} \quad \text{Run program} \]
\[ \text{from ES Driver} \quad \text{Target Emulation Menu} \quad \text{Run} \]

Step 6. Stop Emulation

\[ \text{ESL commands} \quad \text{STP} \quad \text{Stop emulation} \]

Step 7. View Time Stamp Data

\[ \text{ESL commands} \quad \text{DRT} \quad \text{Display the trace} \]
\[ \text{from ES Driver} \quad \text{Trace Menu:} \quad \text{Display the trace} \]

Step 8. Interpret Time Stamp Information

The last column of the trace display gives you the absolute time stamp information. Note that if event A and B are called more than once, you will get the time between events for each occurrence.

**IMPORTANT**

*You must multiply this number by the time base you selected on the Time Stamp Module switch in order to determine the elapsed time in seconds.*

The following screen shows the raw trace display. Since the Time Stamp Module switch was set to position #1 (1 uSec), the time to go from A to B is shown to be 4 uSec.
Figure 6-5: Sample DRT Screen for Measuring Time from A to B

<table>
<thead>
<tr>
<th>&gt;DRT LINE</th>
<th>ADDRESS</th>
<th>DATA</th>
<th>R/W</th>
<th>M/IO</th>
<th>BCYC</th>
<th>QUE</th>
<th>ABS TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>#20</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#19</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#18</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#17</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#16</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#15</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#14</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#13</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#12</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#11</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#10</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#9</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#8</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#7</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#6</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#5</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#4</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#3</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#2</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#1</td>
<td>00111A</td>
<td>&gt; 9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>#0 BREAK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F 5</td>
</tr>
</tbody>
</table>
Range Mode

In range mode, the trace display will show the amount of time the program is in the specified range.

The manual reset button should be pressed prior to performing this measurement.

Step 1. Set LSA Display Type

SET 9, 1  Set display format to absolute time stamp

Step 2. Select Time Stamp Module Switch Setting

Use positions 5-9, depending on your preferred time base. In these positions, the Event Monitor System TGR enables the counter.

If you’re not sure which time base to use, use position 9 for the slowest. If the counter overflows, the yellow overflow LED will light. See page 6-9 for a chart of maximum time periods per setting.

Step 3. Set up the Trigger Input

To measure elapsed time, use the Event System Trigger input.

Step 4. Set up the Event Monitor System

AC1 = 'range  Specify address comparator 1 in group 1 to be the specified address range
AC1.2 = 'range  Specify address comparator 1 in group 2 to be the specified address range

WHEN AC1 THEN TGR, GRO 2  
While the range is being accessed, enable the counter and go to group 2

WHEN AC1.2 OR NOT AC1.2 THEN TGR  
Keep counter enabled while in group 2

WHEN NOT AC1.2 THEN GRO 1  
Disable counter when not accessing range

Step 5. Run your Program

ESL commands  RUN  Run program
from ES Driver  Target Emulation Menu  Run

Step 6. Stop Emulation

ESL commands  STP  Stop emulation

Step 7. View Time Stamp Data

ESL commands  DRT  Display the trace
from ES Driver  Trace Menu:  Display the trace
Step 8. Interpret Time Stamp Information

The last column of the trace display gives you the amount of time accumulated while the program was in the specified range.

**IMPORTANT**

*You must multiply this number by the time base you selected on the Time Stamp Module switch in order to determine the elapsed time in seconds.*

The following screen shows the raw trace display, for the above example using a range of $1106$ to $1115$. Since the Time Stamp Module switch was set to position #5 (0.1 uSec), the time spent in this range was 3.4 uSec.

*Figure 6-6: Sample DRT Screen for Measuring Time in Range*
Interrupt Latency

To measure the amount of time between when an interrupt is detected and when it is serviced, you must connect your target interrupt line directly to the TGR or \overline{TGR} lines on the Time Stamp Module. As you can see in Figure 6-7, these lines perform exactly the same function as the Event Monitor System TGR signal, but the direct trigger bypasses the delays inherent in going through the additional Event Monitor System logic.

*Figure 6-7: Trigger Input Logic*

There are two external TGR inputs: TGR and \overline{TGR}. The external TGR is used with Motorola and Zilog processors: when the line is pulled low, the interrupt is asserted. The external \overline{TGR} is used with Intel processors: when the line is pulled high, the interrupt is asserted.

Figure 6-8 shows the trigger pattern for the TGR and \overline{TGR} inputs.
**Step 1. Set LSA Display Type**

**SET 9, 1**  
Set display format to absolute time stamp

**Step 2. Select Time Stamp Module Switch Setting**

Use positions 0-4, depending on your preferred time base. In positions 0-4, the TGR from the external TGR, external TGR or Event Monitor System TGR resets the time stamp counter to 0.

If you’re not sure which time base to use, use position 4 for the slowest. If the counter overflows, the yellow overflow LED will light. See page 6-9 for a chart of maximum time periods per setting.

**Step 3. Set up the Trigger Input**

Connect either the TGR or TGR input on the Time Stamp Module to the interrupt line on your target that you want to check. For example, to check the interrupt latency for interrupt INTO on the 80186, use the setup shown in Figure 6-9.
**Interrupt Latency**

Figure 6-9: Target Setup for Measuring Interrupt Latency

![Diagram showing target setup](image)

**Step 4. Set up the Event Monitor System**

\[ \text{AC1 = 'intservice_start} \]

Specify address comparator 1 in group 1 to be the start of the interrupt service routine

**WHEN AC1 THEN TRC**

Start tracing at the beginning of the interrupt service routine

**Step 5. Run your Program**

**ESL commands**

- **RUN** Run program
- **from ES Driver** Target Emulation Menu Run

**Step 6. Stop Emulation**

**ESL commands**

- **STP** Stop emulation

**Step 7. View Time Stamp Data**

**ESL commands**

- **DRT** Display the trace
- **from ES Driver** Trace Menu: Display the trace

**Step 8. Interpret Time Stamp Information**

The Event Monitor System traces the first cycle of the interrupt service routine. The last column of the trace display shows the amount of time elapsed between the start of the interrupt service routine and the actual interrupt processing.

**IMPORTANT**

You must multiply this number by the time base you selected on the Time Stamp Module switch in order to determine the elapsed time in seconds.
COUNTING OCCURRENCES: A to B

Counting Occurrences

The number of occurrences measurement can be used to measure memory and program activity, module linkage activity and program flow activity. Use switch position F (count TGR pulses) for all counting measurements.

Conceptually, there are two types of counting occurrences measurements:

1. Counting the number of times the program transitions from event A to event B
   - used for measuring module linkage activity
2. Counting the number of accesses to some memory location(s).
   - used for measuring memory program activity

A to B Mode

This mode records the number of times the transition from event A to event B occurs. Trace is only recorded on exit from module B. The manual reset button should be pressed prior to performing this measurement.

Step 1. Set LSA Display Type

SET 9, 1 Set display format to absolute time stamp

Step 2. Select Time Stamp Module Switch Setting

Use position F. For counting occurrences, the time base is irrelevant. In position F, when the TGR from the Event Monitor System goes high, the time stamp counter increments.

Step 3. Set up the Trigger Input

To count occurrences, use the Event System Trigger input.

Step 4. Set up the Event Monitor System

AC1.1 = 'start-a Specify address comparator 1 in group 1 to be the start of module A
AC1.2 = 'start-b Specify address comparator 1 in group 2 to be the start of module B
AC2.2 = 'end-b Specify address comparator 2 in group 2 to be the end of module B

WHEN AC1 THEN GRO 2
Go to group 2 while in module A

WHEN AC1.2 THEN TGR
Increment counter when entering module B from module A

WHEN AC2.2 THEN TRC, GRO 1
Exit module A, record count in trace memory
Step 5. Run your Program

**ESL commands**

- **RUN** Run program
- **Target Emulation Menu** Run

Step 6. Stop Emulation

**ESL commands**

- **STP** Stop emulation

Step 7. View Time Stamp Data

**ESL commands**

- **DRT** Display the trace

**from ES Driver**

- **Trace Menu:** Display the trace

Step 8. Interpret Time Stamp Information

The last column gives you the number of times module B was entered from module A. Note that only the locations 'start-a (1100) and 'end-b (2008) are traced. In the following screen we see that module B is called once each time from module A. The total number of calls is 16.

Figure 6-10: Sample DRT Screen for Counting Occurrences

<table>
<thead>
<tr>
<th>LINE</th>
<th>ADDRESS</th>
<th>DATA</th>
<th>R/W</th>
<th>M/ID</th>
<th>BCYC</th>
<th>QUE</th>
<th>ABS TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>#20</td>
<td>001100</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>F 4  #6</td>
</tr>
<tr>
<td>#19</td>
<td>002008</td>
<td>C3C3</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>4  #7</td>
</tr>
<tr>
<td>#18</td>
<td>001100</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>F 4  #7</td>
</tr>
<tr>
<td>#17</td>
<td>002008</td>
<td>C3C3</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>4  #8</td>
</tr>
<tr>
<td>#16</td>
<td>001100</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>F 4  #8</td>
</tr>
<tr>
<td>#15</td>
<td>002008</td>
<td>C3C3</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>4  #9</td>
</tr>
<tr>
<td>#14</td>
<td>001100</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>F 4  #9</td>
</tr>
<tr>
<td>#13</td>
<td>002008</td>
<td>C3C3</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>4  #10</td>
</tr>
<tr>
<td>#12</td>
<td>001100</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>F 4  #10</td>
</tr>
<tr>
<td>#11</td>
<td>002008</td>
<td>C3C3</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>4  #11</td>
</tr>
<tr>
<td>#10</td>
<td>001100</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>F 4  #11</td>
</tr>
<tr>
<td>#9</td>
<td>002008</td>
<td>C3C3</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>4  #12</td>
</tr>
<tr>
<td>#8</td>
<td>001100</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>F 4  #12</td>
</tr>
<tr>
<td>#7</td>
<td>002008</td>
<td>C3C3</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>4  #13</td>
</tr>
<tr>
<td>#6</td>
<td>001100</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>F 4  #13</td>
</tr>
<tr>
<td>#5</td>
<td>002008</td>
<td>C3C3</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>4  #14</td>
</tr>
<tr>
<td>#4</td>
<td>001100</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>F 4  #14</td>
</tr>
<tr>
<td>#3</td>
<td>002008</td>
<td>C3C3</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>4  #15</td>
</tr>
<tr>
<td>#2</td>
<td>001100</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>F 4  #15</td>
</tr>
<tr>
<td>#1</td>
<td>002008</td>
<td>C3C3</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>4  #16</td>
</tr>
<tr>
<td>#0</td>
<td>BREAK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F   1</td>
<td></td>
</tr>
</tbody>
</table>

Time Stamp Module
COUNTING OCCURRENCES: Range Mode

Range Mode

This mode records the number of accesses to some memory location(s). Trace is always recorded. The last trace cycles recorded show the accumulated access counts. The manual reset button should be pressed prior to performing this measurement.

Step 1. Set LSA Display Type

SET 9, 1
Set display format to absolute time stamp

Step 2. Select Time Stamp Module Switch Setting

Use position F. For counting occurrences, the time base is irrelevant. In this position, when the TGR from the Event Monitor System goes high, the time stamp counter increments.

Step 3. Set up the Trigger Input

To count accesses, use the Event System Trigger input.

Step 4. Set up the Event Monitor System

AC1.1 = 'here TO 'there
Specify the range to be monitored

WHEN AC1 THEN TGR
Increment counter whenever range is accessed

Step 5. Run your Program

ESL commands
RUN
Run program
from ES Driver
Target Emulation Menu Run

Step 6. Stop Emulation

ESL commands
STP
Stop emulation

Step 7. View Time Stamp Data

ESL commands
DRT
Display the trace
from ES Driver
Trace Menu Display the trace

Step 8. Interpret Time Stamp Information

The last column of the last line of the trace display gives you the number of times the range was accessed. In the following sample screen, the range is set from $2000 to $2006.
Figure 6-11: Sample DRT Screen Counting Occurrences in a Range

<table>
<thead>
<tr>
<th>DRT</th>
<th>ADDRESS</th>
<th>DATA</th>
<th>R/W</th>
<th>M/IO</th>
<th>CYC</th>
<th>QUE</th>
<th>ABS</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>#20</td>
<td>001108</td>
<td>&gt;9090</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>IF</td>
<td>4</td>
<td>#30</td>
</tr>
<tr>
<td>#19</td>
<td>00110A</td>
<td>&gt;F3E8</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
<td>#30</td>
</tr>
<tr>
<td>#18</td>
<td>00110C</td>
<td>&gt;900E</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
<td>#30</td>
</tr>
<tr>
<td>#17</td>
<td>00110E</td>
<td>&gt;9090</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
<td>#30</td>
</tr>
<tr>
<td>#16</td>
<td>002000</td>
<td>&gt;9090</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>IF</td>
<td>F</td>
<td>#30</td>
</tr>
<tr>
<td>#15</td>
<td>003FFE</td>
<td>&lt;110D</td>
<td>W</td>
<td>OYL</td>
<td>M</td>
<td>WM</td>
<td>2</td>
<td>#30</td>
</tr>
<tr>
<td>#14</td>
<td>002002</td>
<td>&gt;9090</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>IF</td>
<td>1</td>
<td>#31</td>
</tr>
<tr>
<td>#13</td>
<td>002004</td>
<td>&gt;9090</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>IF</td>
<td>2</td>
<td>#31</td>
</tr>
<tr>
<td>#12</td>
<td>002006</td>
<td>&gt;9090</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>IF</td>
<td>3</td>
<td>#31</td>
</tr>
<tr>
<td>#11</td>
<td>002008</td>
<td>&gt;C3C3</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>IF</td>
<td>4</td>
<td>#31</td>
</tr>
<tr>
<td>#10</td>
<td>00200A</td>
<td>&gt;ED17</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>IF</td>
<td>4</td>
<td>#31</td>
</tr>
<tr>
<td>#9</td>
<td>00200C</td>
<td>&gt;0040</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
<td>#32</td>
</tr>
<tr>
<td>#8</td>
<td>003FFE</td>
<td>&gt;110D</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>RM</td>
<td>F</td>
<td>#32</td>
</tr>
<tr>
<td>#7</td>
<td>00110D</td>
<td>&gt;90</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>IF</td>
<td>F</td>
<td>#32</td>
</tr>
<tr>
<td>#6</td>
<td>00110E</td>
<td>&gt;9090</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>IF</td>
<td>1</td>
<td>#32</td>
</tr>
<tr>
<td>#5</td>
<td>001110</td>
<td>&gt;9090</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>IF</td>
<td>2</td>
<td>#32</td>
</tr>
<tr>
<td>#4</td>
<td>001112</td>
<td>&gt;9090</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>IF</td>
<td>3</td>
<td>#32</td>
</tr>
<tr>
<td>#3</td>
<td>001114</td>
<td>&gt;9090</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>IF</td>
<td>4</td>
<td>#32</td>
</tr>
<tr>
<td>#2</td>
<td>001116</td>
<td>&gt;9090</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>IF</td>
<td>4</td>
<td>#32</td>
</tr>
<tr>
<td>#1</td>
<td>001118</td>
<td>&gt;9090</td>
<td>R</td>
<td>OYL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
<td>#32</td>
</tr>
<tr>
<td>#0</td>
<td>BREAK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
</tr>
</tbody>
</table>

Time Stamp Module
Using the Time Stamp Counter Value as a Condition

The ES 1800 Event Monitor System lets you specify complex program states, using WHEN-THEN statements:

WHEN conditions THEN actions

You can use the absolute value of the time stamp counter as one condition.

Conditions are defined as logical combinations of address, data and status comparators. The comparator LSA reads the value of the time stamp counter.

Due to the sequencing of the bit information from the Time Stamp Module, the count value needs to be converted to the same format used by the ES 1800, using the CTS (convert time stamp) command.

Sample Situation:
Suppose you want to break 2 seconds after reaching a specified address. If the pod is set to the 1 millisecond setting, this is 2000 counts. It would make sense to say 'LSA=#2000' as the Event Monitor System condition, but as we’ve explained above, this value must be converted.

Step 1. Set LSA Display Type
SET 9, 1 Set display format to absolute time stamp

Step 2. Select Time Stamp Module Switch Setting
Use position 4 to count every millisecond. In this position, the TGR from the Event Monitor System resets the counter.

Step 3. Set up the Trigger Input
To measure elapsed time, use the Event System Trigger input.

Step 4. Convert Value
CTS #2000 Convert time stamp value for ES 1800. The ES 1800 responds with $0438. This is the value the LSA port actually sees when the pod has counted 2000 times

Step 5. Set up the Event Monitor System
AC1 = address to reset counter Specify the address at which to reset the counter

WHEN AC1 THEN TGR,GRO 2 Reset counter and switch to group 2 when AC1 is reached

LSA.2=$0438 Specify the converted time stamp value to break at

2 WHEN LSA THEN BRK Break when counter value is reached.
IMPORTANT

The ES 1800 Event Monitor System samples address, data and status once every processor bus cycle. If the time base is shorter than the bus cycle, then a particular LSA value may be missed by the Event Monitor System.

For most processor systems, a time base of 0.01 mS, 0.1 mS or 1 mS is slow enough to prevent this problem.

Step 6. Stop Emulation

ESL commands  STP  Stop emulation

Step 7. View Time Stamp Data

ESL commands  DRT  Display the trace

from ES Driver  Trace Menu  Display the trace

Step 8. Interpret Time Stamp Information

In this setup, you chose to break when a timestamp count limit was reached. At this point, you could do any of the steps listed in Section 4: Isolating the Problem.

Figure 6-12: Sample DRT Screen After Breaking at Time Stamp Counter Value

<table>
<thead>
<tr>
<th>#</th>
<th>ADDRESS</th>
<th>DATA</th>
<th>R/W</th>
<th>M/IO</th>
<th>BCYC</th>
<th>QUE</th>
<th>ABS TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>001122</td>
<td>C004</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>19</td>
<td>001124</td>
<td>96FB</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>18</td>
<td>001100</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>F 4</td>
</tr>
<tr>
<td>17</td>
<td>001102</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>001104</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>001106</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>4</td>
</tr>
<tr>
<td>14</td>
<td>001108</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>00110A</td>
<td>F3E8</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>00110C</td>
<td>900E</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>11</td>
<td>00110E</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>002000</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>F 4</td>
</tr>
<tr>
<td>9</td>
<td>003FFC</td>
<td>110D</td>
<td>W</td>
<td>OVL</td>
<td>M</td>
<td>MM</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>002002</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>002004</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>002006</td>
<td>9090</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>002008</td>
<td>C3C3</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>00200A</td>
<td>ED17</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>00200C</td>
<td>0040</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>003FFC</td>
<td>110D</td>
<td>R</td>
<td>OVL</td>
<td>RM</td>
<td>F 5</td>
<td>#2000</td>
</tr>
<tr>
<td>1</td>
<td>001110D</td>
<td>90</td>
<td>R</td>
<td>OVL</td>
<td>M</td>
<td>IF</td>
<td>F 0</td>
</tr>
<tr>
<td>0</td>
<td>BREAK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F 1</td>
</tr>
</tbody>
</table>
Introduction

This section contains all the ESL commands, listed in alphabetical order. Commands which begin with non-alphanumeric keys are at the beginning of the section, in the following order:

@  
/  
*  
-  
' < register >

The following syntax is used:

**bold type**  Type the command exactly as printed.

*italic type*  A substitution is required. For example, if you see *file*, you must specify a file name.

< angle brackets >  These indicate mandatory arguments. Do not type the brackets.

[ square brackets ]  These indicate optional arguments. Do not type the square brackets.
@: Read/Write Memory

@: Read/Write Memory

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>@&lt;address&gt;</td>
<td>Read data from memory at &lt;address&gt;.</td>
</tr>
</tbody>
</table>
| @<address>=value | Write value to memory at <address>. No read-
|               | after-write verify occurs.                                            |

Comments

The @ command provides a quick way to read from or write to memory in the target. It functions in much the same way as memory mode, but it is a simple command, rather than an operating mode.

Two system parameters affect the operation of the @ command.

- The default data length determines whether a byte or word access is made. (BYM and WDM)
- The value in the MMS register specifies the memory space accessed.

The @ command will read from or write to the overlay memory if the specified address is mapped. If the address is not mapped, the access will occur in the target system memory.

<address> and <value> may be any valid ESL expression. This means you may use registers, symbol names or numeric values as the address or value.

You may execute this command while in run mode, but if you do, emulation will be halted briefly in order to complete the command. You will not be executing in real-time if you enter @ commands while in run mode.
Examples

@WDM
@0
$00001012
@SS:SP
$00003F01
>

Use the @ command to patch program data.

@DS:DI=102F
@((DS:DI-2)=44E2
@DS:DI;@((DS:DI-2)

$0000102F
$000044E2
>
': Symbol and Section Definition

': Symbol and Section Definition

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>' '&lt;symbol&gt;'</td>
<td>Display value of specified symbol.</td>
</tr>
<tr>
<td>' '&lt;section&gt;'</td>
<td>Display value of specified section.</td>
</tr>
<tr>
<td>' '&lt;symbol&gt;' =&lt;value&gt;'</td>
<td>Assign &lt;value&gt; to the symbol.</td>
</tr>
<tr>
<td>' '&lt;section&gt;' =&lt;range&gt;'</td>
<td>Assign &lt;range&gt; to the section. Section range values cannot overlap.</td>
</tr>
</tbody>
</table>

Comments

A space indicates the end of the symbol or section name. 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.

- `<range>`
  - A 32-bit integer range. Ranges can be specified as follows:
    - `start_address LEN length`
    - `start_address TO end_address`

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 ES 1800 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.
Examples

>'testing =2000  Set symbol to 2000.
>'end_loop =GR0  Set symbol to value in general purpose register 0.
>'section_3 =10000 TO 1FFF  Define section range using start/end syntax.
>'main_loop =prog_start TO 'RAM_START-1  Define section range using symbols for start and end addresses.
>'section_4 =1000 LEN 1F  Define section range using start/length syntax.
/: Repeat Command Line

/: Repeat Command Line

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>Re-execute the previous command line. No &lt;return&gt; is necessary.</td>
</tr>
</tbody>
</table>

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  Single step and disassemble instruction.
/>        Repeat previous command.
/>        "
/>        "
/>        "

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

>STP;DIS CS:IP LEN 10 Single step, then disassemble memory beginning at CS:IP location.
/>        Repeat previous command.
*: Repeat Command Line

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>* [n]</td>
<td>Repeat the last command (n) times. If no number is specified, repeat command indefinitely. If (n=0), * does not cause the command to be repeated. * must be the first character on a line.</td>
</tr>
</tbody>
</table>

**Comments**

You cannot use a register, variable or symbol as the repeat argument. The repeat argument must be entered as a number. The number will be interpreted as a decimal value. Do not enter a base prefix before entering the repeat value. When no repeat argument is specified, it is assumed to be \(4,294,967,295(2^{32} - 1)\).

**Examples**

In these three equivalent examples, the **STP;DT** command is repeated five times.

\[
>*\ 5\text{STP;}DT \\
>*\ 5\text{STP;}DT \\
>*\ 5\text{STP;}DT \\
\]

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

\[
>*\text{STP;}DT;\ \text{TST=}\text{CS:IP}–\$C324 \\
\]

*.: Repeat Command Line
_ : Define/Use Macros

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>_&lt;0-9&gt;=&lt;com, exp, op&gt;</td>
<td>Define the specified macro.</td>
</tr>
<tr>
<td>_&lt;0-9&gt;</td>
<td>Use the specified macro.</td>
</tr>
<tr>
<td>,</td>
<td>Use macro 1. Must be first character on line.</td>
</tr>
<tr>
<td>.</td>
<td>Use macro 2. Must be first character.</td>
</tr>
</tbody>
</table>

**Comments**

When a macro is defined, there is no display on the screen, the syntax is not checked. Macros are expanded when they are executed, not when they are defined. A space between the underscore, digit, or equals sign causes an error.

**Examples**

In this example, four macros are defined. Macros #1 and #2 can be executed independently. Macro #3 contains two nested macros (#1 and #2).

>_1=STP;DT Set macro 1 to single step and display trace.
>_2=GR1=GR1+1 Set macro 2 to increment a general purpose register.
>_3=_1;_2 Set macro 3 to do macro 1, then macro 2.
>_1= DB SS:SP LEN 20;RET;DIS CS:IP LEN 12 Display the first 20B bytes on the stack, skip a line for readability and disassemble the next instructions that will be executed.

In the next example, macros one and three are executed.

>, Execute macro 1. Could also use _1
>_3 Execute macro 3.
ASM: Line Assembler

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASM</td>
<td>Begin assembly at the last address displayed during a previous assembly session. At power-up the start address is zero.</td>
</tr>
<tr>
<td>ASM &lt;arg&gt;</td>
<td>Begin assembly at the specified address.</td>
</tr>
</tbody>
</table>

**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 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 Line Assembler

#### Directive | Result
---|---
CSEG | Set 64K byte code segment window:
- `1012 >CSEG D400H`
- `1012 >`

ORG | Set 64K byte offset into the code segment window:
- `1012 >ORG 3ACH`
- `03AC >`

END or X | Exit line assembler and return to the command level:
- `58FD >X`
- `**** END OF LINE ASSEMBLY ****`
- `>`

DB | Define constant byte data:
- `58FD >DB 1,2,3,4, "TEST", 0`
- `58FD 01 02 03 04 54 53 54 00`
- `5907 >`

DW | Define 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 | Toggle to preview mode (causes next instruction to be disassembled):
- `6590 >PRE`
- `6590 C6470234 MOV BYTE PTR [BX+2H], 34H`
- Toggle out of preview mode:
- `6590 C6470234 MOV BYTE PTR [BX+2H], 34H`
- `>PRE`
- `6590>`

EQU | Define/redefine local symbol (L0-L9):
- `6590 >L3 EQU 7A44H`
- `6590 >`
or if symbolic debug hardware is installed:

```
6590 > 'Unit EQU OFDE0H
```

L0,L1...L9

Print value of local symbol:

```
756A >L3
```

'symbol

Print value of symbol. This is only valid if symbolic debug hardware is installed:

```
756A >'Unit
756A >'Unit EQU FDE0H
```

<return>

Disassemble one line at current address.

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

$

Current assembler offset address.

NEAR

Within current line assembly segment.

FAR

Outside current line assembly segment.
BAS: Set/Display Register Default Base

**Command**

**BAS** <register>

Display 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>

Set 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 ES 1800 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 ES 1800 displays a question mark (?) preceding the base value when asked to display the base in effect.

Refer to the default base command, DFB to display the system global default base.

**Examples**

> BAS FLX

Display default base of FLX register.

> #16

In the next example, the value of general data register GD3 is displayed in binary until you change its display base or power down the ES 1800.

> GD3

Display GD3 using default base.

$0000AA55

> BAS GD3 = 2

Set base of GD3 register.

> BAS GD3

Display new base of GD3 registers.

#2

> GD3

Display register

%000000000000010101001010101
### BKX: Break On Instruction Execution

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ON BKX</strong></td>
<td>The Event Monitor System breaks on the execution of the instruction rather than the instruction prefetch.</td>
</tr>
<tr>
<td><strong>OFF BKX</strong></td>
<td>The Event Monitor System breaks whenever an address is seen on the bus.</td>
</tr>
</tbody>
</table>

**Default:** OFF

**Comments**

The 80186 family microprocessors prefetch 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 ES 1800 may break before the instruction is executed (it was prefetched). Set this switch to force the break to occur only on address execution.
BMO: Block Move

**Command**

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMO&lt;range&gt;,&lt;address&gt;</td>
<td>Moves &lt;range&gt; to the new &lt;address&gt;. The current value of MMS specifies the relocation register used during the transfer.</td>
</tr>
<tr>
<td>BMO&lt;range&gt;,&lt;space&gt;,&lt;address&gt;</td>
<td>Moves &lt;range&gt; to the new &lt;address&gt;. The &lt;space&gt; argument specifies the memory mode status to use during the transfer.</td>
</tr>
<tr>
<td>BMO&lt;range&gt;,&lt;address&gt;,&lt;space&gt;</td>
<td>Moves &lt;range&gt; to the new &lt;address&gt;. The range is read from the space specified in the MMS register. The block is written to &lt;space&gt;.</td>
</tr>
<tr>
<td>BMO&lt;range&gt;,&lt;space&gt;,&lt;address&gt;,&lt;space&gt;</td>
<td>Moves &lt;range&gt; to the new &lt;address&gt;. The range is read from &lt;space&gt; specified in the argument following the range. The block is written to &lt;space&gt; specified in the argument following the address.</td>
</tr>
</tbody>
</table>

**Comments**

This command is valid in pause mode only.

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

1. If there is no space specified for the source argument, MMS is always used.
2. If no space is specified for the destination address, the source space is always used.
3. A non-overlapping block move can be verified using the VBL command.
Examples

The examples show two ways to move a range to a new location in data space, and moving a range from the stack space to data space.

> MMS=DAT
  Set the MMS to data space

> BMO 100 TO 500, 1000
  Move a range to the new location.

or

> BMO 100 to 500, DAT, 1000
  Same effect as two commands above.

> BMO SS:SP LEN 20, STA, DX, DAT
  Move 20 bytes from the stack in stack space to the value pointed to by the data register in data space.
BRK: Break Emulation

**Command**

**Result**

WHE <events> THE BRK, <action>....

If all of the conditions specified in the event portion of the WHEN/THEN clause are satisfied, 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.

**Comments**

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.
Examples

The first example shows breaking when the instruction at address $3000$ is executed.

```plaintext
>ON BKX            Enable breakpoints on instruction execution.
>AC1=3000          Set address comparator to 3000.
>WHEN AC1 THEN BRK Break when AC1 is accessed.
>RBK               Run til breakpoint.
R>                 Run mode prompt will appear.
```

The next example shows tracing a limited range of accesses, and breaking after ten accesses to the range. Trace only accesses between 1000 and 113C; break after ten accesses to this address range.

```plaintext
>AC1=1000 to 113C  Set up range.
>CTL=#10           Set up counter limit.
>WHEN AC1 THEN CNT,TRC Set up WHEN/THEN to trace only accesses in range, and begin counting whenever range is accessed.
>WHEN CTL THEN BRK Break after 10 accesses.
>RBV               Load restart vectors and begin emulation.
R>                 Run mode prompt will appear.
```

The third example shows breaking when a data value is written to a port. Break when 55AA is written to I/O port A.

```plaintext
>AC1='PORT_A       Set address comparator to port address.
>DC1=55AA          Set data comparator to 55AA.
>S1=W10            Set status comparator to Write I/O Status.
>WHEN AC1 AND DC1 AND S1 THEN BRK Set WHEN/THEN statement.
>RBK               Run til breakpoint.
R>                 Run mode prompt will appear.
```
**BUS: Display Status Of Bus Status Lines**

**Command** | **Result**
--- | ---
BUS | Display the bus status.

**Comments**

The status of the following 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**

1 indicates an active condition

0 indicates an inactive condition

**80186/88 and 80C186/C188:**

```
> BUS
NMI ARDY SRDY INTO INT1 INT2/INTA0 INT3/INTA1 TEST
0 1 0 0 0 0 0 0 1 0
> I
```

**80C186EB and 80C188EB:**

```
> BUS
NMI READY INT4 INTO INT1 INT2/INTA0 INT3/INTA1 TEST
0 0 0 0 0 0 0 0 0 0
> 
```
**BTE: Bus Timeout Enable**  
(80C18x and 80C18xEB only)

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON BTE</td>
<td>Enable the bus timeout. Supply RDY after 1 second without target RDY. Force emulation break if in RUN mode.</td>
</tr>
<tr>
<td>OFF BTE</td>
<td>Do not supply RDY, even if target does not. Allows the CPU to wait indefinitely for target RDY. Default: OFF</td>
</tr>
</tbody>
</table>

**Comments**

With BTE set ON, the emulator will automatically time out after waiting for 1 second for the ARDY or SRDY signal to be supplied by the target system, ensuring that the emulator will not hang after attempting an invalid memory location access.

During RUN mode, the emulator will wait one second, then force SRDY to the CPU, then attempt to break emulation.

During peeks and pokes, the emulator will just force SRDY to allow the cycle to complete.

With BTE set to OFF, the emulator will not interfere with target signals. Lack of a target-supplied ARDY or SRDY in this instance will cause the CPU to wait indefinitely.
BTO: Bus Timeout Register
(80C18x and 80C18xEB only)

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTO</td>
<td>Display the value of the bus timeout register</td>
</tr>
<tr>
<td>BTO = value</td>
<td>Assign a value to the bus timeout register. This 8-bit integer value specifies the number of milliseconds to wait before the emulator displays a NO BUS CYCLE error message. Default = 0</td>
</tr>
</tbody>
</table>

Comments

Use this register value to increase the amount of time the processor waits for an address latch enable (ALE). By default, the emulator reports a NO BUS CYCLE error message if no ALE is detected for at least .7 milliseconds.

If your target system uses a HALT instruction and then waits for an interrupt for a longer period than this, be sure to set the BTO register to provide a sufficient length of time.

The BTO register specifies the number of milliseconds to wait before generating the NO BUS CYCLE error message. The value in the register may be any integer value from 0–255 (hexadecimal).

Some operating systems (such as iRMX®) make extensive use of the HALT instruction. Use the BTO register, or the IHE softswitch, to avoid having the NO BUS CYCLE message scroll across your screen.
BYM: Set Global Data Length

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYM</td>
<td>Set the global data length to byte mode.</td>
</tr>
<tr>
<td>WDM</td>
<td>Set the global data length to word mode.</td>
</tr>
<tr>
<td></td>
<td>Default: <strong>BYM</strong> - byte mode</td>
</tr>
</tbody>
</table>

**Comments.**

The global data length determines whether memory commands use byte or word data lengths. If byte mode is set and you enter a word value as a command parameter, only the least significant byte is used as the command parameter. If word mode is set and you enter a byte parameter, the high byte is padded with a zero.

The global data length affects the following commands.

**Commands Affected by Global Data Length**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMO</td>
<td>block move data in memory</td>
</tr>
<tr>
<td>DB</td>
<td>display block of memory</td>
</tr>
<tr>
<td>FIN</td>
<td>find data pattern in memory</td>
</tr>
<tr>
<td>FIL</td>
<td>fill memory with data pattern</td>
</tr>
<tr>
<td>LOV</td>
<td>load overlay memory from target</td>
</tr>
<tr>
<td>M</td>
<td>memory mode</td>
</tr>
<tr>
<td>MIO</td>
<td>I/O mode</td>
</tr>
<tr>
<td>SF 0-9,11,12</td>
<td>special functions: RAM tests and scope loops</td>
</tr>
<tr>
<td>VBL</td>
<td>verify data pattern in memory</td>
</tr>
<tr>
<td>VFO</td>
<td>verify overlay memory with target memory</td>
</tr>
</tbody>
</table>
BYM: Set Global Data Length

Examples
The following example demonstrates how the global data length affects the FIL and DB
commands.
>BYM

Set byte mode

>FIL 0 LEN 10,123

Fill the range with 123

>DB 0 LEN 10

High byte is truncated

000000 23 23 23 23 23 23 23 23 - 23 23 23 23 23 23 23 23 23 23 ################
>

>WDM

Set word mode

>FIL 0 LEN 10,3F

Fill the range with 3F
Pattern is padded with zero

>DB 0 LEN 10
000000

003F

003F

003F

003F - 003F

003F

003F

003F

>

7-22

Alphabetical Command Reference


CCT: Computer Port Control

**Command**

**CCT**

**Result**

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 ES 1800 emulator. This command is meant to be executed from the terminal port, and is essentially a null command when entered from the computer port.

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

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

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 ES 1800. The host system must be able to handle incoming data at high rates. Both hardware and software handshakes are supported (see Section 4, “Serial Communications.”)

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 ES 1800 off and then on.
CDH: Clear DMA Halt
(8018x and 80C18x only)

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON CDH</td>
<td>DMA is re-enabled during pause-to-run.</td>
</tr>
<tr>
<td>OFF CDH</td>
<td>During pause-to-run, DMA status is unchanged from status while paused. Default: OFF</td>
</tr>
</tbody>
</table>

Comments

The ES 1800 transitions from run to pause mode by using a non-maskable interrupt (NMI). An NMI has the effect of setting the DHLT bit (bit 15) of the Interrupt Status Register. When DHLT is true, the processor disables DMA cycles.

DMA cycles will be disabled when the emulator enters the run mode unless the CDH softswitch is in the ON state.

This command is not recognized by the emulator for the 80C186EB or 80C188EB processors.
CES: Clear When/Then Statements

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CES</td>
<td>Clear all of the WHEN/THEN statements currently active within the event monitor system.</td>
</tr>
<tr>
<td>CES &lt;group number&gt;</td>
<td>Clear all of the WHEN/THEN statements for the specified group within the event monitor system.</td>
</tr>
</tbody>
</table>

Comments

The comparator values are not affected by the CES command.
### CK: Internal/External Clock

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ON CK</strong></td>
<td>The CPU uses an internally generated clock. A 4 MHz nonadjustable clock is supplied via a divide-by-two network. The CPU runs at 2 MHz. (The 80C18x CPU clock is set at 12.5 MHz.) Newer revisions run at 16 MHz. Unterminated inputs are set inactive.</td>
</tr>
<tr>
<td><strong>OFF CK</strong></td>
<td>The CPU uses the target system clock. Appendix C contains information on jumper configurations for specific target clock configurations.</td>
</tr>
</tbody>
</table>

**Default:** OFF

**Comments.**

This command is valid only in pause mode.

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

See also the DNL command, the RDY command and Section 4 “Mapping Overlay Memory.”
CLK: Read Target System Clock

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>Read the target system clock frequency and display the value in KHz. The value is accurate to plus or minus 2 KHz.</td>
</tr>
</tbody>
</table>

**Examples**

- `>CLK`  
  Display clock frequency.  
  `CLOCK FREQUENCY = #2001 KHZ`
- `>`
### CLM: Clear Memory Map

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLM</td>
<td>Assign the entire address range the TGT attribute.</td>
</tr>
</tbody>
</table>

#### Comments

This command clears all addresses from the overlay map.

This command is valid only in pause mode.
CLR: Clear CPU Registers

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR</td>
<td>Clear the four CPU data registers; AX, BX, CX, and DX.</td>
</tr>
</tbody>
</table>

Comments

The CPU registers are automatically copied from ES 1800 internal memory to the microprocessor when run mode is entered. When emulation is broken, they are copied from the processor to ES 1800 internal memory.

See DR for more information.
## CMC: Clear Macros

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMC</td>
<td>Clear all defined macros.</td>
</tr>
<tr>
<td>_&lt;0-9&gt;=</td>
<td>Clear the specified macro.</td>
</tr>
</tbody>
</table>

### Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;_1=</td>
<td>Clear macro #1.</td>
</tr>
<tr>
<td>&gt;CMC</td>
<td>Clear all macros.</td>
</tr>
</tbody>
</table>
CNT: Decrement Hardware Counter

Command | Result
---|---
WHE <events> THE CNT, <action>,... | If all of the conditions specified in the event portion of the WHEN/THEN clause are satisfied, the counter is decremented. 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.

Comments

Events can be defined to selectively count bus cycles. There is one hardware counter, and 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.

Whenever the reset count, RCT, action is specified, the count comparator value for the specified group is loaded into the hardware counter. When switching groups, the current value of the hardware counter is passed along as a global count value unless a RCT action is specified in the same list of events that causes the group switch.

The toggle count, TOC, command allows you to turn counting on and off. When a TOC event is detected, the count is toggled to the opposite state, either on or off. You can specify an event that starts and stops the counter each time it is detected or specify any number of events that toggle the counter on and off.

The current value of the counter cannot be read. You can only detect when you have reached a limit.
**CNT: Decrement Hardware Counter**

This table describes the count conditions immediately before and after a group change.

<table>
<thead>
<tr>
<th>Previous Group</th>
<th>New Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><em>No Count Action Specified</em></td>
</tr>
<tr>
<td><em>No Count specified</em></td>
<td>No cycles counted</td>
</tr>
<tr>
<td><em>CNT</em></td>
<td>No cycles counted</td>
</tr>
<tr>
<td><em>TOC OFF (not counting)</em></td>
<td>No cycles counted</td>
</tr>
<tr>
<td><em>TOC ON (counting)</em></td>
<td>No cycles counted</td>
</tr>
</tbody>
</table>

This table describes initial count conditions (always group 1).

<table>
<thead>
<tr>
<th>Action Specified</th>
<th>Trace Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>No count</td>
<td>No cycles counted</td>
</tr>
<tr>
<td>CNT</td>
<td>Count only qualified CNT events</td>
</tr>
<tr>
<td>TOC</td>
<td>Count nothing until TOC event</td>
</tr>
</tbody>
</table>

**Examples**

This example counts the times that the specified data is written to a specific address and breaks if the data is written 20 times.

>\texttt{CTL=\#20} \quad \text{Set count limit to 20.}
>\texttt{S1=WR} \quad \text{Set status comparator to read/write.}
>\texttt{AC1=4020; DC1=\$XXF3} \quad \text{Set address and data comparators.}
>\texttt{WHEN AC1 AND DC1 AND S1 THEN CNT} \quad \text{Set WHEN/THEN statement to begin counting}
The second example looks for a read from a specific I/O port. After it is found go to group 2, load the group 2 counter register, and set a group 2 address comparator to count every bus cycle (all addresses). Break after 100 bus cycles.

>AC1='I0port
Set address of I/O port.

>S1=RD
Set status comparator to look for read access.

>WHEN AC1 AND S1 THEN GRO 2, RCT
When I/O port is read, go to group 2 and reset counter

>CTL.2=#100
Set group 2 count limit to 100.

>AC1.2=0 TO -1
Set address comparator to range.

>2 WHEN AC1 THEN CNT
When range accessed, count.

>2 WHEN CTL THEN BRK
When count limit reached, break

>RBK
Run til breakpoint.

>R>
Run mode prompt will appear.
COM: Communication With Target Programs

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM &lt;address&gt;</td>
<td>Establish communication with the target program through a two-byte pseudo-port at the specified address. Exit COM mode by entering the two-character transparent mode escape sequence (&lt;esc&gt;&lt;esc&gt; default).</td>
</tr>
</tbody>
</table>

Comments

COM is only useful during run mode. It affects real time operation.

In effect, the COM mode establishes a transparent mode between the running target program and the controlling port of the ES 1800. Whenever the ES 1800 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 ES 1800. 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 COM command requires special target code: 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 ES 1800, the target program first checks the most significant bit (MSB) of the byte at <address>. If this bit is set (1), the ES 1800 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 ES 1800, the target examines the byte at <address> + 1. If the MSB of this byte is cleared, the ES 1800 has not yet transmitted a new character. If the MSB is set, the character is new. If the controlling port of the ES 1800 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.

The flow diagram on the next page summarizes the COM process.
Figure 7-1: Flow Chart
Examples

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

>MAP 0 TO -1       Map all available overlay memory
>DNL
%cat serial.driver Download program to overlay (enter transparent mode escape sequence: 
<Rnv>esc><esc> default)
>RNV
R>COM 'serial_port Use serial data port as COM address

NOTE

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.
## CPY: Copy Data To Both Ports

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON CPY</td>
<td>Sends all data to both the terminal and computer ports. Data sent to the controlling port is echoed to the other port (noncontrolling port).</td>
</tr>
<tr>
<td>OFF CPY</td>
<td>Only sends data from the ES 1800 to the controlling port.</td>
</tr>
<tr>
<td></td>
<td>Default: OFF</td>
</tr>
</tbody>
</table>

### Comments

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

See Section 4, “Serial Communications,” for more information on the terminal and computer ports.
CRC, CRE, CRO: Target Cyclic Redundancy Check

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC &lt;range&gt;</td>
<td>The system calculates a cyclic redundancy check on all addresses in &lt;range&gt;.</td>
</tr>
<tr>
<td>CRE &lt;address range&gt;</td>
<td>Calculates a cyclic redundancy check on even addresses.</td>
</tr>
<tr>
<td>CRO &lt;address range&gt;</td>
<td>Calculates a cyclic redundancy check on odd addresses.</td>
</tr>
</tbody>
</table>

**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 $X^{16} + X^{15} + X^2 + 1$. 

*Alphabetical Command Reference* 7-39
**CTS: Convert Time Stamp**

**Command**

CTS # <countlimit>

**Result**
Convert countlimit to value required by ES 1800's Event Monitor System.

**Comments**

The absolute value of the time stamp counter can be used as one event in an Event Monitor System WHEN/THEN statement. The comparator LSA is used for the absolute value of the time stamp counter.

**Examples**

> CTS #2000
> AC1='counter_reset_address
> WHE AC1 THE TGR,GRO 2
> LSA.2=$0438
> 2 WHE LSA THE BRK

Convert desired count limit to value understood by the ES 1800. The ES 1800 will respond with $0438. This is the value the LSA port actually sees when the pod has counted 2000 times.

Specify address at which to reset counter.

Reset counter and switch to group 2 when AC1 is reached.

Specify the converted time stamp value as the limit at which to break emulation.

Break when counter value is reached.
**DB: Display Memory Block**

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB &lt;address range&gt;</td>
<td>Read and display the specified address range.</td>
</tr>
<tr>
<td>DB</td>
<td>Read and display 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.</td>
</tr>
<tr>
<td>DB &lt;address&gt;</td>
<td>Read and display one page of memory, starting at the specified address.</td>
</tr>
</tbody>
</table>

**Comments**

The page length is defined by the CRT length parameter in the SET menu. 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 (/). This repeats the DB command to increment the address and scroll through memory.

If the display is longer than one page, the XON/XOFF characters can be used to start and stop scrolling. (<ctrl-s>, <ctrl-q> default)

**Examples**

```
>WDM
>DB DS:DX LEN 20
>DB @SS:SP
```

Set global data length to word.
Display 20 words pointed to by DS:DX.
Display a page of values pointed to by the value on top of the stack.

(See Section 8, “Expressions,” for more information on @ operator).
The next example shows displaying a block in byte mode and word mode.

>BYM
>DB 0 LEN 20
000000 80 48 45 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 .HELLO../../
000010 030F 40F0 0C0F 40F0 - 0607 90F0 0C0F 00D8 ...@
000020 0FFF F9FF 1FFF 7FFF - 3FFF BDFB 1FFF FFFF
DEL: Delete A Symbol Or Section

**Command**

DEL ’<symbol>

DEL ’<section>

**Result**

Deletes the specified symbol.

Deletes the specified section.

**Examples**

>SYM

$00001000 Sym

$00008000 start

>DEL 'Sym; SYM

$00008000 start

> Display current symbols.

Delete symbol "Sym", and show remaining symbols.
# DES: Display Event Specifications

## Command

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES</td>
<td>Display all of the WHEN/THEN statements currently active from all groups.</td>
</tr>
<tr>
<td>DES &lt;group number&gt;</td>
<td>Display all of the WHEN/THEN statements and the comparator values for the specified group.</td>
</tr>
</tbody>
</table>

## Examples

Display the statements and comparators for groups 1 and 2.

```
>DES 1;RET;DES 2

Display information on group 1 and 2 setup, separated by a <return>.

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
```
**DFB: Default Base**

<table>
<thead>
<tr>
<th><strong>Command</strong></th>
<th><strong>Result</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>DFB</td>
<td>Display the global default base. On power-up the default base is hexadecimal unless another default base was loaded by the EEPROM on power-up.</td>
</tr>
<tr>
<td>DFB = #n</td>
<td>Set the default base to ( n ) (2-binary, 8-octal, 10-decimal, or 16-hexadecimal).</td>
</tr>
</tbody>
</table>

**Comments.**

Specific base prefixes can override the default base. Values not preceded by one of these prefixes are presumed by the ES 1800 to be in the default base.

<table>
<thead>
<tr>
<th><strong>Base prefix</strong></th>
<th><strong>Description</strong></th>
<th><strong>Example</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>Binary</td>
<td>%10011100001111</td>
</tr>
<tr>
<td>\</td>
<td>Octal</td>
<td>\23417</td>
</tr>
<tr>
<td>#</td>
<td>Decimal</td>
<td>#9999</td>
</tr>
<tr>
<td>$</td>
<td>Hexadecimal</td>
<td>$270F</td>
</tr>
</tbody>
</table>

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 ES 1800 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 ES 1800 defaults to the hexadecimal base.
**DIA: Display Character String**

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIA &lt;address&gt;</td>
<td>Read and display characters from target memory starting at the specified address. The DIA routine terminates when it reads $00 from target memory.</td>
</tr>
<tr>
<td></td>
<td>Affects real time operation when entered in run mode.</td>
</tr>
</tbody>
</table>

**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 ES 1800 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
>             
```

The next 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 ES 1800 immediately begins emulation and waits for another breakpoint and message.
>AC1 = 'Error_stop
Set address comparator at error routine.

>WHE AC1 THE BRK
Break when AC1 is reached.

>* RBK; WAI; DIA ES: BX; TST = DX
Run till breakpoint, and wait until breakpoint is reached. Display message pointed to by ES: BX. Then test to see if DX is 0. The * at the beginning of the line repeats the command, so that if the TST fails, the whole line is repeated.
## DIS: Memory Disassembler

### Command | Result
---|---
DIS `<range>` | Disassemble and display the data in the specified range.
DIS `<address>` | Disassemble one page of memory beginning at a specified address.
DIS | Disassemble and display 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 8018x, 80C18x or 80C18xEB assembly language programming and have the appropriate hardware manual:


Page length is defined by the CRT length parameter in the SET menu.

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 `<return>`. Exit disassembly by typing any other character.

- `<space>`: Continue disassembling one line at a time.
- `<return>`: Continue disassembling one page at a time.
- *any char except `<space>` or `<return>`*: Exit disassembly mode.
DM: Display Memory Map

Command | Result
---|---
DM | Display the memory map currently in effect.

Comments

This command is valid only in pause mode.

If the memory map scrolls off the screen, you may have a heat related problem with your emulator. See Section 2, Power-Up Sequence, for details.

Examples

>DM

Display memory map.

MEMORY MAP:

This is the default map at power-up.

$000000 TO $FFFFFF:TGT
## DME: Enable Data

*(8018x and 80C18x only)*

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ON DME</strong></td>
<td>The DMA controllers are active during pause. The values in DMA0 and DMA1 registers are not reloaded to the physical PCB during run-pause and pause-run transitions. 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.</td>
</tr>
<tr>
<td><strong>OFF DME</strong></td>
<td>The DMA controllers are not active during pause mode. Default: OFF</td>
</tr>
</tbody>
</table>

### 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. All bus read cycles go to target space during PAUSE mode if **DME** is ON.

Overlay will not respond to DMA during pause. All DMA cycles executed during pause will be directed to the target system.

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

This command is not recognized by the emulator for the 80C186EB or the 8C188EB processors.
## DNL: Download File

### Command | Result
--- | ---
DNL | **DNL** readies the ES 1800 to receive data. If in terminal control mode, the ES 1800 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 ES 1800. Refer to SET menu parameter #26 for verification of ES 1800 format. Use transparent mode (TRA) to verify host system format and the address in the file.

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.

See Section 4, “Downloading,” for more information.
DNV: Verify Download Data
(80C18x and 80C18xEB only)

Command Result

ON DNV Data received with the DNL command is verified after being written to memory.

Default: ON

OFF DNV Data is not verified after being written to memory.

Comments

The DNV command allows you to turn on and off the data verification performed by the ES 1800 after each byte of data is written. With the DNV switch ON, data is first written and then verified as successfully and accurately written. If the data is not successfully verified, an error message is displayed.

With the DNV switch OFF, you can perform write operations to non-readable memory space, such as MMU's. With this setting, memory writes are not immediately verified with a read operation.

With DNV OFF, code downloads are significantly faster than with DNV ON. With a reliable target, you may want to set this switch to OFF to more quickly download code.
## DR: Display/Load Microprocessor Registers

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR</td>
<td>Display values of all microprocessor registers.</td>
</tr>
<tr>
<td>(&lt;register\ name)&gt;</td>
<td>Display the value of the specified microprocessor register in its display base.</td>
</tr>
<tr>
<td>(&lt;register\ name&gt;=&lt;exp)&gt;</td>
<td>Assign the specified register the value (&lt;exp&gt;).</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear the four CPU data registers; AX, BX CX, and DX.</td>
</tr>
<tr>
<td>LDV</td>
<td>Load 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.</td>
</tr>
</tbody>
</table>

### 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 ES 1800 overlay memory to the microprocessor when run mode is entered. When emulation is broken, they are copied from the processor to ES 1800 overlay memory.

If a CPU register is loaded with a value during run mode, a warning message is 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 ES 1800’s internal memory, but when emulation is broken, the new value of the CPU register overwrites the value just entered.
DR: Display/Load Microprocessor Registers

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

O - Overflow
D - Direction
I - Interrupt
T - Trap
C - Carry

S - Sign
Z - Zero
A - Auxiliary carry
P - Parity

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 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

Examples

>DS=$A700;DS
Load the data segment and verify that it contains the correct value.

$A700
>
**DRT: Display Raw Trace Bus Cycles**

<table>
<thead>
<tr>
<th><strong>Command</strong></th>
<th><strong>Result</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>DRT</td>
<td>Display the last page of bus cycles recorded in trace memory.</td>
</tr>
<tr>
<td>DRT <em>&lt;line number&gt;</em></td>
<td>Display a page of the trace buffer starting with <em>&lt;line number&gt;</em>.</td>
</tr>
</tbody>
</table>
| DRT *<range>* | Display 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.

This command is valid only in pause mode.

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.
**Examples**

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

**LINE**

Line number 0 in the trace buffer indicates the last bus cycle prefetched or executed before the ES 1800 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.

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

### TAR/OVL
TAR/OVL indicates whether the access was in the target memory area or in the ES 1800’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:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFS</td>
<td>refresh</td>
</tr>
<tr>
<td>IAK</td>
<td>interrupt acknowledge</td>
</tr>
<tr>
<td>RIO</td>
<td>read from I/O</td>
</tr>
<tr>
<td>WIO</td>
<td>write to I/O</td>
</tr>
<tr>
<td>HLT</td>
<td>halt</td>
</tr>
<tr>
<td>IF</td>
<td>instruction fetch</td>
</tr>
<tr>
<td>RM</td>
<td>read memory</td>
</tr>
<tr>
<td>WM</td>
<td>write memory</td>
</tr>
<tr>
<td>NBC</td>
<td>no bus cycles</td>
</tr>
<tr>
<td>DMA</td>
<td>direct memory access</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Alternate Data</td>
</tr>
<tr>
<td>C</td>
<td>Code</td>
</tr>
<tr>
<td>D</td>
<td>Data</td>
</tr>
<tr>
<td>S</td>
<td>Stack</td>
</tr>
</tbody>
</table>

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 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 Section 4, "Breaking Emulation").
DT: Disassemble Trace Memory

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT</td>
<td>Disassemble and display the last instruction in trace memory. A sequence number is not included. Overwrites current display line.</td>
</tr>
<tr>
<td>DT &lt;range&gt;</td>
<td>Disassemble a range of bus cycles, starting at the specified value and proceeding back in time.</td>
</tr>
<tr>
<td>DT &lt;value&gt;</td>
<td>Disassemble a page of trace starting at &lt;value&gt;.</td>
</tr>
</tbody>
</table>

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.
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.
### Examples

>STP;DT  
>DT 0

<table>
<thead>
<tr>
<th>SEQ#</th>
<th>ADDR</th>
<th>OPCODE</th>
<th>MNEMONIC</th>
<th>OPERAND FIELDS</th>
<th>BUS CYCLE DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0028</td>
<td>000A</td>
<td>8B4600</td>
<td>MOV AX,WORD PTR [BP+0]</td>
<td></td>
<td>0800&gt;10C5</td>
</tr>
<tr>
<td>0027</td>
<td>000D</td>
<td>050100</td>
<td>ADD AX,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0024</td>
<td>0010</td>
<td>EBF4</td>
<td>JMP SHORT 0006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0020</td>
<td>0006</td>
<td>90</td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0019</td>
<td>0009</td>
<td>90</td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0018</td>
<td>000A</td>
<td>8B4600</td>
<td>MOV AX,WORD PTR [BP+0]</td>
<td></td>
<td>0800&lt;10C6</td>
</tr>
<tr>
<td>0017</td>
<td>000D</td>
<td>050100</td>
<td>ADD AX,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0014</td>
<td>0010</td>
<td>EBF4</td>
<td>JMP SHORT 0006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>0006</td>
<td>90</td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0009</td>
<td>0009</td>
<td>90</td>
<td>NOP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0008</td>
<td>000A</td>
<td>8B4600</td>
<td>MOV AX,WORD PTR [BP+0]</td>
<td></td>
<td>0800&gt;10C7</td>
</tr>
<tr>
<td>0007</td>
<td>000D</td>
<td>050100</td>
<td>ADD AX,1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

>  

**SEQ#**  
Correlates the disassembled instruction to the raw trace bus cycle. This is a decimal number and must be preceded by a # sign when referenced for selective disassembling of the trace. This corresponds to the line number in the DRT command display.

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

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

**MNEMONIC**  
The assembly-language instruction.

**OPERAND FIELD**  
The instruction operands.

**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.
DTB, D TF: Disassemble Trace Page

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTB</td>
<td>Disassemble the previous page of trace memory from current trace memory pointer.</td>
<td>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, DRT and / commands. 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.</td>
</tr>
<tr>
<td>DTF</td>
<td>Disassemble the following page of trace memory from the current trace memory pointer.</td>
<td></td>
</tr>
</tbody>
</table>

Refer also to the DT, DRT and / commands.

Alphabetical Command Reference 7-61
FIL: Fill Operator

Command Result

FIL <range>,<constant> Fill <range> with the <constant> data pattern.

Comments

This command may be used in run mode; however, it will affect real time emulation as it writes to memory.

<constant> must be an integer.

The FIL command uses the default data length, regardless of the length of <constant>. (See BYM and WDM).

The FIL command can be verified using the VBL (Verify Block) command.

Examples

> FIL 2000 LEN 50,0 Fill RAM with zero to initialize data space.
> FIL 'ram, 'init_data Fill RAM section with initialization data.
FIN: Find Pattern In Memory

**Command**

FIN <range>,<data>

**Result**

Search <range> for the data pattern. All occurrences of the pattern are displayed:

$\langle address\rangle=$<data>

If the pattern is not found within the range, you’ll see the error message:

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 for default data length in memory commands.)

Refer also to the “don’t care” description in Section 8, “Numbers.”

**Examples**

To find a bit pattern using don’t cares, use either of the following forms:

> WDM
> FIN 1000 TO 2FFF, 60XX

Use TO syntax to specify range.

or

> FIN 1000 LEN 1000,6000 DC OFF

Use LEN syntax to specify range.

The next examples shows finding the initialization data in the start module section and finding any NOPs in a range.

> BYM
> FIN ‘start_module,’init_uart

Find ‘init_uart data in ‘start_module.

> FIN 100 TO 1000,90

Find any NOPs in the range.

---

Alphabetical Command Reference 7-63
FSI: Force Special Interrupt

Command | Result
--- | ---
WHE `<events>` THE FSI, `<action>`;... | If all of the conditions specified in the event portion of the WHEN/THEN clause are satisfied, the force special interrupt action, FSI, allows you to jump to a specified address when a specific event is detected.

Comments

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 special interrupt address register, SIA, should be set prior to entering the run mode if you are using the FSI event. The SIA is a 32 bit integer, and 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 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.

Examples

Make a patch using overlay memory

```
>MAP 1000
>AC1=8F36
>WHEN AC1 THEN FSI
>SIA=1000
>ASM SIA
.
.
>RUN
```

Set up overlay map.

Set up address comparator.

When address reached, jump to special interrupt address.

Set up special interrupt address.

Use single line assembler beginning at special interrupt address. Patch code can be assembled here.

Begin emulation.
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.

```
>R> Run mode prompt will appear.
>SIA='SHUT_down
>AC1=$7F4E2
>AC2='SHUT_down + 4E
>WHEN AC1 THEN FSI
>WHEN AC2 THEN BRK
>RBK
>R>
```

Set up address of beginning of special shutdown routine.
Set up address comparator 1 as location to break at.
Set up address comparator 2 to be end of special shutdown routine.
At the first address, jump to special shutdown routine.
At end of shutdown routine, break.
Run til breakpoint.
Run mode prompt will appear.
### FSX: FSI On Instruction Execution

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON FSX</td>
<td>An Event Monitor System forced special interrupt (FSI) occurs when an instruction is executed. Refer to the FSI command.</td>
</tr>
<tr>
<td>OFF FSX</td>
<td>Forced special interrupt (FSI) occurs when an address is seen on the bus.</td>
</tr>
</tbody>
</table>

**Default:** ON

#### Comments

The 80186 family microprocessors prefetch 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.
GD: General Purpose Data Registers

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD&lt;0-7&gt;</td>
<td>Display the value of the specified general purpose data register.</td>
</tr>
<tr>
<td>GD&lt;0-7&gt; = &lt;value&gt;</td>
<td>Assign a value to one of the eight general purpose data registers. Value can be any integer or don't care value, but not a range.</td>
</tr>
</tbody>
</table>

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.

Examples

GD4 = 5000 General purpose data register 4 is loaded with 5000. GD4 can now be used anywhere you would use the number 5000.

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

GD2 = %01100101100 DC % 10011 Set GD2 to a specific pattern.

LSA = GD2; LSA.2 = GD2 Set up LSA registers in two groups.

GD3 = 'datpat1 DC %FF00 Set up GD3 to look for one byte of a specified word

General purpose registers can be used to help simplify using mode status mnemonics.

GD6 = ALT

MMS = GD6 Set MMS to ALT

GD1 = OVL+RD+IOA Set up a breakpoint on an overlay

S1 = GD1 read from I/O space.
GR: General Purpose Address Registers

**Command**

GR<0-7>  
GR<0-7> = <value>

**Result**

Display the value of the specified register.  
Assign a value to one of the eight general purpose address registers. <value> can be any integer or range.

**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.

**Examples**

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

The next example assigns a register to a commonly used range. Then you can use the register as a parameter for other commands.

`>GR0 = 'start_code LEN 20`  
Set up register.  
`>DIS GR0`  
Disassemble range specified in register.  
`>DB GR0`  
Display trace beginning at register.

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`
GRO: Change Event Groups

Command Result

WHE <events> THE GRO n, <action>,...
If all of the conditions specified in the event portion of the WHEN/THEN clause are satisfied, switch to group n (1-4).

Comments

The four event groups allow you to detect sequential events. When emulation is entered, the Event Monitor system always begins in group 1.

Examples

The example below describes a common use of the Event Monitor System 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.

>‘Module_A =1240 LEN 246 Define module A.
>’Module_B =8750 LEN 408 Define module B.
>’Sub_X =8934 LEN 56 Define subroutine X.
>ON BXX Enable breakpoints on instruction execution so that prefetching instructions don’t trigger event actions.

>AC1=’Module_A Set up address comparators for entire
>AC2=’Module_B range of modules A and B.

>WHE AC1 OR AC2 THE GRO 2 Set up WHEN/THEN statement so that any time you’re in either module, go to group 2.

>AC1.2=’Sub_X Set up comparator for subroutine X.
>2 WHEN AC1 THE TRC Look for Sub_X and start trace.

>2 WHE NOT AC1 THE GRO 1 At end of subroutine, return to group 1.

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.
IDP: Interrupts During Pause

IDP: Interrupts During Pause
(80C18x and 80C18x EB only)

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON IDP</td>
<td>Honor interrupts from the target system during pause mode. The associated interrupt routine will be executed.</td>
</tr>
<tr>
<td>OFF IDP</td>
<td>Ignore interrupts from the target system during pause mode. Default: OFF</td>
</tr>
</tbody>
</table>

Comments

If interrupts are not enabled with this soft-switch, no interrupts during pause mode are possible. The following requirements must be met in order to execute target interrupts during pause mode.

- The ESL variable PIA must be set to the address of a block of 16 bytes of unused memory. This block may be located in overlay, but it MUST BE UNUSED AND WRITABLE!
- The interrupt service routine must return execution to the location where the interrupt occurred (i.e., a normal return-from-interrupt).
- The interrupt service routine may not execute a halt (HLT) instruction.

If the above requirements are not met, proper operation of your emulator cannot be guaranteed.

NOTE

1. Enabling the IDP switch will slow the response time to some commands, such as memory reads. In order to speed command response time, interrupt service routines should not take excessive time because ESL cannot communicate with the pod while a target interrupt is being serviced.

   The worst case interrupt latency time in the target will be approximately 100 clock cycles when IDP is enabled and no ESL commands are being executed. However, in ninety percent of the cases, no additional latency at all will occur. The vast majority of interrupt services will reflect normal target operation.

2. Interrupt service routines executed while the emulator is in pause mode will not appear in the trace memory.
3. If you enter the reset character (default is <ctrl-z>), the IDP switch is automatically reset to the OFF state. You must enter the ON IDP command after resetting the emulator if you wish to honor target interrupts during pause mode.

4. INT4 may not be used during pause for the 80C18xEB.
**IHE: Ignore Halt Errors**  
(80C18x and 80C18x EB only)

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON IHE</td>
<td>Ignore halt errors during RUN mode.</td>
</tr>
<tr>
<td>OFF IHE</td>
<td>Display the message Processor Halted if a HLT instruction has been executed. Default: OFF</td>
</tr>
</tbody>
</table>

**Comments**

With Intel’s RMX86 operating system, the processor is frequently halted during normal operation between interrupts. The emulator recognizes these halts and reports an error message each time. To avoid numerous “Processor Halted” error messages, you can turn the emulator’s IHE switch ON and ignore halt errors during RUN mode.

With the IHE switch OFF, the emulator properly reports any RUN mode halt errors.
IOP: I/O Mode Pointer

**Command**  
IOP  
IOP = <exp>  

**Result**  
Display the current value of the I/O mode pointer.  
Assign 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 Section 7, “Memory and I/O Modes.”)

**Examples**  

```
> IOP=$1100;IOP
$00001100
>
```

Set the IOP and verify that it was set.
LD: Load System Variables From EEPROM

LD: Load System Variables From EEPROM

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>Copies all system variables stored in EEPROM into ES 1800 memory.</td>
</tr>
<tr>
<td>LD &lt;category&gt;</td>
<td>Copies the variables from one of the six categories in the EEPROM to the emulator RAM.</td>
</tr>
</tbody>
</table>

Comments

This command is valid only in pause mode.

Executing a LD command reads system variables from the EEPROM and copies them 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 ES 1800 registers
- 2 Event Monitor System WHEN/THEN statements
- 3 Overlay map
- 4 Software switch settings
- 5 Macros

Examples

- >LD 3
  Load the overlay map from EEPROM to internal RAM.

- >DM
  Verify the new map.

- ...

Alphabetical Command Reference
LDV: Load Reset Vectors

**Command**

LDV

**Result**

Load 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 = 0H
- 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 Section 4, “Setting Up Registers.”

**Examples**

```
>DR

Display registers

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

Load reset vectors, clear data registers, verify changes.

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
```

Alphabetical Command Reference 7-75
LOV: Load Overlay Memory

Command | Result
--- | ---
LOV <range> | Move data from the target system memory to the ES 1800 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 ES 1800 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 to verify the memory move.

Refer also to Section 4, “Mapping Overlay Memory,” and to the note under the discussion of the DIS command in this section.

Examples

> LOV 80000 LEN 7FFF
> LOV 'BOOT_RANGE

Load a section of overlay memory.
Load a section of overlay memory defined by a section.
M: 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 | Exit memory mode.

**Comments**

The M command affects real-time operation when entered in run mode.

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.

The commands to scroll the information displayed in memory mode are as follows:

- **<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.

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.

*Alphabetical Command Reference* 7-77
M: Enter Memory Mode

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 the example below.

Examples

>WDM; MMP=$FF000; NXT

Set global data length to word. Set the Memory Mode Pointer, and use the NXT command to enter memory mode.

$0FF000 $1234 >1122 Change a word of memory.
$0FF001 $00FF >, Verify the change.
$0FF000 $1122 >X Exit memory mode.

 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 (see ASM command).

>WDM; MMP=$FF000; NXT

Set global data length to word. Set the Memory Mode Pointer, and use the NXT command to enter memory mode.

$0FF000 $1234 >1122 Change a word of memory.
$0FF001 $00FF >, Verify the change.
$0FF000 $1122 >X Exit memory mode.

>  

Enter memory mode

$001000 $00 >0,1,2,3,4,5,6,7,8 Initialize data.

$001009 $00 >X Exit memory mode.

>ASM MMP Start line assembly at MMP.

**** 8086/88/186/188 LINE ASSEMBLER Vx.xLA ****

Enter your program here. Use "X" or "END" to exit the line assembler.
**MAC: Display Defined Macros**

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC</td>
<td>Display all defined macros in order #1-9,0.</td>
</tr>
</tbody>
</table>

**Examples**

```
> _1=DR; DIS CS:IP LEN 4; RUN
    Set up macro 1.
> _2=DB; SS:SP LEN 10; @'Data_ptr
    Set up macro 2.
> MAC
    Display macros.
> _1=DR; DIS CS:IP LEN 4; RUN
> _2=DB; SS:SP LEN 10; @'Data_ptr
> 
```
MAP: Set Memory Map

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAP &lt;range&gt;</td>
<td>Map the specified range and assign it the default attribute type, RW.</td>
</tr>
<tr>
<td>MAP &lt;value&gt;</td>
<td>Map a 2K-byte block surrounding the specified value. Assign the block the default attribute type, RW.</td>
</tr>
<tr>
<td>MAP &lt;range&gt;&lt;attribute&gt;</td>
<td>Map the specified range and assign it the specified attribute type.</td>
</tr>
<tr>
<td>MAP &lt;value&gt;&lt;attribute&gt;</td>
<td>Map a 2K-byte block surrounding the specified value. Assign the block the specified attribute.</td>
</tr>
</tbody>
</table>

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 ES 1800 aborts run mode and displays the error message, MEMORY WRITE VIOLATION. The contents of RO overlay cannot be altered by a running target program.

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

Memory mapped as illegal can be used to mark address ranges that should not be accessed by the target program. Any access to an address range mapped as ILG causes the ES 1800 to abort run mode and display the error message, *MEMORY ACCESS VIOLATION*. Memory mapped as ILG does not use up available overlay memory.

**TGT**

Memory is mapped to the target. 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 OVE).

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. You cannot save the contents of overlay memory in EEPROM.

**Examples**

The following command sequence might reflect a common mapping:

```plaintext
>CLM                        Clear map to all TGT.
>LDV                        Set CS:IP to OFFFF0 (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: Set Memory Map

MAP $040800 TO $0FF7FF:ILG
MAP $0FF800 TO $0FFFFFF:RO
MAP $100000 TO $FFFFFF:ILG >

Reset vectors.
MIO: Enter I/O Mode

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIO &lt;address&gt;</td>
<td>Enters I/O mode at &lt;address&gt;. The port address is displayed, but no data is read until a &lt;return&gt; is entered as the first character on the line.</td>
</tr>
<tr>
<td>MIO</td>
<td>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.</td>
</tr>
<tr>
<td>X</td>
<td>Exit I/O mode</td>
</tr>
</tbody>
</table>

Comments

Affects real-time operation when entered in run mode.

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)

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.

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.
MIO: Enter I/O Mode

Examples

> MIO $2F00
IO:$2F00 >$7F
IO:$2F00 >
IO:$2F00 $7F >X
>

> WDM
> MIO
IO:$2F00 >.
IO:$2F01 >
IO:$2F01 $05A6
> X
>

Enter I/O mode at address $2F00.
Write to a port.
Verify write.
Exit I/O mode.
Set global data length to word.
Enter I/O mode at last address.
Increment address.
Read the data.
Exit I/O mode.
**MMP: Memory Mode Pointer**

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMP</td>
<td>Display the current value of the memory mode pointer.</td>
</tr>
<tr>
<td>MMP = &lt;exp&gt;</td>
<td>Assign the value &lt;exp&gt; to the memory mode pointer.</td>
</tr>
</tbody>
</table>

**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 Section 4, “Memory Mode.”

**Examples**

The first example set the MMP and verifies that it has been set.

```
>MMP=$12330;MMP
$00012330
>
```

Set MMP and verify setting.

The second example 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
```

Exit memory mode.

Set address comparator to last address examined.
**NXT — SCROLLING IN MEMORY MODE**

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;return&gt;</td>
<td>Scrolls through memory addresses either one byte (8 bits) at a time, one word (16 bits) at a time, or one long word (32 bits) at a time. See <strong>BYT, WRD, LWM</strong>.</td>
</tr>
<tr>
<td>LST</td>
<td>The &lt;return&gt; key now decrements addresses in memory mode.</td>
</tr>
<tr>
<td>NXT</td>
<td>The &lt;return&gt; key now increments addresses in memory mode.</td>
</tr>
<tr>
<td>.</td>
<td>Increments the address in memory mode.</td>
</tr>
<tr>
<td>,</td>
<td>Decrements the address in memory mode.</td>
</tr>
</tbody>
</table>

**Comments**

The NXT and LST commands may be entered from either pause, run or memory mode. If entered from the 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 the memory mode, this temporarily overrides the scrolling direction.
ON/OFF: Switch Setting

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON</td>
<td>Display the ON/OFF menu. This menu is different for the 80186/188, 80C186/C188, and for the 80C186EB/80C188EB.</td>
</tr>
<tr>
<td>OFF</td>
<td>Display the ON/OFF menu. This menu is different for the 80186/188, 80C186/C188, and for the 80C186EB/80C188EB.</td>
</tr>
<tr>
<td>ON &lt;switch&gt;[+&lt;switch&gt;...]</td>
<td>Set the specified switch(es) to the ON position.</td>
</tr>
<tr>
<td>OFF &lt;switch&gt;[+&lt;switch&gt;...]</td>
<td>Set the specified switch(es) to the OFF position.</td>
</tr>
<tr>
<td>ON -1</td>
<td>Turn all switches on.</td>
</tr>
<tr>
<td>OFF -1</td>
<td>Turn all switches off.</td>
</tr>
</tbody>
</table>

Comments

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

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.

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 3-4) 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.

If it becomes necessary for you to reset the emulator (<ctrl-z> by default), remember that some switch settings are set to a default state. If you do not want them in their default state, you must reset the switches after resetting the emulator. You can conveniently do this with a macro or you may wish to save the switch values to EEPROM and execute an LD 5 command after resetting the emulator. A typical macro example is _3=ON IDP+DME.

For more information on any switch, see the alphabetical listing in this section.
**80186/188 Switch Settings Menu**

> 

> ON

**ES SWITCH SETTINGS MENU**

LD/SAY 4: LOAD/SAVE SWITCH SETTINGS IN EEPROM

EXAMPLES: > ON BKX+CK

> OFF FSX+CPY

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

>
80C186/C188 Switch Settings Menu

>ON

ES SWITCH SETTINGS MENU

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

<table>
<thead>
<tr>
<th>VALUE</th>
<th>NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>BKX</td>
<td>BREAK ON INSTRUCTION EXECUTION (NOT PREFETCH)</td>
</tr>
<tr>
<td>OFF</td>
<td>CPY</td>
<td>COPY DATA TO TERMINAL &amp; COMPUTER PORTS</td>
</tr>
<tr>
<td>ON</td>
<td>FSX</td>
<td>FSI ON INSTRUCTION EXECUTION (NOT PREFETCH)</td>
</tr>
<tr>
<td>ON</td>
<td>TCE</td>
<td>ENABLE TRACE MEMORY DURING RUN</td>
</tr>
<tr>
<td>OFF</td>
<td>PPT</td>
<td>ENABLE PEEK/POKE TRACE</td>
</tr>
<tr>
<td>OFF</td>
<td>RDY</td>
<td>SELECT INTERNAL READY WHEN ACCESSING OVERLAY</td>
</tr>
<tr>
<td>OFF</td>
<td>STI</td>
<td>ENABLE STEP THROUGH INTERRUPTS</td>
</tr>
<tr>
<td>OFF</td>
<td>BTE</td>
<td>BUS(RDY) TIMEOUT ENABLE</td>
</tr>
<tr>
<td>OFF</td>
<td>IHE</td>
<td>IGNORE HALT ERRORS</td>
</tr>
<tr>
<td>OFF</td>
<td>CK</td>
<td>SELECT INTERNAL CLOCK</td>
</tr>
<tr>
<td>OFF</td>
<td>IDP</td>
<td>ENABLE INTERRUPTS DURING PAUSE</td>
</tr>
<tr>
<td>ON</td>
<td>DNY</td>
<td>VERIFY DOWNLOADED DATA</td>
</tr>
<tr>
<td>OFF</td>
<td>DME</td>
<td>ENABLE DMA DURING PAUSE</td>
</tr>
<tr>
<td>OFF</td>
<td>TE0</td>
<td>ENABLE TIMER 0 DURING PAUSE</td>
</tr>
<tr>
<td>OFF</td>
<td>TE1</td>
<td>ENABLE TIMER 1 DURING PAUSE</td>
</tr>
<tr>
<td>OFF</td>
<td>TE2</td>
<td>ENABLE TIMER 2 DURING PAUSE</td>
</tr>
<tr>
<td>OFF</td>
<td>CDH</td>
<td>CLEAR DHLT BIT IN IST REGISTER ON PAUSE TO RUN</td>
</tr>
<tr>
<td>OFF</td>
<td>RCS</td>
<td>ENABLE CHIP SELECT REGISTERS DISPLAY</td>
</tr>
<tr>
<td>OFF</td>
<td>PRE</td>
<td>REFRESH DURING PAUSE</td>
</tr>
<tr>
<td>OFF</td>
<td>PCS</td>
<td>CHIP SELECTS DURING PAUSE</td>
</tr>
</tbody>
</table>

>
ON/OFF: Switch Setting

80C186EB/C188EB Switch Settings Menu

>ON

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)
OFF     CPY    COPY DATA TO TERMINAL & COMPUTER PORTS
ON      FSX    FSI ON INSTRUCTION EXECUTION (NOT PREFETCH)
OFF     PPT    ENABLE PEAK/POKE TRACE
OFF     RDY    SELECT INTERNAL READY WHEN ACCESSING OVERLAY
OFF     STI    ENABLE STEP THROUGH INTERRUPTS
OFF     BIE    BUS(RDY) TIMEOUT ENABLE
OFF     IHE    IGNORE HALT ERRORS
OFF     CK     SELECT INTERNAL CLOCK
OFF     IDP    ENABLE INTERRUPTS DURING PAUSE
ON      DNY    VERIFY DOWNLOADED DATA
OFF     TE0    ENABLE TIMER 0 DURING PAUSE
OFF     TE1    ENABLE TIMER 1 DURING PAUSE
OFF     TE2    ENABLE TIMER 2 DURING PAUSE
OFF     PRE    REFRESH DURING PAUSE
OFF     PCS    CHIP SELECTS DURING PAUSE
OFF     RSS    ENABLE SERIAL STATUS REGISTERS DISPLAY

Examples

If you want a hard copy of an emulation session, attach a printer to the computer port on the back chassis of the ES 1800. Turn on the copy switch so that all data is copied to both serial ports.

>ON CPY  Set the copy switch to on.
>
Assume that you are debugging a program on a new piece of hardware. The program has already been debugged using the ES 1800'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  Set two switches to ON using a +.
>OFF RDY+CK  Set two switches to OFF using a +.

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

Alphabetical Command Reference
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.

\[ \text{GR0} = \text{RDY+CK} \]  
Set general purpose register.

\[ \text{ON GR0} \]  
Turn on switches.

\[ \text{OFF GR0} \]  
Turn off switches.

The next way is to use two macros for the commands. Assign macros 1 and 2 to the **ON** and **OFF** commands. Execute these macros by typing a . and , as the first character on each line.

\[ _1=\text{ON RDY+CK} \]  
Define macro 1.

\[ _2=\text{OFF RDY+CK} \]  
Define macro 2.

\[ . \]  
Execute macro 1.

\[ , \]  
Execute macro 2.
## OVE: Overlay Memory Enable

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVE=CD+DTA</td>
<td>The overlay memory decodes both code and data space.</td>
</tr>
<tr>
<td>OVE=CD</td>
<td>Only code status space accesses are decoded by overlay memory.</td>
</tr>
<tr>
<td>OVE=DTA</td>
<td>Only data status space accesses (including ALT, DAT and STA space) are decoded by overlay memory.</td>
</tr>
</tbody>
</table>

### 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 (Section 4, "Trace Memory") 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.
# OVS: Overlay Memory Speed

(80C18x and 80C18xEB only)

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OVS</strong></td>
<td>Display the current value of the overlay memory speed register.</td>
</tr>
<tr>
<td><strong>OVS &lt;0-15&gt;</strong></td>
<td>Specify the number of wait states inserted before the overlay memory supplies a <strong>RDY</strong> signal to terminate the cycle. No wait states are inserted if <strong>OVS</strong> is zero. <strong>OVS</strong> is automatically set to 1 if <strong>CLK</strong> (clock frequency) is greater than 12.5 MHz and you are using a standard overlay memory board. You cannot override this automatic setting. If you are using a fast overlay memory board, <strong>OVS</strong> is set to zero.</td>
</tr>
</tbody>
</table>

## Comments

The value of **OVS** determines how many cycles occur before a **RDY** signal is returned by the overlay memory. The emulator’s wait state generator is only active when the **RDY** softswitch is on (ON RDY).

Assigning **OVS** a value of zero indicates that no wait states are inserted and the processor runs at full speed. A value of one inserts a single wait state, a value of two inserts two wait states, etc. The maximum number of wait states is fifteen.

## Chip-select Registers, Wait States, and Overlay Memory

The chip-select control registers in the Peripheral Control Block allow you to automatically insert wait states for memory affected by a given chip select. If **RDY** is ON, and overlay memory is mapped, the actual number of wait states inserted will be the greater of the number selected with the **OVS** command and the number selected by the PCB chip-select register. For example, if the chip-select register inserts 2 wait states and **OVS** is set to 4 wait states, the processor inserts 4 wait states.

When you set the R2 bit in the PCB chip-select register, the CPU ignores external **RDY** signals. In this case, the **OVS** value will have no effect, and the number of wait states inserted will always be as programmed in the chip-select register (from 0–3 wait states).

For standard overlay memory to run properly at speeds greater than 12.5 MHz, at least one wait state is required. If you use a chip-select control register to set the number of overlay wait states (using bits R0 and R1), be sure to program at least 1 wait state.
If you have the fast overlay memory board, no chip-select wait states need to be inserted.

16 MHz Overlay Operation (Standard Overlay Memory Board)

The standard overlay memory board cannot operate at 16 MHz without wait states. If you are running your target system at 16 MHz and you wish to access overlay memory, one of the following statements must be true.

- OVS is set to a value between one and fifteen, and the RDY switch is turned on.
- or -

Your target system is running with at least one wait state per memory access.

16 MHz Overlay Operation (Fast Overlay Memory Board)

A fast overlay memory board is available that supports memory accesses with zero wait-states (even with a 16 MHz target clock). Using the fast overlay memory board, you no longer need to carefully monitor the value of OVS, and the target clock frequency, or to program the chip-select control registers to add wait states, when operating at a clock frequency above 12.5 MHz.

NOTE

Note that OVS is not used unless RDY is ON, and both OVS and RDY apply to overlay memory accesses only.
### PCB: Display PCB Registers

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCB</td>
<td>Display contents of the peripheral control block registers.</td>
</tr>
</tbody>
</table>

#### Comments

Since the PCB is different for the 8018x, 80C18x and 80C18xEB processors, examples of all screens are shown on the following pages.

#### Examples

**8018x PCB Screen Display**

> `PCB`

<table>
<thead>
<tr>
<th><strong>RELOCATION REGISTER</strong></th>
<th>REL = 20FF</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMCS</td>
<td>LMCS</td>
</tr>
<tr>
<td>FFFB</td>
<td>0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>CHIP SELECT CONTROL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>UMCS</td>
</tr>
<tr>
<td>FFFB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>TIMER REGISTERS</strong></th>
<th>TC</th>
<th>MA</th>
<th>MB</th>
<th>MCV</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER 0</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>TIMER 1</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>TIMER 2</td>
<td>0000</td>
<td>0000</td>
<td>----</td>
<td>0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>DMA REGISTERS</strong></th>
<th>USRC</th>
<th>SRC</th>
<th>UDST</th>
<th>DST</th>
<th>XC</th>
<th>CV</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHANNEL 0</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>CHANNEL 1</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>INTERRUPT CONTROL REGISTERS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>EOI</td>
</tr>
<tr>
<td>0000</td>
</tr>
</tbody>
</table>
### PCB: Display PCB Registers

#### 80C18x PCB Screen Display

```plaintext
> 

```>PCB

* * RELLOCATION 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

** REFRESH/POWER DOWN REGISTERS  MDR  CDR  EDR  PDC
    0000  0000  0000  0000

** INTERRUPT CONTROL REGISTERS

    EDI  POL  POS  MSK  PLM  ISV  IRQ  IST  TCR  DMA0  DMA1  INTO  INT1  INT2  INT3
    0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  0000  000F  000F  000F
```
80C18xEB PCB Screen Display

>PCB
** RELocation REGISTER REL = 00FF
** CHIP SELECT CONTROL UCT UCP LCT LCP
            FF0F FFCF FFCF FFC3
    GCS0 GCS1 GCS2 GCS3 GCS4 GCS5 GCS6 GCS7
    STR FF0F FF0F FF0F FF0F FF0F FF0F FF0F FF0F
    STP FF0F FF0F FF0F FF0F FF0F FF0F FF0F FF0F
** TIMER REGISTERS TC MA MB MCW
    TIMER 0 0000 0000 0000 0000
    TIMER 1 0000 0000 0000 0000
    TIMER 2 0000 0000 ---- 0000
** I/O PORT CONTROL PDR PPN PCN PLT
    PORT 1 00FF 00FF 00FF 00FF
    PORT 2 00FF 00FF 00FF 00FF
** SERIAL PORT CONTROL SBD SCT SCN SST SRB STB
    PORT 0 0000 0000 0000 0008 0000 0000
    PORT 1 0000 0000 0000 0008 0000 0000
** REFRESH/POWER CONTROL RFBS RFTM RFCN RFAD PMC
    0000 0000 0000 1FFF 0000
** INTERRUPT CONTROL REGISTERS
EOI POL POS MSK PLM ISV IRQ IST TCR SCR INT4 INT0 INT1 INT2 INT3
    0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
>
PCS: Enable Chip Selects
(80C1Bx and 80C1BxEB only)

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON PCS</td>
<td>Chip selects are sent to the target system during PAUSE mode.</td>
</tr>
<tr>
<td>OFF PCS</td>
<td>Chip selects are not sent to the target system during PAUSE mode.</td>
</tr>
</tbody>
</table>

**Default**: OFF

**Comments.**

If **PCS** is set ON, all PCB chip select lines (UCS, LCS, etc.) will be driven to the target system during PAUSE mode.

If **PCS** is set OFF, all chip selects will be held de-asserted to the target system during PAUSE mode, but will be active during RUN mode. You may want to use this setting to prevent the selection of logic on your target by internal emulator activity. Such activity could corrupt memory, or activate I/O devices, etc.

**NOTE**

If you are using a target with an attached CPU in ONCE mode and plan to perform RESET operations, be sure to keep the **PCS** softswitch set to OFF to avoid driving a grounded chip select line.
**PPT: Trace Peeks and Pokes**  
*(80C18x and 80C18xEB only)*

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON PPT</td>
<td>Trace peek and poke cycles.</td>
</tr>
</tbody>
</table>
| OFF PPT | Do not trace peek and poke cycles.  
  Default: OFF |

**Comments**

With **PPT** ON, peeks and pokes (internal reads and writes) to target and overlay memory will be traced (provided that the **TCE** switch is also ON). Peeks and pokes are done by the MM, MIO, DB, DNL, FIL, @, UPL, LOV, VFO, and BMO ESL commands.

With this switch ON, proper disassembly of trace cannot be guaranteed due to the extra data cycles being traced.

With **PPT** OFF, the peek and poke trace cycles will not appear in trace.
PRE: DRAM Refresh During Pause
(80C18x and 80C18xEB only)

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON PRE</td>
<td>The DRAM refresh controller is active during pause mode.</td>
</tr>
<tr>
<td>OFF PRE</td>
<td>The DRAM refresh controller is <em>not</em> active during pause mode.</td>
</tr>
<tr>
<td></td>
<td>Default: OFF</td>
</tr>
</tbody>
</table>

**Comments**

When the emulator transitions between pause and run modes, the setting of the PRE switch determines whether the refresh register values are read from or written to the physical PCB and whether the refresh controller continues to run while the emulator is paused. The refresh control registers MDR, CDR and EDR are affected by the switch setting.

**Pause to Run Transition**

When the emulator transitions from pause to run mode, the PRE switch setting determines if the values of certain registers in the emulator’s RAM image are loaded to the physical PCB. For the 80C18x, these registers are the MDR, CDR and EDR; for the 80C18xEB, the registers loaded are RFBS, RFTM, RFCN, and RFAD.

If the PRE switch is OFF, the registers are loaded to the physical PCB.

If the PRE switch is ON, the registers are *not* loaded to the physical PCB. This prevents the currently active register values being overwritten with values from a previous run state.

**NOTE**

This means that if you set up the registers with ESL, you must STP first with PRE OFF to load the physical PCT. After that, you may turn PRE ON.

**Run to Pause Transition**

When the emulator transitions from run to pause mode, the current values of certain registers are loaded from the physical PCB to the emulator’s RAM image of the CPU registers. For the 80C18x, the registers are the MDR, CDR and EDR; for the 80C18xEB, the registers loaded are RFBS, RFTM, RFCN, and RFAD.

If the PRE switch is ON, no other action occurs and the refresh controller continues to run while the emulator is paused. All read bus cycles go to target space during PAUSE mode if PRE is ON.
If the **PRE** switch is OFF, the refresh controller is disabled immediately after the transition to pause mode by clearing bit 15 of the EDR register (80C18x) or the RFCN register (80C18xEB) in the physical PCB.

**NOTE**

If you enter the reset character (default is `<ctrl-z>`), the **PRE** switch is automatically reset to the OFF state.

You can modify refresh registers while you are in pause mode, and, if **PRE** is off, those values continue to be active when run mode is entered. Registers are modified using a `<register> = <value>` command.

The table below summarizes the effect of the refresh switch.

**Effect of PRE switch on Run/Pause Transitions**

<table>
<thead>
<tr>
<th>Switch Setting</th>
<th>Pause to Run Transition</th>
<th>Run to Pause Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ON</strong></td>
<td>The emulator’s RAM image of the refresh registers are <em>not</em> loaded to the physical PCB before entering run mode.</td>
<td>The value in the refresh registers are loaded into the emulator’s RAM image of the CPU registers.</td>
</tr>
<tr>
<td><strong>OFF</strong></td>
<td>The emulator’s RAM image of the refresh registers are loaded to the physical PCB just before running the target code.</td>
<td>The values in the refresh registers are loaded into the emulator’s RAM image of the CPU registers. The refresh controller is then disabled by clearing bit 15 of the EDR register.</td>
</tr>
</tbody>
</table>
PUR: Delete All Symbols And Sections

**Command**

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUR</td>
<td>Purge all symbols and section references.</td>
</tr>
</tbody>
</table>

**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 View symbols that are currently set.
00001000 sym
$00008000 start
$0000837E end

>SEC View sections that are currently set.
$00001000 TO $0000101F sec
$00008000 TO $0000837E init_mod
$00000000 TO $0000FFFFFF RAM

>PUR;SYM;SEC Purge symbols and sections, and verify purge.

>
RBK: Run Target Program

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBK</td>
<td>Begin executing the target program at the current CS:IP memory location with breakpoints enabled.</td>
</tr>
<tr>
<td>RBV</td>
<td>Load the restart vectors and begin executing the target program at memory location FFFFF0H with breakpoints enabled.</td>
</tr>
<tr>
<td>RUN</td>
<td>Begin executing the target program at the current CS:IP memory location with breakpoints disabled.</td>
</tr>
<tr>
<td>RNV</td>
<td>Load the restart vectors and begin executing the target program at memory location FFFFF0H with breakpoints disabled.</td>
</tr>
</tbody>
</table>

**Comments**

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.
For more information, see Section 4, “Breaking Emulation.”
# RBV: Run Target Program

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBK</td>
<td>Begin executing the target program at the current CS:IP memory location with breakpoints enabled.</td>
</tr>
<tr>
<td>RBV</td>
<td>Load the restart vectors and begin executing the target program at memory location FFF0H with breakpoints enabled.</td>
</tr>
<tr>
<td>RUN</td>
<td>Begin executing the target program at the current CS:IP memory location with breakpoints disabled.</td>
</tr>
<tr>
<td>RNV</td>
<td>Load the restart vectors and begin executing the target program at memory location FFF0H with breakpoints disabled.</td>
</tr>
</tbody>
</table>

## Comments

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.

For more information, see Section 4, “Breaking Emulation.”
RCS: Read Chip Select
8018x and 80C18x only

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON RCS</td>
<td>All chip select control registers are read upon run-to-pause.</td>
</tr>
<tr>
<td>OFF RCS</td>
<td>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</td>
</tr>
</tbody>
</table>

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. This command is not recognized by the emulator for the 80C186EB or the 80C188EB processors. The 80C18xEB behaves as though RCS is set to ON.
RCT: Reset Hardware Counter

**Command**

WHE <events> THE RCT,<action>,...

If all of the conditions specified in the event portion of the WHEN/THEN clause are satisfied, the RCT action loads the count comparator value for the specified group 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.

**Comments**

See the CNT action for a complete description of how the hardware counter works.

**Examples**

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=’I/Oport
- >S1=RD
- >WHEN AC1 AND S1 THEN GRO 2, RCT

When I/O port read occurs, go to group 2 and reset counter.

- >CTL.2=#100
- >AC1.2=0 TO -1

Set count limit in group 2. Set address comparator to match every address.

- >2 WHEN AC1 THEN CNT
- >2 WHEN CTL THEN BRK

Increment counter at every address. After 100 bus cycles, break.

- >RBK

Run til breakpoint.

- R>

Run mode prompt will appear.
RDY: Select Internal or External Ready Signal

Command | Result
--- | ---
ON RDY | Select an internally generated ready signal to complete memory accesses. This allows use of overlay memory when no target system is being used.

OFF RDY | Select 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 ES 1800 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 ES 1800 before the target hardware is ready.

NOTE: The default is ON if there is no target clock on power-up and if internal clock has been selected.
RET: Display A Blank Line

Command | Result
---|---
RET | Outputs a <return>, line feed.

Comments

This command improves readability when displaying a large amount of data.

Examples

Display two blocks of data, separating them with a blank line.
```plaintext
>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
```
REV: Display The Software Revision Dates

**Command**

REV

**Result**

Display 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 will ask you what software revisions are in your machine. This command gives you the necessary information.

**Examples**

```
>REV

Display revision of ESL and firmware.

WED AUG 6 08:50:26 PDT 1986 - ESL 2.2
WED AUG 6 16:50:26 PDT 1986 - FW 3.12
>
```
## RNV: Run Target Program

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBK</td>
<td>Begin executing the target program at the current CS:IP memory location with breakpoints enabled.</td>
</tr>
<tr>
<td>RBV</td>
<td>Load the restart vectors and begin executing the target program at memory location FFFF0H with breakpoints enabled.</td>
</tr>
<tr>
<td>RUN</td>
<td>Begin executing the target program at the current CS:IP memory location with breakpoints disabled.</td>
</tr>
<tr>
<td>RNV</td>
<td>Load the restart vectors and begin executing the target program at memory location FFFF0H with breakpoints disabled.</td>
</tr>
</tbody>
</table>

**Comments**

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.

For more information, see Section 4, “Breaking Emulation.”
### RSS: Read Serial Status

**80C18xEC only**

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON RSS</td>
<td>Enables reading of the serial status registers upon run-to-pause.</td>
</tr>
<tr>
<td>OFF RSS</td>
<td>Disables reading of the serial port status registers. The displayed values of serial status registers do not show what is actually in the PCB. Default: OFF</td>
</tr>
</tbody>
</table>

### Comments

The RSS switch only affects the status registers. All other serial port registers are read on every transition from run to pause. The entire status register, with the exception of the CTS bit, is cleared every time it is accessed, either for read or for write.

The serial port status register values in the internal RAM table are not written to the PCB, whether RSS is ON or OFF. If you change a serial port status register value using ESL, the new value will appear on the screen, but the value of the register in the microprocessor does not change. There are two ways to change the value: one is through your code; the other is through either the M or MIO commands.
RST: Reset

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>RST</td>
<td>Reset the pod microprocessor and loads the reset vectors.</td>
</tr>
<tr>
<td></td>
<td>$CS = \text{FFFFH}$</td>
</tr>
<tr>
<td></td>
<td>$IP = 0$</td>
</tr>
<tr>
<td></td>
<td>$FLX = \text{F002H}$</td>
</tr>
</tbody>
</table>

Comments

The RST command can be issued from either run or pause mode. When in pause mode, the RST command resets the microprocessor 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 target reset signal; therefore no target hardware is reset. This may cause problems when the target program tries to interact with uninitialized hardware.

Both <ctrl-z> and the RST command stop emulation in run mode. <ctrl-z> does not initialize the emulator registers.

Examples

In the example below, the ES 1800 is in run mode. The microprocessor is reset in the target environment and emulation continues.

```
R> RST
```

From run mode, enter a microprocessor reset.

```
R>
```

In the next example, the ES 1800 is in pause mode. The microprocessor is reset and the reset vectors are loaded into the ES 1800 registers.

```
> RST
```

From pause mode, enter a microprocessor reset.

>
**RUN: Run Target Program**

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBK</td>
<td>Begin executing the target program at the current CS:IP memory location with breakpoints enabled.</td>
</tr>
<tr>
<td>RBV</td>
<td>Load the restart vectors and begin executing the target program at memory location FFFF0H with breakpoints enabled.</td>
</tr>
<tr>
<td>RUN</td>
<td>Begin executing the target program at the current CS:IP memory location with breakpoints disabled.</td>
</tr>
<tr>
<td>RNV</td>
<td>Load the restart vectors and begin executing the target program at memory location FFFF0H with breakpoints disabled.</td>
</tr>
</tbody>
</table>

**Comments**

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.

For more information, see Section 4, “Breaking Emulation.”
SAV: Save System Variables In EEPROM

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAV</td>
<td>Copies all system variables from ES 1800 memory into EEPROM.</td>
</tr>
<tr>
<td>SAV &lt;category&gt;</td>
<td>Saves one of the six categories of variables from ES 1800 RAM to EEPROM.</td>
</tr>
</tbody>
</table>

**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 ES 1800. There is room in EEPROM to save the system variables for two different users. The user is determined by a parameter in the SET menu. When you execute a SAV, the variables are saved to the user partition currently defined in the SET menu.

This chart shows the categories of information that can be saved in EEPROM.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SET menu</td>
</tr>
<tr>
<td>1</td>
<td>Contents of ES 1800 registers</td>
</tr>
<tr>
<td>2</td>
<td>Event Monitor System</td>
</tr>
<tr>
<td></td>
<td>WHEN/THEN statements</td>
</tr>
<tr>
<td>3</td>
<td>Overlay map</td>
</tr>
<tr>
<td>4</td>
<td>Software switch settings</td>
</tr>
<tr>
<td>5</td>
<td>Macros</td>
</tr>
</tbody>
</table>

Variables are loaded from EEPROM back to the ES 1800 using the LD command.

When you first use the ES 1800, you should execute a SAV command with no parameter. This initializes EEPROM, so that subsequent LD commands will work properly with the ES 1800 board and pod. Factory default positions of the thumbwheel switch are shown on Table 3-1 on page 3-4.

**Examples**

>SAV 1

Save current value of ES 1800 registers to EEPROM.
# SEC: Display Section

## Command

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEC</td>
<td>Display all currently defined sections and their values.</td>
</tr>
<tr>
<td>SEC &lt;value&gt;</td>
<td>Display the section assigned the specified value.</td>
</tr>
<tr>
<td>'section'</td>
<td>Display the value of the specified section.</td>
</tr>
<tr>
<td>'section' = &lt;range&gt;</td>
<td>Assign the &lt;range&gt; to the specified section.</td>
</tr>
</tbody>
</table>

## Examples

- `>sec = 1000 LEN IF` Define section using LEN syntax.
- `>RAM = $0000 TO $FFFF` Define section using TO syntax.
- `>init_mod = 'start TO 'end` Define section using TO syntax and symbols.
- `>SEC` Display sections.

```
$00001000 TO $0000101F sec
$00000000 TO $0000FFFF RAM
$00008000 TO $0000837E init_mod
```
SET: Set Up Parameters

Command | Result
--- | ---
SET | Display the SET menu. The parameters in this menu specify the external communication details.
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 preceded 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.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Reset Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET #1,#0</td>
<td>User 0</td>
<td>No</td>
</tr>
<tr>
<td>SET #1,#1</td>
<td>User 1</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Two users may save and load values to the EEPROM. This parameter indicates which user is active when executing the SAV and LD commands.</td>
<td></td>
</tr>
<tr>
<td>SET #2,$n</td>
<td>Reset character</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>The reset character resets the ES 1800 and the pod CPU. The system default is &lt;ctrl-z&gt; ($1A).</td>
<td></td>
</tr>
<tr>
<td>SET #3,$n,$m</td>
<td>XON/XOFF characters</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>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).</td>
<td></td>
</tr>
</tbody>
</table>
### SET: Set Up Parameters

| SET #9,#0 | LSA value shows as 16 bits (default) | Yes |
| #1 | Display absolute time stamp value |
| #2 | Display relative time stamp value |

| 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 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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Reset Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET #13,#n</td>
<td>CRT length (default: 24 lines)</td>
<td>No</td>
</tr>
</tbody>
</table>

The maximum number of lines displayed for commands that use paging
SET: Set Up Parameters

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th>Reset Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET #23,$n,$m</td>
<td>Transparent mode escape sequence</td>
<td>No</td>
</tr>
</tbody>
</table>
When entered from the computer port, transparent mode is exited. The default sequence is <esc><esc> ($1B,$1B).
### SET #24,$n,$m,$0

Command terminator sequence

The default sequence is <return>, null, null ($OD, $00, $00).

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Displayed</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### SET #25,#n

Upload record length

The maximum length for an upload record. (The default length is 32 bytes of data.)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Displayed</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### SET #26,#0

Intel (default)

<table>
<thead>
<tr>
<th>#1</th>
<th>MOS</th>
<th>No</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>#2</td>
<td>Motorola</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#3</td>
<td>Signetics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#4</td>
<td>Tektronix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#5</td>
<td>Extended Tekhex</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Upload/download serial data format

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel (default)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Motorola</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signetics</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tektronix</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extended Tekhex</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### SET #27,$n

Acknowledge character

The acknowledge character is sent when a valid record is received when downloading in computer control. The default is $06.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Displayed</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 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 3-4) before turning on the ES 1800. To load manually, enter LD (to load all variables and settings) or enter the LD 0 command (to load just the SET parameters).

See Section 4, "Serial Communication," for information on communicating with a host computer.
## SF: Special Functions List

### Command | Result
---|---
SF | Display list of all available RAM tests, scope loops and miscellaneous tests.

### Examples

```plaintext
>SF
SF 0,<RANGE><CR> SIMPLE RAM TEST, SINGLE PASS
SF 1,<RANGE><CR> COMPLETE RAM TEST, SINGLE PASS
SF 2,<RANGE><CR> SIMPLE RAM TEST, LOOPING
SF 3,<RANGE><CR> COMPLETE RAM TEST, LOOPING

SCOPE LOOPS: (SELECT NUMBER FOR I/O LOOPS)
SF 4 (24),<ADDRESS>,<PATTERN><CR> TOGGLE DATA AT ADDRESS
SF 5 (25),<ADDRESS><CR> READ FROM ADDRESS
SF 6 (26),<ADDRESS>,<DATA><CR> WRITE DATA TO ADDRESS
SF 7 (27),<ADDRESS>,<PATTERN><CR> WRITE PATTERN, THEN PATTERN COMPLEMENT
SF 8 (28),<ADDRESS>,<PATTERN><CR> WRITE PATTERN, THEN ROTATE
SF 9 (29),<ADDRESS>,<DATA><CR> WRITE DATA, THEN READ
SF 11 (31),<ADDRESS>,<DATA><CR> WRITE INCREMENTING VALUE
SF 12 (32),<RANGE><CR> READ DATA OVER ENTIRE RANGE

MISCELLANEOUS:
SF 13<CR> CRC CHECK OF EMULATOR FIRMWARE
CLK <CR> DISPLAY TARGET CLOCK FREQUENCY
CRC <RANGE><CR> CALCULATE CRC OF SPECIFIED RANGE
CRE/CRO <RANGE><CR> CALCULATE CRC OF EVEN/ODD BYTES ONLY
```
## SF 0: Simple RAM Test, Single Pass

### Command

SF 0,<\text{range}>  

### Result

Write a test pattern to all locations within the specified range, then reads each location to verify the data. The following pattern sequence is used:

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF 0, &lt;range&gt;</td>
<td>Write a test pattern to all locations within the specified range, then reads each location to verify the data. The following pattern sequence is used:</td>
</tr>
</tbody>
</table>

### Pattern

<table>
<thead>
<tr>
<th>Pattern</th>
<th>BYM</th>
<th>WDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>000000000000000000</td>
<td></td>
</tr>
<tr>
<td>10000000</td>
<td>000000001000000000</td>
<td></td>
</tr>
<tr>
<td>11000000</td>
<td>000000001100000000</td>
<td></td>
</tr>
<tr>
<td>11100000</td>
<td>000000001110000000</td>
<td></td>
</tr>
<tr>
<td>11110000</td>
<td>000000001111000000</td>
<td></td>
</tr>
<tr>
<td>11111000</td>
<td>000000001111100000</td>
<td></td>
</tr>
<tr>
<td>11111100</td>
<td>000000001111110000</td>
<td></td>
</tr>
<tr>
<td>11111110</td>
<td>000000001111111000</td>
<td></td>
</tr>
<tr>
<td>11111111</td>
<td>000000001111111110</td>
<td></td>
</tr>
<tr>
<td>01111111</td>
<td>000000000111111111</td>
<td></td>
</tr>
<tr>
<td>00111111</td>
<td>000000000011111111</td>
<td></td>
</tr>
<tr>
<td>00011111</td>
<td>000000000001111111</td>
<td></td>
</tr>
<tr>
<td>00001111</td>
<td>000000000000111111</td>
<td></td>
</tr>
<tr>
<td>00000111</td>
<td>000000000000011111</td>
<td></td>
</tr>
<tr>
<td>00000011</td>
<td>000000000000001111</td>
<td></td>
</tr>
<tr>
<td>00000001</td>
<td>000000000000000111</td>
<td></td>
</tr>
<tr>
<td>00000000</td>
<td>000000000000000011</td>
<td></td>
</tr>
</tbody>
</table>

---

**Alphabetical Command Reference**
SF 0: Simple RAM Test, Single Pass

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.
SF 1: Complete RAM Test, Single Pass

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF 1,&lt;range&gt;</td>
<td>Write, then read, a test pattern to all locations in the specified range. Refer to <em>Efficient Algorithms for Test Semiconductor Random-Access Memories</em> mentioned in the introduction to Diagnostic Functions for the test pattern.</td>
</tr>
</tbody>
</table>

**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.
## SF 2: Simple RAM Test, Looping

### Command

<table>
<thead>
<tr>
<th>SF 2,&lt;range&gt;</th>
</tr>
</thead>
</table>

### Result

Write 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 use the reset character to terminate this test (`<ctrl-z>` default, can be changed with SET).
SF 3: Complete RAM Test, Looping

**Command**

SF 3, <range>

**Result**

Write a test pattern to all locations within <range>, then read 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 use the reset character to terminate this test. (<ctrl>-z> default, can be changed with SET).
SF 4: Toggle Data At Address

SF 4: Toggle Data At Address

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF 4&lt;address&gt;,&lt;data&gt;</td>
<td>Write &lt;data&gt; to the specified address in the memory space defined by MMS.</td>
</tr>
<tr>
<td>SF 24,&lt;address&gt;,&lt;data&gt;</td>
<td>Write &lt;data&gt; to the specified address in I/O space.</td>
</tr>
<tr>
<td></td>
<td>Write the user defined data pattern to &lt;address&gt;, alternating with a data pattern of zeros.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SEQ</th>
<th>BYM</th>
<th>WDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>XX</td>
<td>XXXX (user data)</td>
</tr>
<tr>
<td>3</td>
<td>00</td>
<td>0000</td>
</tr>
<tr>
<td>4</td>
<td>XX</td>
<td>XXXX (user data)</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

Comments

These commands are valid in pause mode only.

You must use the reset character to terminate these tests. (<ctrl-z> default, can be changed with SET).

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
.
.
.
SF 5: Peeks Into The Target System

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF 5, &lt;address&gt;</td>
<td>Consecutively read from the specified memory address using MMS as status space register.</td>
</tr>
<tr>
<td>SF 25, &lt;address&gt;</td>
<td>Consecutively read from the specified I/O address.</td>
</tr>
</tbody>
</table>

Comments

These commands are valid in pause mode only.

You must use the reset character to terminate these tests. (<ctrl-z> default, can be changed with SET).

Examples

> SF 5, 2

YOU MUST RESET ME TO TERMINATE THIS FUNCTION
**SF 6: Pokes Into The Target System**

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF 6,&lt;address&gt;,&lt;data&gt;</td>
<td>Consecutively write the user defined data pattern to the specified memory address using MMS as status space register.</td>
</tr>
<tr>
<td>SF 26,&lt;address&gt;,&lt;data&gt;</td>
<td>Consecutively write the user defined data pattern to the specified I/O address.</td>
</tr>
</tbody>
</table>

**Comments**

These commands are valid in pause mode only.

You must use the reset character to terminate these tests. (<ctrl-z> default, can be changed with SET).

**Examples**

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

The data pattern written to address 10 is:

- (BYM) (WDM)
- FF FFFF
- FF FFFF
- FF FFFF
- FF FFFF
### SF 7: Write Alternate Patterns

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF 7,&lt;address&gt;,&lt;pattern&gt;</td>
<td>Consecutively write 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.</td>
</tr>
<tr>
<td>SF 27,&lt;address&gt;,&lt;pattern&gt;</td>
<td>Consecutively write the user defined data pattern to the specified I/O address followed by the complement of that data pattern to the same address.</td>
</tr>
</tbody>
</table>

**Comments**

These commands are valid in pause mode only.

You must use the reset character to terminate these tests. (ctrl-z default, can be changed with SET).

**Examples**

```plaintext
> SF 7, 10, 55
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```

The following data pattern is written to address 10:

```
BYM  WDM
55   0055
AA   FFAA
55   0055
AA   FFAA
  .   .
  .   .
  .   .
```
**SF 8: Write Pattern Then Rotate**

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF 8,&lt;address&gt;,&lt;pattern&gt;</td>
<td>Consecutively write 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.</td>
</tr>
<tr>
<td>SF 28,&lt;address&gt;,&lt;pattern&gt;</td>
<td>Consecutively write the data pattern to the specified I/O address, rotates the pattern 1 bit to the left, and write to the same address.</td>
</tr>
</tbody>
</table>

**Comments**

These commands are valid in pause mode only.

You must use the reset character to terminate these tests. (<ctrl-z> default, can be changed with SET).

**Examples**

> SF 8,1000,05

YOU MUST RESET ME TO TERMINATE THIS FUNCTION
The following data pattern is written to address 10:

<table>
<thead>
<tr>
<th>BYM</th>
<th>WDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>05</td>
<td>0005</td>
</tr>
<tr>
<td>0A</td>
<td>000A</td>
</tr>
<tr>
<td>14</td>
<td>0014</td>
</tr>
<tr>
<td>28</td>
<td>0028</td>
</tr>
<tr>
<td>50</td>
<td>0050</td>
</tr>
<tr>
<td>A0</td>
<td>00A0</td>
</tr>
<tr>
<td>41</td>
<td>0140</td>
</tr>
<tr>
<td>82</td>
<td>0280</td>
</tr>
<tr>
<td>050</td>
<td>0A00</td>
</tr>
<tr>
<td>1400</td>
<td>2800</td>
</tr>
<tr>
<td>5000</td>
<td>A000</td>
</tr>
<tr>
<td>4001</td>
<td>8002</td>
</tr>
</tbody>
</table>
SF 9: Write Data Then Read

**Command**

SF 9,<address>,<data>

SF 29,<address>,<data>

**Result**

Consecutively write the specified data pattern to the specified memory address using MMS as status space register, then read from that same address.

Consecutively write the specified data pattern to the specified I/O address, then read from that same address.

**Comments**

These commands are valid in pause mode only.

You must use the reset character to terminate these tests. (<ctrl-z> default, can be changed with SET).

**Examples**

> SF 9, 100,$FFFF

YOU MUST RESET ME TO TERMINATE THIS FUNCTION
### SF 11: Write Incrementing Value

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF 11,&lt;address&gt;</td>
<td>Consecutively write a constantly incrementing value to the specified memory address using MMS as status space register.</td>
</tr>
<tr>
<td>SF 31,&lt;address&gt;</td>
<td>Consecutively write a constantly incrementing value to the specified I/O address.</td>
</tr>
</tbody>
</table>

**Comments**

These commands are valid in pause mode only.

You must use the reset character to terminate these tests. (<ctrl-z> default, can be changed with SET).

**Examples**

```
> SF 11, 100
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```
## SF 12: Read Data Over An Entire Range

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF 12,&lt;range&gt;</td>
<td>Consecutively read from the specified memory address range using MMS as status space register.</td>
</tr>
<tr>
<td>SF 32,&lt;range&gt;</td>
<td>Consecutively read from the specified I/O address range.</td>
</tr>
</tbody>
</table>

**Comments**

These commands are valid in pause mode only.

The ES 1800 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 use the reset character to terminate these tests. (<ctrl-z> default, can be changed with SET).

**Examples**

```plaintext
> SF 12, 10 TO 20
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```
SF 13: Cyclic Redundancy Check

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF 13</td>
<td>A CRC is calculated on the ES 1800 internal PROM that contains the ES 1800 firmware.</td>
</tr>
</tbody>
</table>

Comments

This command is valid in pause mode only.
This is an ES 1800 self-test.
If a failure is detected, a CRC error is displayed.
This is a single pass routine.
When the text completes without an error, the command prompt (>) is displayed.
SF 24: Toggle Data At Address

**SF 24: Toggle Data At Address**

**Command** | **Result**
--- | ---
SF 4<address>,<data> | Write <data> to the specified address in the memory space defined by MMS.
SF 24,<address>,<data> | Write <data> to the specified address in I/O space. Write the user defined data pattern to <address>, alternating with a data pattern of zeros.

<table>
<thead>
<tr>
<th>SEQ</th>
<th>BYM</th>
<th>WDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>XX</td>
<td>XXXX (user data)</td>
</tr>
<tr>
<td>3</td>
<td>00</td>
<td>0000</td>
</tr>
<tr>
<td>4</td>
<td>XX</td>
<td>XXXX (user data)</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

**Comments**

These commands are valid in pause mode only.
You must use the reset character to terminate these tests. (<ctrl-z> default, can be changed with SET).

**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
```

.
SF 25: Peeks Into The Target System

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF 5,&lt;address&gt;</td>
<td>Consecutively read from the specified memory address using MMS as status space register.</td>
</tr>
<tr>
<td>SF 25,&lt;address&gt;</td>
<td>Consecutively read from the specified I/O address.</td>
</tr>
</tbody>
</table>

**Comments**

These commands are valid in pause mode only.
You must use the reset character to terminate these tests. (<ctrl-z> default, can be changed with SET).

**Examples**

> SF 5, 2  
> YOU MUST RESET ME TO TERMINATE THIS FUNCTION
SF 26: Pokes Into The Target System

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF 6,&lt;address&gt;,&lt;data&gt;</td>
<td>Consecutively write the user defined data pattern to the specified memory address using MMS as status space register.</td>
</tr>
<tr>
<td>SF 26,&lt;address&gt;,&lt;data&gt;</td>
<td>Consecutively write the user defined data pattern to the specified I/O address.</td>
</tr>
</tbody>
</table>

Comments

These commands are valid in pause mode only.
You must use the reset character to terminate these tests. (<ctrl-z> default, can be changed with SET).

Examples

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

The data pattern written to address 10 is:

(BYM) (WDM)

FF FFFF
FF FFFF
FF FFFF
FF FFFF
SF 27: Write Alternate Patterns

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF 7,&lt;address&gt;,&lt;pattern&gt;</td>
<td>Consecutively write 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.</td>
</tr>
<tr>
<td>SF 27,&lt;address&gt;,&lt;pattern&gt;</td>
<td>Consecutively write the user defined data pattern to the specified I/O address followed by the complement of that data pattern to the same address.</td>
</tr>
</tbody>
</table>

**Comments**

These commands are valid in pause mode only.
You must use the reset character to terminate these tests. (<ctrl-z> default, can be changed with SET).

**Examples**

> SF 7, 10, 55
YOU MUST RESET ME TO TERMINATE THIS FUNCTION

The following data pattern is written to address 10:

```
BYM    WDM
55  0055
AA    FFAA
55  0055
AA    FFAA
   .
   .
   .
   .
```
SF 28: Write Pattern Then Rotate

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF 8, &lt;address, &lt;pattern&gt;</td>
<td>Consecutively write 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.</td>
</tr>
<tr>
<td>SF 28, &lt;address&gt;, &lt;pattern&gt;</td>
<td>Consecutively write the data pattern to the specified I/O address, rotates the pattern 1 bit to the left, and write to the same address.</td>
</tr>
</tbody>
</table>

**Comments**

These commands are valid in pause mode only.

You must use the reset character to terminate these tests. (<ctrl-z> default, can be changed with SET).

**Examples**

```
>SF 8, 1000, 05
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```
The following data pattern is written to address 10:

<table>
<thead>
<tr>
<th>BYM</th>
<th>WDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>05</td>
<td>0005</td>
</tr>
<tr>
<td>0A</td>
<td>000A</td>
</tr>
<tr>
<td>14</td>
<td>0014</td>
</tr>
<tr>
<td>28</td>
<td>0028</td>
</tr>
<tr>
<td>50</td>
<td>0050</td>
</tr>
<tr>
<td>A0</td>
<td>00A0</td>
</tr>
<tr>
<td>41</td>
<td>0140</td>
</tr>
<tr>
<td>82</td>
<td>0280</td>
</tr>
<tr>
<td></td>
<td>0500</td>
</tr>
<tr>
<td></td>
<td>0A00</td>
</tr>
<tr>
<td></td>
<td>1400</td>
</tr>
<tr>
<td></td>
<td>2800</td>
</tr>
<tr>
<td></td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>A000</td>
</tr>
<tr>
<td></td>
<td>4001</td>
</tr>
<tr>
<td></td>
<td>8002</td>
</tr>
</tbody>
</table>
SF 29: Write Data Then Read

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF 9,&lt;address&gt;,&lt;data&gt;</td>
<td>Consecutively write the specified data pattern to the specified memory address using MMS as status space register, then read from that same address.</td>
</tr>
<tr>
<td>SF 29,&lt;address&gt;,&lt;data&gt;</td>
<td>Consecutively write the specified data pattern to the specified I/O address, then read from that same address.</td>
</tr>
</tbody>
</table>

Comments

These commands are valid in pause mode only. You must use the reset character to terminate these tests. (<ctrl-z> default, can be changed with SET).

Examples

```
>SF 9, 100, $FFFF
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```
## SF 31: Write Incrementing Value

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF 9, &lt;address&gt;, &lt;data&gt;</td>
<td>Consecutively write a constantly incrementing value to the specified memory address using MMS as status space register.</td>
</tr>
<tr>
<td>SF 29, &lt;address&gt;, &lt;data&gt;</td>
<td>Consecutively write a constantly incrementing value to the specified I/O address.</td>
</tr>
</tbody>
</table>

### Comments

These commands are valid in pause mode only.

You must use the reset character to terminate these tests. (<ctrl-z> default, can be changed with SET).

### Examples

```
> SF 11, 100
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```
SF 32: Read Data Over An Entire Range

**Command**

SF 12,<range>  
SF 32,<range>

**Result**

Consecutively read from the specified memory address range using MMS as status space register.

Consecutively read from the specified I/O address range.

**Comments**

These commands are valid in pause mode only.

The ES 1800 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 use the reset character to terminate these tests. (<ctrl-z> default, can be changed with SET).

**Examples**

```
> SF 12, 10 TO 20
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```
## STI: Step Through Interrupts

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ON STI</strong></td>
<td>The ES 1800 recognizes an interrupt and steps through the interrupt service routine. If this switch is ON at the return to run mode, the interrupt flag will be active.</td>
</tr>
<tr>
<td><strong>OFF STI</strong></td>
<td>The ES 1800 ignores interrupts while stepping through a program. If this switch is off at the return to run mode, the interrupt flag will not be active. Default: OFF</td>
</tr>
</tbody>
</table>

### 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).
**STP: Stop And Step Target System**

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>R&gt;STP</td>
<td>From run mode the STP stops emulation and returns to pause mode. Display the current CS:IP address and the Event Monitor System group number.</td>
</tr>
<tr>
<td>&gt;STP</td>
<td>From pause mode, the STP command executes one instruction. To receive visual feedback, combine this command with a display command such as STP;DT.</td>
</tr>
</tbody>
</table>

**Comments**

R> indicates that the ES 1800 is in run mode. > indicates that the ES 1800 is in pause mode.

See the switch information under STI 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.

**Examples**

>STP;DR
>STP;DT
>STP;DIS IP LEN 4
SYM: Display Symbols

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYM</td>
<td>Display all defined symbols.</td>
</tr>
<tr>
<td>SYM &lt;value&gt;</td>
<td>Display all symbols assigned the specified value.</td>
</tr>
<tr>
<td>'&lt;symbol&gt;'</td>
<td>Display the value of the specified symbol.</td>
</tr>
<tr>
<td>'&lt;symbol&gt;=&lt;value&gt;'</td>
<td>Assign the &lt;value&gt; to the specified symbol or section.</td>
</tr>
</tbody>
</table>

**Examples**

```plaintext
>'sym = 1000
>'start = 8000
>'end = 'start +37E
>SYM
$00001000 sym
$00008000 start
$0000837E end
```
TCE: Dynamic Trace Capture Enable

**Command** | **Result**
--- | ---
ON TCE | Start trace acquisition. With TCE on, the DT, DTB, DTF and DRT commands work only in pause mode.
OFF TCE | Stop trace acquisition to allow examination of your trace memory. With TCE off, you can observe trace without stopping emulation.

Default: ON

**Comments**

This command is only available with the dynamic trace feature. Operation of the dynamic trace feature requires three steps:

1. Stop trace acquisition using OFF TCE.
2. Examine the trace using DT, DRT, DTB or DTF.
3. Restart trace acquisition using ON TCE.

While the target system is running, you must freeze the trace buffer before you can read trace memory.

While the OFF TCE command is in effect, the entire Event Monitor System is disabled. If an Event Monitor System condition is reached, the system will not recognize it or take the appropriate action.

You can toggle the TCE switch while in run mode so you can alternate between using the Event Monitor System and reading trace while running.
# TCT: Terminal Port Control

<table>
<thead>
<tr>
<th><strong>Command</strong></th>
<th><strong>Result</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>TCT</td>
<td>The terminal port becomes the controlling port.</td>
</tr>
</tbody>
</table>

**Comments**

This command, along with the **CCT** command, allows control to be switched between to two serial ports without powering down the ES 1800 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 3-4)
## TE: Timers

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON TE&lt;0,1,2&gt;</td>
<td>The specified PCB timer (0, 1 or 2) is active during pause mode.</td>
</tr>
<tr>
<td>OFF TE &lt;0,1,2&gt;</td>
<td>The specified PCB timer (0, 1 or 2) is not active during pause mode.</td>
</tr>
<tr>
<td></td>
<td>Default: OFF</td>
</tr>
</tbody>
</table>

### Comments

Timers 0 and 1 only apply to the 80186/188.

When the emulator transitions between pause and run modes, the settings of the TE switches determine whether the timer register values are read from or written to the physical PCB and whether the timer continues to run while the emulator is paused. The mode control word registers (MCW0, MCW1 and MCW2) and the timer count registers (TC0, TC1 and TC2) are affected by the switch setting.

#### Pause to Run Transition

When the emulator transitions from pause to run mode, the TE switch setting determines if the values of the MCW and TC registers in the emulator’s RAM image are loaded to the physical PCB.

- If the TE switch is OFF, the registers are loaded to the physical PCB. The value loaded into the MCW register determines whether or not the timer becomes active during run mode.
- If the TE switch is ON, the registers are not loaded to the physical PCB. This prevents the timer count register being overwritten by the old count value (this is undesirable if the timer was counting while the emulator was paused).

#### Run to Pause Transition

When the emulator transitions from run to pause mode, the current value of the MCW and TC registers are loaded from the physical PCB to the emulator’s RAM image of the CPU registers.

- If the TE switch is ON, no other action occurs and the timer continues to run while the emulator is paused.
- If the TE switch is OFF, the timer is disabled immediately after the transition to pause mode by clearing bit 15 of the mode control word register in the physical PCB.
You can modify timer registers while you are in pause mode, and, if OFF TE is specified, those values continue to be active when run mode is entered. Registers are modified using a `<register> = <value>` command.

The position of pod jumper JP4 determines when timers 0 and 1 are enabled for internal clocking.

The table below summarizes the effect of the timer switches.

### Effect of TE switches on Run/Pause Transitions

<table>
<thead>
<tr>
<th>Switch Setting</th>
<th>Pause to Run Transition</th>
<th>Run to Pause Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON</td>
<td>The emulator's RAM image of the specified timer register is <em>not</em> loaded to the physical PCB before entering run mode.</td>
<td>The value in the specified timer register is loaded into the emulator's RAM image of the CPU registers.</td>
</tr>
<tr>
<td>OFF</td>
<td>The emulator's RAM image of the specified timer register is loaded to the physical PCB just before running the target code.</td>
<td>The value in the specified timer register is loaded into the emulator's RAM image of the CPU registers. The timer is then disabled by clearing bit 15 of the appropriate mode control word register.</td>
</tr>
</tbody>
</table>
TGR: Send Trigger Signal

Command | Result
---------|---------

**WHE <events> THE TGR, <action> ;...**

If all of the conditions specified in the event portion of the WHEN/THEN clause are satisfied, 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.

**Comments**

The trigger signal is an output that is available from the BNC connector labelled TRIG on the back panel of the ES 1800 chassis and from pin 19 of the optional LSA pod.

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 Define location of UART.
>S1=RIO Look for read access.
>WHEN AC1 AND S1 THEN TGR When data is read, send trigger.
```

Determine the duration of a subroutine using the trigger pulse. The trigger pulse can be the input to a counter/timer or a scope. The duration of the subroutine can be determined from the pulse width displayed on the scope or the counter/timer readout.

```
>AC1=2500 Start of subroutine.
>AC1.2=AC1+38E End of subroutine.
>DC1.2=XXXX Detect any data pattern.
>WHEN AC1 THEN TGR, GRO 2 Go to group 2 when subroutine is entered.
>2 WHEN DC1 THEN TGR Trigger during all cycles while in group 2
>2 WHEN AC1 THEN GRO 1 Go back to group 1 when last instruction in subroutine is executed.
>RUN
R> Run mode prompt will appear.
```
TOC: Toggle Hardware Counter

**Command**

WHE <events> THE TOC, <action> ,...

**Result**

If all of the conditions specified in the event portion of the WHEN/THEN clause are satisfied, 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.

**Comments**

See the CNT action for a complete description of how the hardware counter works.
TOT: Toggle Trace

**Command**

WHE <events> THE TOT , <action> , ...

If all of the conditions specified in the event portion of the WHEN/THEN clause are satisfied, 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.

**Comments**

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.

<table>
<thead>
<tr>
<th>Previous Group</th>
<th>New Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No Trace Action Specified</td>
</tr>
<tr>
<td>No Trace specified</td>
<td>Trace all cycles</td>
</tr>
<tr>
<td>TRC</td>
<td>Trace all cycles</td>
</tr>
<tr>
<td>TOT OFF (not tracing)</td>
<td>Trace all cycles</td>
</tr>
<tr>
<td>TOT ON (tracing)</td>
<td>Trace all cycles</td>
</tr>
</tbody>
</table>

This table describes initial trace conditions.

<table>
<thead>
<tr>
<th>Action Specified</th>
<th>Trace Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>No trace</td>
<td>Trace all cycles</td>
</tr>
<tr>
<td>TRC</td>
<td>Trace only qualified TRC events</td>
</tr>
<tr>
<td>TOT</td>
<td>Trace nothing until TOT event</td>
</tr>
</tbody>
</table>

Alphabetical Command Reference 7-155
# TRA: 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.

**Comments**

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

In transparent mode the ES 1800 acts only as an interface between the two serial ports. The ES 1800 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 Section 4, “Serial Communications.”

**Examples**

```
>TRA
```

Enter transparent mode. Data entered at either port is transmitted directly to the other port.
TRC: Trace Events

**Command**

WHE <events> THE TRC , <action>, ...

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.

**Comments**

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.

<table>
<thead>
<tr>
<th>Previous Group</th>
<th>New Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>No Trace Action Specified</strong></td>
</tr>
<tr>
<td>No Trace specified</td>
<td>Trace all cycles</td>
</tr>
<tr>
<td>TRC</td>
<td>Trace all cycles</td>
</tr>
<tr>
<td>TOT OFF (not tracing)</td>
<td>Trace all cycles</td>
</tr>
<tr>
<td>TOT ON (tracing)</td>
<td>Trace all cycles</td>
</tr>
</tbody>
</table>
TRC: Trace Events

This table describes initial trace conditions.

<table>
<thead>
<tr>
<th>Action Specified</th>
<th>Trace Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nothing</td>
<td>Trace all cycles</td>
</tr>
<tr>
<td>TRC</td>
<td>Trace only qualified TRC events</td>
</tr>
<tr>
<td>TOT</td>
<td>Trace nothing until TOT event</td>
</tr>
</tbody>
</table>

Examples

Trace only a specific subroutine. Break at the end of the routine.

```plaintext
>AC1='Sub_start
>AC2='Sub_end
>WHEN AC1 THEN TOT
>WHEN AC2 THEN BRK
>RBK
R>
```

- Define beginning of subroutine.
- Define end of subroutine.
- Start tracing at beginning of subroutine.
- Break at end of subroutine.
- Run till breakpoint.
- Run mode prompt will appear.
## TST: Test Register

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>TST</td>
<td>Stop a repeating command. The test register is set to an expression in a command line. When it becomes zero, the repeat halts. The TST variable is set to all 1’s 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.</td>
</tr>
</tbody>
</table>

### Comments

See Section 4, “Repeat Operators,” for more detailed information.

### Examples

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

```
>STP;DT; TST=CS:IP-$C324
```
UPL: Upload Serial Data

Command Result

UPL <range. The ES 1800 formats and sends data to the computer port.

Comments

Data is transferred from the ES 1800 to a host system or other peripheral interfaced to the ES 1800 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 Section 4, "Serial Communications.")

Examples

For UNIX:

    cat ><filename>

For VMS:

    COPY TT: <filename>
    or
    TYPE SYS$INPUT/OUTPUT=<filename>

(Create or EDT are also acceptable.)

For DOS:

    COPY CON: <filename>

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

After all data has been uploaded and the ES 1800 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 Section 4, “Serial Communications.”
UPS: Upload Symbols

**Command**

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPS</td>
<td>All currently defined symbols and sections are sent to the computer port in Extended Tekhex format.</td>
</tr>
</tbody>
</table>

**Comments**

Extended Tekhex restricts the number and range of characters that can be used for symbol names. When formatting symbols for upload, the ES 1800 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)

When uploading to a file on a host system, enter transparent mode first and open a file to store the uploaded data records. (Review Section 4, “Serial Communications.”)

**Examples**

For UNIX:  
```
cat > <filename>
```

For VMS:  
```
COPY TT: <filename> or TYPE SYS$INPUT/OUTPUT=<filename>
```

(Create or EDT are also acceptable.)

For DOS:
```
COPY CON: <filename>
```

Next, type the transparent escape sequence and begin uploading.

After all data has been uploaded and the ES 1800 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 Section 4, “Serial Communications,” and Section 4, “Symbols.”
VBL: 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 BYM or WDM for more information on the default data length.

Examples

> VBL 0 TO 2000, 3F Verify that a range contains $3F.
$00000004 - $00, NOT $3F
$00000126 - $76, NOT $3F >
## VBM: Verify Block Move

### Command

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBM &lt;range&gt;,&lt;address&gt;</td>
<td>Verifies move of <code>&lt;range&gt;</code> to the new <code>&lt;address&gt;</code>. The current value of MMS specifies the relocation register used during the transfer.</td>
</tr>
<tr>
<td>VBM &lt;range&gt;,&lt;space&gt;,&lt;address&gt;</td>
<td>Verifies move of <code>&lt;range&gt;</code> to the new <code>&lt;address&gt;</code>. The <code>&lt;space&gt;</code> argument specifies the memory mode status used during the transfer.</td>
</tr>
<tr>
<td>VBM &lt;range&gt;,&lt;address&gt;,&lt;space&gt;</td>
<td>Verifies move of <code>&lt;range&gt;</code> to the new <code>&lt;address&gt;</code>. The range is read from the space specified in the MMS register. The block is written to the <code>&lt;space&gt;</code> specified in the argument following the address.</td>
</tr>
<tr>
<td>VBM &lt;range&gt;,&lt;space&gt;,&lt;address&gt;,&lt;space&gt;</td>
<td>Verifies move of <code>&lt;range&gt;</code> to the new <code>&lt;address&gt;</code>. The range is read from <code>&lt;space&gt;</code> specified in the argument following the range. The block was written to the <code>&lt;space&gt;</code> specified in the argument following the address.</td>
</tr>
</tbody>
</table>

### Comments

This command is valid only in pause mode.

Verifies that a non-overlapping block move was successful.
**VFO: Verify Overlay Memory**

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>VFO &lt;range&gt;</td>
<td>Compare the specified range in the target memory to the same range in the overlay memory.</td>
</tr>
<tr>
<td></td>
<td>If there are no differences between the data in the overlay and target, the emulator prompts you for the next command.</td>
</tr>
<tr>
<td></td>
<td>If there are any differences, the address of each difference displays &lt;ADDRESS&gt; = XX NOT YY</td>
</tr>
<tr>
<td></td>
<td>XX denotes the data present in overlay memory. YY is the data at that location in the target system memory.</td>
</tr>
</tbody>
</table>

**Comments**

This command is valid only in pause mode.
Refer also to Section 4, “Mapping Overlay Memory.”

**Examples**

```
> VFO 80000 LEN 7FFFVerify overlay load using hex addresses.
> VFO 'BOOT_RANGEVerify overlay load using symbols.
```
VFY: Verify Serial Data

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
</table>
| 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 MAP).

This command is also useful for determining differences between object files. Follow instructions for downloading a file in Section 4 “Downloading to Target or Overlay Memory.”
### WAI: Wait Until Emulation Break

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAI</td>
<td>Delays executing the specified command until emulation is broken.</td>
</tr>
</tbody>
</table>

**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 ES 1800 out of run mode. When this happens, use the system reset character to reset the system. (<ctrl-z> default, can be changed with SET).

After a reset, the delayed command is lost from the input buffer.

**Examples**

The ES 1800 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
```

**Run to breakpoint, wait til emulation stops and disassemble previous page of trace.**

The ES 1800 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
```

**Run to breakpoint, wait til emulation stops and display string at address BX.**
**WDM: Set Global Data Length**

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYM</td>
<td>Set the global data length to byte mode.</td>
</tr>
<tr>
<td>WDM</td>
<td>Set the global data length to word mode.</td>
</tr>
<tr>
<td></td>
<td>Default: <strong>BYM</strong> - byte mode</td>
</tr>
</tbody>
</table>

**Comments.**

The global data length determines whether memory commands use byte or word data lengths. If byte mode is set and you enter a word value as a command parameter, only the least significant byte is used as the command parameter. If word mode is set and you enter a byte parameter, the high byte is padded with a zero.

The global data length affects the following commands.

**Commands Affected by Global Data Length**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMO</td>
<td>block move data in memory</td>
</tr>
<tr>
<td>DB</td>
<td>display block of memory</td>
</tr>
<tr>
<td>FIN</td>
<td>find data pattern in memory</td>
</tr>
<tr>
<td>FIL</td>
<td>fill memory with data pattern</td>
</tr>
<tr>
<td>LOV</td>
<td>load overlay memory from target</td>
</tr>
<tr>
<td>M</td>
<td>memory mode</td>
</tr>
<tr>
<td>MIO</td>
<td>I/O mode</td>
</tr>
<tr>
<td>SF 0-9,11,12</td>
<td>special functions: RAM tests and scope loops</td>
</tr>
<tr>
<td>VBL</td>
<td>verify data pattern in memory</td>
</tr>
<tr>
<td>VFO</td>
<td>verify overlay memory with target memory</td>
</tr>
</tbody>
</table>
**WDM: Set Global Data Length**

**Examples**

The following example demonstrates how the global data length affects the **FIL** and **DB** commands.

```
>BYM
>FIL 0 LEN 10,123
>DB 0 LEN 10
```

Set byte mode. 
Fill the range with 123.
High byte is truncated.

```
000000 23 23 23 23 23 23 23 23 23 - 23 23 23 23 23 23 23 23 23 23
```

```
>
```

```
>WDM
>FIL 0 LEN 10,3F
>DB 0 LEN 10
```

Set word mode.
Fill the range with 3F.
Pattern is padded with zero.

```
000000 003F 003F 003F 003F - 003F 003F 003F 003F
```

>
WHEN: Begin WHEN/THEN Statement

**Command**

\[ \text{WHE} <\text{events}> \text{ THE} <\text{action}>,<\text{action}>..., \]

Perform specified actions when the events are reached.

**Comments**

You can define an event to be some combination of address, data, status, count, and Logic State Analyzer pod 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 (AND and OR).

\[ \text{WHEN ACl AND NOT DCl THEN BRK} \]

means break whenever any data pattern other than that in DCl is written to an address in ACl.

AND and OR can be used to form more restrictive event definitions. AND terms have higher precedence than OR terms. For example:

\[ \text{WHEN ACl AND DCl OR DC2 THEN BRK} \]

is the same as

\[ \text{WHEN ACl AND DCl THEN BRK} \]
\[ \text{WHEN DC2 THEN BRK} \]

If you are looking for two different data values at an address, you would use

\[ \text{WHEN ACl AND DCl OR ACl AND DC2 THEN BRK} . \]

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 WHEN/THEN statements for clarity.

\[ \text{WHEN ACl AND S1 OR AC2 AND S2 THEN BRK} \]

is the same as

\[ \text{WHEN ACl AND S1 THEN BRK} \]
\[ \text{WHEN AC2 AND S2 THEN BRK} . \]
### X: Exit Memory, I/O Modes, and Line Assembler

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Exit memory or I/O mode.</td>
</tr>
</tbody>
</table>
ES LANGUAGE

Structure of the ES Language

The command language used to control the ES 1800 emulator is a formal language. Once you understand the basic concepts of this language, you can apply the full debugging power of the ES 1800. An overview of the structure of the ES language (ESL) is presented in the accompanying table. A more detailed description of the language elements, the help menus, prompts, special operating modes, and ES language error messages are also included in this section.

Items in angle brackets (< >) are mandatory and must be entered as part of the command. Items shown in square brackets ([ ]) are optional. Do not type the angle or square brackets when typing a command.

If the ESL command interpreter detects an illegal statement, it beeps and places a question mark under the command line at the position the error was detected. Entering a ? following an error will cause the appropriate error message to be displayed.

ES Language syntax

<table>
<thead>
<tr>
<th>Language Element</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Line</td>
<td>&lt;RETURN&gt;</td>
</tr>
<tr>
<td>[Repeat] Command Statement [:Cmd Statement] ...</td>
<td></td>
</tr>
<tr>
<td>Single Character Instant Command</td>
<td></td>
</tr>
<tr>
<td>Repeat</td>
<td></td>
</tr>
<tr>
<td>&lt;*&gt; *STP;DT</td>
<td>*9 STP;DT</td>
</tr>
<tr>
<td>&lt;*&gt;&lt;Repeat limit&gt;</td>
<td></td>
</tr>
<tr>
<td>Repeat Limit:</td>
<td></td>
</tr>
<tr>
<td>Decimal number only (1 to $2^{32}$ -1)</td>
<td>87651234</td>
</tr>
<tr>
<td>Command Statement</td>
<td></td>
</tr>
<tr>
<td>Command Mnemonic</td>
<td>DTB</td>
</tr>
<tr>
<td>Command Mnemonic &lt;Expression&gt;</td>
<td>MM CS:IP +4</td>
</tr>
<tr>
<td>Command Mnemonic &lt;Expression List&gt;</td>
<td>SET #20,#14</td>
</tr>
<tr>
<td>Assignment Command</td>
<td>CS = 0FA9</td>
</tr>
<tr>
<td>Expression</td>
<td></td>
</tr>
<tr>
<td>Event Monitor System Control Statement</td>
<td></td>
</tr>
<tr>
<td>WHE AC1 THE BRK</td>
<td></td>
</tr>
</tbody>
</table>
Structure of the ES Language

**Language Element**

**Example**

**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 chars.>[1 or more dec. chars.]  
  **ASM**

**Expression**

- [Unary Operator] Ivalue
- Ivalue <Operator> Expression
- <@> Expression
- <(> Expression <)>  
- Nvalue <-> Nvalue

  **Ivalue:**
  - Symbol
  - Nvalue
  - Symbol:
    - <1 or more printable chars.> <sp or cr>

  **Nvalue:**
  - Number
  - Register Name

  **Register Name:**
  - <1 - 3 alpha chars.>[0 - 2 dec. digits]

  **Number:**
  - [Base]<1 or more digits>
    - Base:
      - <%> (binary)
      - <> (octal)
      - <#> (decimal)
      - <$> (hexadecimal)

  **Expression List**

  Expression <,> Expression [,Expr. list]...  
  1,CS:IP,2+2,-6
Structure of the ES Language

*Language Element*

**Assignment Command**

\[ \text{Svalue} \Leftrightarrow \text{Expression} \]
\[ \langle @ \rangle \text{Expression} \Leftrightarrow \text{Expression} \]

\[ \text{Svalue:} \]
- Symbol
- Register Name

**Event Monitor System Control Statement**


**Group:**
- <1>
- <2>

**BRK**
- <3>
- <4>

**Event:**
- [Disjunctive] <Event Comparator>
- Event <Conjunctive> <Event>

**Disjunctive:**
- <NOT>

**Event Comparator**
- <AC1>[.Group]
- <AC2>[.Group]
- <DC1>[.Group]
- <DC2>[.Group]
- <S1>[.Group]
- <S2>[.Group]
- <CTL>[.Group]
- <LSA>[.Group]

**Conjunctive:**
- <AND>
- <OR>

**Action List**
- <Action>[,Action]...

**Example**

\[ \text{IP} = @0FFFF0 \]
\[ @SS:SP = CS:IP \]

\[ \text{Test_result} \]
\[ \text{MMP} \]

\[ \text{WHE AC1 THE BRK} \]

\[ 2 \ 	ext{WHE AC1 THE} \]

\[ \text{NOT AC1} \]

\[ \text{DC2 OR NOT AC1} \]

\[ \text{AC1.3} \]

\[ \text{CTL.4} \]

\[ \text{TRC, TGR, FSI} \]
**Structure of the ES Language**

<table>
<thead>
<tr>
<th>Language Element</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Action:</strong></td>
<td></td>
</tr>
<tr>
<td>&lt;BRK&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;TRC&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;TOT&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;CNT&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;TOC&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;RCT&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;TGR&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;FSI&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;GRO Group&gt;</td>
<td>GRO 3</td>
</tr>
<tr>
<td><strong>Unary Operator</strong></td>
<td></td>
</tr>
<tr>
<td>&lt;ABS&gt;</td>
<td>ABS GD3</td>
</tr>
<tr>
<td>&lt;!&gt;</td>
<td>!0AA</td>
</tr>
<tr>
<td>&lt;&lt;-</td>
<td>-3</td>
</tr>
<tr>
<td><strong>Operator</strong></td>
<td></td>
</tr>
<tr>
<td>Mul.op</td>
<td></td>
</tr>
<tr>
<td>Add.op</td>
<td></td>
</tr>
<tr>
<td>Shft.op</td>
<td></td>
</tr>
<tr>
<td>&lt;&amp;&gt;</td>
<td>GD4 &amp; OFF</td>
</tr>
<tr>
<td>^&gt;</td>
<td>DC2.3 ^ OFF00</td>
</tr>
<tr>
<td>Mul.Op</td>
<td></td>
</tr>
<tr>
<td>&lt;*&gt;</td>
<td>2 * 3</td>
</tr>
<tr>
<td>&lt;/&gt;</td>
<td>0FAC / %01001</td>
</tr>
<tr>
<td>&lt;MOD&gt;</td>
<td>GD5 MOD 7</td>
</tr>
<tr>
<td>Add.op</td>
<td></td>
</tr>
<tr>
<td>&lt;+&gt;</td>
<td>GRO + IP</td>
</tr>
<tr>
<td>&lt;-&gt;</td>
<td>@(SS:SP - 4)</td>
</tr>
<tr>
<td>Shft.op</td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;&lt;&gt;</td>
<td>DC1 &lt;&lt; 3</td>
</tr>
<tr>
<td>&gt;&gt;&gt;&gt;</td>
<td></td>
</tr>
</tbody>
</table>
Notes on ESL

Command Line

A command line is created by entering one or more characters after any of the ESL 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 Macros in Section 4, or _ in Section 7).

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> redispays the current line (useful for hardcopy terminals).

Repeat

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

Command Statement

There are several special modes in which the normal command statement rules do not apply. In memory mode entering a <return> on an empty line causes the next location to be read. Entering a value followed by <return> will cause that value to be written to memory. I/O mode, the memory disassembler, and the main help menu all have special modes which prevent 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 (/) will cause the previously entered command line to be repeated.
This example single steps three times.

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

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

A ? entered at any other time (i.e., not after an error), causes a two-page help menu to be displayed. A <return> moves you from the first page to the second. Any other character terminates the help menu.

Command Mnemonics

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

Expressions

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 ES 1800 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.

\[@(GD0 +3) = IP + \#100 \times (DX >> 4) + 0AF34\]

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:

\[\text{'interrupt} + 1A6\]

by itself will add 1A6 to the current value of the symbol interrupt and display the result. If you don’t 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. \(@\ USP\) is a simple way to read something from the stack pointer. It is legal to have multiple indirections, e.g., \(@@\ GR0 = @@(USP + 6)\). Byte mode and word mode affect the length of data transferred to or from the target by the \(\@\) operator. (See the \(\text{BYM}\) and \(\text{WDM}\) commands in Section 7 for more information on BYT/WRD modes.)

The \(:\) operator mimics the arithmetic combination of segment and pointer registers in the 80186/88 and 80C186/C188 microprocessors. 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., CS:IP is fine but CS:(IP+3) is illegal.

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

Certain combinations of expression types and operators are illegal or have complex results. See the table “Results of Dyadic Operator Combinations.”

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 ES 1800 emulator, you can use symbolic references. Every symbol must begin with a single quote ('). Symbols are composed of 1 to 64 printable characters followed by a space or <return>. Symbols can be used anywhere a register or a number is used, with the exceptions that symbols are not valid with the colon operator or the repeat (*) operator.

Numbers

The ES 1800 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 the DFB command in Section 7 for more information in changing the default base register.

There are three different types of numbers.
1. An integer is a 32-bit signed value.
2. A don’t care is a 32-bit value with a 32-bit mask. For each-bit set in the mask, the corresponding bit position in the value is ignored during Event Monitor comparisons. Don’t cares can be entered in two ways. 1234 DC OFF0 is explicit. 1XX4 is equivalent to 1FF4 DC OFF0. Don’t cares are useful for setting the Event Monitor System Event Comparators (see the Event Monitor System in Section 4 for more information.)

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

Ranges, don’t cares, and integers are not generally interchangeable. Certain registers can only hold certain data types. All registers can hold integers. Address type registers cannot be loaded with don’t care values. Status and data registers cannot be loaded with range values. See “Registers” in Section 4 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.
Notes on ESL

(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**

Registers are grouped into three types: integer only, don't care, and range. Any register can be assigned an integer value. Don't care registers can be loaded with don't care values or integers but not ranges. Range registers can be loaded with integers or ranges but not don't care values. See Registers in Section 4 for a list of all registers and their data types.

**Indirection Operator**

The indirection operator `@` 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 or word.

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

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

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

In the first example the indirection operator is applied to `GD4`. The command interpreter accesses the target system location pointed at by `GD4`, adds six to the value stored there, and displays the final results.
In the second example, the ES 1800 displays the value stored in the sixth location above the address pointed to by GD4.

The indirection operator can be used to write values to memory-mapped I/O without causing a read after write. Memory mode always performs memory reads. This may be unacceptable for certain hardware configurations. To store values without entering memory mode, use:

```plaintext
>@[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 event comparator names. For example: 

```plaintext
3 WHEN AC1 THEN BRK
```

is functionally the same as

```plaintext
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, 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
```

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 $FFFFFFFE. 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. Signed numbers must be enclosed in parentheses. For example, \[
\text{ABS}(-3) + 2 \quad \text{returns} \quad 5.
\]

! is a logical NOT operator and complements 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. 
\[-2+3 \quad \text{is} \quad 1.\]

**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

- **Shift.op:**
  - >> Right shift
  - << Left shift
  - & Logical AND
  - ^ Logical OR

**Modulo (MOD)**

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

\[
>29 \quad \text{MOD} \quad 4

equals \quad 1
\]

\[
>38 \quad \text{MOD} \quad 6
\]

result \quad = \quad 2
**Results of Single-Argument Operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Argument</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Integer</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td>DC</td>
<td>Don’t care bits are not affected</td>
</tr>
<tr>
<td></td>
<td>IRA</td>
<td>Complement (IRA becomes XRA)</td>
</tr>
<tr>
<td>ABS</td>
<td>Integer</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td>DC</td>
<td>Don’t care bits are not affected</td>
</tr>
<tr>
<td></td>
<td>IRA</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td>XRA</td>
<td>Invalid</td>
</tr>
<tr>
<td>-</td>
<td>Integer</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td>DC</td>
<td>Don’t care bits are not affected</td>
</tr>
<tr>
<td></td>
<td>IRA</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td>XRA</td>
<td>Invalid</td>
</tr>
<tr>
<td>@</td>
<td>Integer</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td>DC</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td>IRA</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td>XRA</td>
<td>Invalid</td>
</tr>
</tbody>
</table>
### Results of Dyadic Operator Combinations

<table>
<thead>
<tr>
<th>Left Hand Expression</th>
<th>Right Hand Expression</th>
<th>Operator</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>Integer</td>
<td>*/ MOD</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&amp; ^</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;&lt; &gt;&gt;</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ -</td>
<td>Valid</td>
</tr>
<tr>
<td>Integer</td>
<td>Don’t care</td>
<td>MOD</td>
<td>Illegal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*/</td>
<td>Don’t care bits are passed to the left hand argument.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&amp; ^</td>
<td>Don’t care bits are passed to the left hand argument.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;&lt; &gt;&gt;</td>
<td>Don’t care bits are passed to the left hand argument.</td>
</tr>
<tr>
<td>Integer</td>
<td>IRA, XRA</td>
<td>*/ MOD</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&amp; ^</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;&lt; &gt;&gt;</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ -</td>
<td>Don’t care bits are ANDed.</td>
</tr>
<tr>
<td>Don’t care</td>
<td>Don’t care</td>
<td>*/ MOD</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&amp; ^</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;&lt; &gt;&gt;</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ -</td>
<td>Don’t care bits are ANDed.</td>
</tr>
<tr>
<td>Don’t care</td>
<td>Integer</td>
<td>*/ MOD</td>
<td>Don’t care bits are kept.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&amp; ^</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;&lt; &gt;&gt;</td>
<td>Don’t care-bit positions are shifted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ -</td>
<td>Don’t care bits are kept.</td>
</tr>
<tr>
<td>IRA, XRA</td>
<td>Integer</td>
<td>*/ MOD</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&amp; ^</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;&lt; &gt;&gt;</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ -</td>
<td>The end points of the range will be altered by the value of the integer expression.</td>
</tr>
</tbody>
</table>
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 information is described below.

Software Switches

Enter either ON or OFF to display the current settings and definitions of all software switches, (see ON in Section 7).

Communications Set-up

Enter SET to display the current configuration settings and possible values (see SET in Section 7).

Special Diagnostic Functions

Enter SF to display a list of the available special functions (RAM/ROM tests, scope loops, etc.) (see SF in Section 7).
First Page of Help Menu

>?

RUN/EMULATION:
  RUN/RNV - RUN/RUN WITH NEW VECTORS
  RBK/REB - RUN TO BREAKPOINT/WITH VECTORS
  WAIT - WAIT UNTIL EMULATION BREAK

STP - SINGLE STEP/STOP

RST - RESET TARGET SYSTEM

TRACE HISTORY:
  DTB/DTF - DISASSEMBLE PAGE BACK/FORWARD
  DRT - DISPLAY PAGE RAW TRACE

MEMORY-REGISTER COMMANDS:
  DR - DISPLAY ALL CPU REGISTERS
  DB X TO Y - DISPLAY BLOCK
  FILL X TO Y, Z - FILL BLOCK WITH Z
  EMO 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 = DC, DAT
  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
  ASM (X) - IN LINE ASSEMBLER

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 DISCRETE 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
Log In Banner

After initial power on, the log in banner should appear on your console screen. After a reset, the first three lines of the banner appear on your screen.

**Satellite Emulator**

The microprocessor type is that of the target system.

**VX.XX**

The version number reflects the released version of the ES language software for the emulator.

**USER = ___ SW = ___**

The user number and software number (SW) indicate the positioning of the thumbwheel switch on the ES 1800 MCB controller board (page 3-4).

**AVAILABLE OVERLAY**

The amount of overlay memory indicated depends on the amount installed in the system. This can be 128K, 256K, 512K, 1M or 2M of memory.

**> No Target VCC**

The console screen displays a NO TARGET VCC (see Appendix A) when you are not connected to a target system.

A `<ctrl-z>` clears this display message and returns the system to the log in banner for reentry of an input command.

**NOTE**

Refer to Section 1 and 4 for using the ES 1800 emulator without a target system.

**Prompt**

The pause mode prompt `>` indicates that the ES 1800 is not running, is in a pause mode and is ready to receive instructions. Make sure that the `>` shows before you enter any command.

If the `>` does not appear after the log in banner: turn off the equipment, check the connections, and then repeat the power-up sequence.

---

**ES Language**  
8-19
Check for proper connection of the cable between the terminal and the ES 1800.

Check the cable connecting the pod to the ES 1800. Is it completely secured?

Check to see if the pod probe package is completely plugged into the target system.

If the unit has just been shipped, one or more of the boards may have become loose in the ES 1800 chassis. Check for loose boards.

If an error message appears, refer to Appendix A for an explanation.
Prompts

Different prompts are displayed depending on the current operating mode of the ES 1800.

> The standard, or pause mode prompt from ESL consists of a space character followed by a right arrow.

$12345678 $00 $00 $0000 $0000 R>

During emulation, the run mode prompt is displayed. Most ESL commands are still valid.

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 (R>) may also be present during memory mode.

**** 8086/88/186/188 LINE ASSEMBLER ****
CSEG=0000
0100 >

The line assembler displays a 16-bit address prompt. This prompt contains an R if you are assembling during emulation.

IO:$1200 >
IO:$1200 $00 >
IO:$1200 $0000 >
IO:$1200 R>
IO:$1200 $00 R>
IO:$1200 $0000 R>

In I/O mode, the prompt includes the I/O address. The data is included when a <return> is entered as the only character on the line. The data field is affected by byte and word mode. 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 and 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.

You can temporarily override the byte and word address and data display prompts by keying in the dot operators (.B and .W) after a command. For example: DB.B means a block of memory is displayed in byte mode. DB.W means a block of memory is displayed in word mode.

**Line Assembler**

The 80186/188/C186/C188 line assembler has a single 16-bit address prompt. Exit by entering an X or the END directive.

**Memory Disassembler**

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

**Memory Mode**

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

**I/O Mode**

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

**Transparent Mode**

No characters are generated by the ES 1800. Exit by entering the two character escape sequence (default is <esc> <esc>), or reset (default <ctrl-z>).

**Special Functions**

Many diagnostic functions are designed to run continuously. The message from the function will inform you to enter the reset character (default is <ctrl-z>) to terminate the function.
| Repeating Command Lines | It is easy to inadvertently create an indefinitely repeating command that does not display anything. Terminate such commands with the reset character (default is `<ctrl-z>`). |
Special Characters

These special characters can be changed through the SET menu. See SET in Section 7 for information on how to change a special character.

- `<delete>,<backspace>`: Either character deletes a character just entered on a command line.
- `<ctrl-x>`: Deletes an entire command line. Also stops a command repeated with * without resetting emulator.
- `<ctrl-r>`: Redisplays the current command line (for hardcopy terminals).
- `<ctrl-z>`: The default reset character. `<ctrl-z>` resets the emulator, stops emulation and/or clears an error condition. It does not clear or update emulator registers. It is also used to terminate certain diagnostic functions. `<ctrl-z>` terminates an indefinitely repeating command.
- `<esc><esc>`: The default transparent mode escape sequence, used to terminate transparent mode.
- `<ctrl-s>`: The XOFF character. When issued from the keyboard, the screen display stops scrolling, allowing you to view the information.
- `<ctrl-q>`: The XON character. Restarts the screen display after an XOFF is issued.
Errors

The ES 1800 software generates two basic types of error messages. ES language syntax and operational errors in a command line are indicated by a beep (BEL code). The next line displayed contains a single \? underneath, and usually just after, the place in your command line that caused the error. At the point the error is detected, the remainder of the command line is discarded. For example, the **DRT** command is invalid during emulation:

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

The **RBK** command was executed, but the **DR** command was not. Whenever you see an error message of this type, you can enter a single ?. The ES 1800 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 ES 1800 can be used. The error messages are quite explicit, such as

**NO TARGET CLOCK** or **RESET ASSERTED**.

Target hardware error messages are explained in Appendix A.
ES Language Error Messages

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

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 setup is the same as the format of the received data. Refer to download command (DNL) for error handling during computer control.

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

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 Corporation 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).
ES Language Error Messages

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. Examples of invalid ranges are 6 to 1 and 10 to 5.

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 ES 1800 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. The number of sentences cannot exceed the available RAM in ESL. This is different for each of the microprocessors supported.

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.
ES Language Error Messages

42  ILLEGAL SETUP SET VALUE. Consult the SET menu for legal values (see SET in Section 7).

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

44  INTERNAL ERROR . . . NULL SHIFTER FILE. Contact Customer Service.

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 ES 1800 has a one year warranty. Contact Customer 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. Check command argument list.

62  CANNOT STORE THIS VARIABLE DURING EMULATION. Must be in pause mode.

63  ILLEGAL ARGUMENT TYPE.
ARGUMENT TOO LARGE. Caused by entering DRT argument that includes numbers greater than #2045.

ILLEGAL RANGE.

STATUS CONSTANTS CANNOT BE ALTERED. System constants (i.e., BYT, OVL) cannot be assigned values.

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

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

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

UNKNOWN EMULATOR ERROR. Call Applied Microsystems.

INCOMPATIBLE EEPROM DATA. Previous data saved to EEPROM was not from an 8018X or 80C18X ES 1800 system.

COMMAND INVALID DURING EMULATION. Must be in pause mode.

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

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

TOO MANY SYMBOLS. Symbols exceeded available RAM. Purge symbols before downloading again.

SYMBOL OR SECTION PREVIOUSLY DEFINED. An attempt was made to redefine an existing symbol or section. Section definitions cannot overlap. Symbols should be purged before downloading.

SYMBOL NAME IN USE. Symbol name cannot be used more than once. You must delete a section before assigning it a new value.

TYPE CONFLICT WITH DEFINED SYMBOL. Please refer to Extended Tekhex specification, in Appendix B.

SECTION TABLE FULL. Too many symbolic section names have been defined.

INVALID ARGUMENT SIZE. Operand doesn't fit into destination register.

INVALID ADDRESSING MODE.

ARGUMENT OUT OF RANGE. Usually caused by reference to a "FAR" location without declaring "FAR."

INVALID TRAP VECTOR NUMBER.

INVALID CONTROL REGISTER.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>94</td>
<td>ARGUMENT NOT SYMBOLIC. Requires a symbolic argument.</td>
</tr>
<tr>
<td>255</td>
<td>UNKNOWN ERROR.</td>
</tr>
</tbody>
</table>
Error messages are divided into 5 categories:

1. Target hardware
2. Emulator hardware
3. Target software
4. ESL (see Section 8)
5. Software debugger (see appropriate software manual)

Within this section, errors are arranged in alphabetical order by category.

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 .7 microseconds or longer, and no other error conditions are found. If your target HALT waits for interrupts for longer than this, you can change the number of milliseconds by changing the value of the BTO register.

When no ALEs 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.

Certain operating systems will cause this error message to occur. There is a jumper on the pod board which, when set, will ignore the NO BUS CYCLES error. See Appendix C for more information.
Target Hardware Error Messages

**No Clock**
8018X microprocessors must have a clock frequency within the range of 1.2 MHz to 9 MHz, and 80C18X microprocessors must be within 0.5 MHz to 16 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 ES 1800 is powered up (see CK in section 7).

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.

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.

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 its internal memory space. 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.

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.

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.
Target Hardware Error Messages

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 STOP 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.
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 B-1: Specifications for MOS Technology Data Files

Copyright 1983, Data I/O Corporation; reprinted by permission.

INPUT

<table>
<thead>
<tr>
<th>DATA RECORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>:</td>
</tr>
<tr>
<td>B C</td>
</tr>
<tr>
<td>AAAA</td>
</tr>
<tr>
<td>H H</td>
</tr>
</tbody>
</table>

| 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 |
| B C | Byte Count. BC = 00 in End of File Record |
| R R R R | Record Count |
| CCCC | Checksum |

OUTPUT

<table>
<thead>
<tr>
<th>NOTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Number of bytes per record is variable. See Table 3.1</td>
</tr>
<tr>
<td>2) Each line ends with nonprinting line feed, carriage return and nulls.</td>
</tr>
</tbody>
</table>

| 2 Hex Characters = 1 byte |
| Data Records |

LEGEND

- Start Character
- Byte Count (BC > 00 in Record, BC = End of File Record)
- Address Field
- Checksum of Record
- Record Count
- Two Hexadecimal Digits (0:9, A:D)
### Motorola Exorcisor Format

**Figure B-2: Specifications for Motorola Exorciser/16-BM Data Files**

Copyright 1983, Data I/O Corporation; reprinted by permission.

<table>
<thead>
<tr>
<th>INPUT DATA RECORD</th>
<th>SIGN ON RECORD OPTIONAL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S 1</strong></td>
<td><strong>S 0</strong></td>
</tr>
<tr>
<td><strong>START CHARACTERS</strong></td>
<td><strong>Start characters of sign on record. Except for start characters S0 record has same format as data record</strong></td>
</tr>
<tr>
<td><strong>BC</strong></td>
<td><strong>BC</strong></td>
</tr>
<tr>
<td><strong>Byte Count. The number of data bytes plus 3 (1 for checksum and 2 for address) in hexadecimal notation</strong></td>
<td></td>
</tr>
<tr>
<td><strong>AAAA</strong></td>
<td><strong>AAAA</strong></td>
</tr>
<tr>
<td><strong>Address of first data byte in record. AAAA in hexadecimal notation only</strong></td>
<td></td>
</tr>
<tr>
<td><strong>HH</strong></td>
<td><strong>HH</strong></td>
</tr>
<tr>
<td><strong>One data byte in hexadecimal notation</strong></td>
<td></td>
</tr>
<tr>
<td><strong>CC</strong></td>
<td><strong>CC</strong></td>
</tr>
<tr>
<td><strong>Checksum. One's complement of binary summation of preceding bytes in record (including byte count, address and data bytes) in hexadecimal notation</strong></td>
<td></td>
</tr>
<tr>
<td><strong>S 1</strong></td>
<td><strong>S 1</strong></td>
</tr>
<tr>
<td><strong>(Beginning of next record)</strong></td>
<td></td>
</tr>
</tbody>
</table>

**LEGEND**

- **S0**: Optional Record Start Characters
- **S1**: Start Characters
- **BC**: Byte Count
- **[(Date Bytes/Record + 3)]**: Address of First Data Byte
- **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

- 2 Hex characters 1 byte
- Data Records

- S1BAAAAHHH
- S1BAAAAHHH
- S1BAAAAHHH
- S1BAAAAHHH
- S1BAAAAHHH
- S9BAAAAHHH

---

**Serial Data Formats** B-3
Intel Intellec Format

Intel Intellec Format

Figure B-3: Specifications for Intel Intellec/8/MDS Data Files/

Copyright 1983, Data I/O Corporation; reprinted by permission.

INPUT

<table>
<thead>
<tr>
<th>DATA RECORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>::</td>
</tr>
<tr>
<td>START CHARACTER</td>
</tr>
<tr>
<td>BC</td>
</tr>
<tr>
<td>Byte Count. The hexadecimal number of data bytes in the record</td>
</tr>
<tr>
<td>AAAA</td>
</tr>
<tr>
<td>Address of first data byte in record. AAAA in hexadecimal notation only</td>
</tr>
<tr>
<td>TT</td>
</tr>
<tr>
<td>Record Type (00)</td>
</tr>
<tr>
<td>HH</td>
</tr>
<tr>
<td>One data byte in hexadecimal notation</td>
</tr>
</tbody>
</table>

END OF FILE RECORD

| ::          |
| START CHARACTER |
| BC          |
| Byte Count. BC = 00 in End of File Record |
| AAAA       |
| Address |
| TT         |
| TT Record Type (01) |

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

Number of bytes per record is variable. See Table 3.1.

Hex characters: 1 byte

Data Records

00

Serial Data Formats
### Signetics/Absolute Object File Format

**Figure B-4: Specifications for Signetics/Absolute Object Data Files**

Copyright 1983, Data I/O Corporation; reprinted by permission.

<table>
<thead>
<tr>
<th>INPUT DATA RECORD</th>
<th>END OF FILE RECORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAAA</td>
<td>START CHARACTER</td>
</tr>
<tr>
<td>AAAAA</td>
<td>Address</td>
</tr>
<tr>
<td>BC</td>
<td>Byte Count</td>
</tr>
<tr>
<td>BC</td>
<td>Byte Count. BC = 00 in End of File Record</td>
</tr>
<tr>
<td>AC</td>
<td>Address Check. Every byte is exclusive OR with the previous byte, then rotated left one bit.</td>
</tr>
<tr>
<td>HH</td>
<td>One data byte in hexadecimal notation</td>
</tr>
<tr>
<td>DC</td>
<td>Data Check. Every byte is exclusive OR with the previous byte, then rotated left one bit.</td>
</tr>
<tr>
<td>DC</td>
<td>Data Check. Checksum of data in record</td>
</tr>
</tbody>
</table>

**LEGEND**
- AAAAA = Address of first data byte in record. AAAAA in hexadecimal notation only
- BC = Byte Count. The hexadecimal number of data bytes in the record
- AC = Address Check. Every byte is exclusive OR with the previous byte, then rotated left one bit
- HH = One data byte in hexadecimal notation
- DC = Data Check. Checksum of data in record

**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

**Serial Data Formats**
Tektronix Hexadecimal Format

Figure B-5: Specifications for Tektronix Hexadecimal Data Files

Copyright 1983, Data 1/0 Corporation; reprinted by permission.
Extended Tekhex Format

Copyright 1983, Tektronix; reprinted by permission

Extended Tekhex uses three types of message blocks:

1. The data block contains the object code.
2. The symbol block that contains information about a program section and the symbols associated with it. This information is only needed for symbolic debug.
3. The termination block contains the transfer address and marks the end of the load module.

NOTE

Extended Tekhex has no specially defined abort block. To abort a formatted transfer, use a Standard Tekhex abort block.

Each block begins with a six-character header field and ends with an end-of-line character sequence. A block can be up to 255 characters long, not counting the end-of-line character. The header field has the format shown in the following table.

<table>
<thead>
<tr>
<th>Item</th>
<th>Number of ASCII Characters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>1</td>
<td>A percent sign specifies that the block is in Extended Tekhex format.</td>
</tr>
<tr>
<td>Block Length</td>
<td>2</td>
<td>The number of characters in the block: a two-digit hex number. This count does not include the leading % or the end-of-line.</td>
</tr>
</tbody>
</table>
| 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 The following table gives the values for all characters that may appear in Extended Tekhex message blocks. |
Extended Tekhex Format

Character Values for Checksum Computation

<table>
<thead>
<tr>
<th>CHARACTERS</th>
<th>VALUES (DECIMAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..9</td>
<td>0..9</td>
</tr>
<tr>
<td>A..Z</td>
<td>10..35</td>
</tr>
<tr>
<td>$</td>
<td>36</td>
</tr>
<tr>
<td>%</td>
<td>37</td>
</tr>
<tr>
<td>. (period)</td>
<td>38</td>
</tr>
<tr>
<td>_ (underscore)</td>
<td>39</td>
</tr>
<tr>
<td>a..z</td>
<td>40-65</td>
</tr>
</tbody>
</table>

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 OKLUDGESTARTSHERE. 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

<table>
<thead>
<tr>
<th>Item</th>
<th>Number of ASCII Characters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>6</td>
<td>Standard header field Block Type = 6</td>
</tr>
<tr>
<td>Load Address</td>
<td>2 to 17</td>
<td>The address where the object code is to be loaded: a variable-length number.</td>
</tr>
<tr>
<td>Object</td>
<td>2n</td>
<td>n bytes, each represented as two hex digits.</td>
</tr>
</tbody>
</table>

Extended Tekhex Termination Block

<table>
<thead>
<tr>
<th>Item</th>
<th>Number of ASCII Characters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>6</td>
<td>Standard header field Block type = 8.</td>
</tr>
<tr>
<td>Transfer</td>
<td>2 to 17</td>
<td>The address where program execution is to begin: a variable-length number.</td>
</tr>
</tbody>
</table>

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 your object code has been generated by an assembler or compiler that does not deal with sections, simply define one section called, for example, MEMORY, with a starting address of 0 and a length greater than the highest address used by your program; and put all your symbols in that section.

The following table gives the format of a symbol block. Tables that follow give the formats for section definition fields and symbol definition fields, which are parts of a symbol block.

### Extended Tekhex Symbol Block Format

<table>
<thead>
<tr>
<th>Item</th>
<th>Number of ASCII Characters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>6</td>
<td>Standard header field</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Block Type = 3</td>
</tr>
<tr>
<td>Section Name</td>
<td>2 to 17</td>
<td>The name of the section that contains the symbols defined in this block: a variable-length symbol.</td>
</tr>
<tr>
<td>Section Definition</td>
<td>5 to 35</td>
<td>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.</td>
</tr>
<tr>
<td>Symbol</td>
<td>5 to 35</td>
<td>Zero or more symbol definition fields as described in the next table.</td>
</tr>
</tbody>
</table>
Extended Tekhex Format

**Extended Tekhex Symbol Block: Section Definition Field**

<table>
<thead>
<tr>
<th>Item</th>
<th>Number of ASCII Characters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>A zero signals a section definition field.</td>
</tr>
<tr>
<td>Base</td>
<td>2 to 17</td>
<td>The starting address of the Address section: a variable-length number.</td>
</tr>
<tr>
<td>Length</td>
<td>2 to 17</td>
<td>The length of the section: a variable-length number, computed as 1 + (high address base address).</td>
</tr>
</tbody>
</table>

**Extended Tekhex Symbol Block: Symbol Definition Field**

<table>
<thead>
<tr>
<th>Item</th>
<th>Number of ASCII Characters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>1</td>
<td>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</td>
</tr>
<tr>
<td>Symbol</td>
<td>2 to 17</td>
<td>A variable-length symbol.</td>
</tr>
<tr>
<td>Value</td>
<td>2 to 17</td>
<td>The value associated with the symbol: a variable-length number.</td>
</tr>
</tbody>
</table>

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 Format

Figure B-6: Extended Tekhex Data Block

- Block length: 15H = 21
- Checksum: 1CH = 28 = 1+5+6+3+1+0+0+0+2+0+2+ ...
- Object Code: 6 bytes
- Load address: 100H
- Block type: 6
- Header character

Figure B-7: Extended Tekhex Termination Block

- Block length: 8
- Checksum: 1AH = 26 = 0+8+8+2+8+0
- Transfer address: 80H
- Block type: 8
- Header character
Figure B-8: 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:

<table>
<thead>
<tr>
<th>Field</th>
<th>Printable Characters</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>2</td>
<td>s-record type -- S0, S1, etc.</td>
</tr>
<tr>
<td>record length</td>
<td>2</td>
<td>The count of the character pairs in the record, excluding the type and record length.</td>
</tr>
<tr>
<td>address</td>
<td>4, 6, or 8</td>
<td>The 2-, 3-, or 4-byte address at or which the data field is to be loaded into memory.</td>
</tr>
<tr>
<td>code/data</td>
<td>0-2n</td>
<td>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).</td>
</tr>
<tr>
<td>checksum</td>
<td>2</td>
<td>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.</td>
</tr>
</tbody>
</table>

Each record may be terminated with a CR/LF/NUL. 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.
Motorola S-Record Format

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:

```
S0060000484421B
S1130000285F245F2212226A00042429000082337CA
S113001000020000800082629001853812341001813
S113002041B9000084B42234300182342000824A952
S107003000144Bd492
S9030000FC
```

The module consist of one S0 record, four S1 records, and an S9 record.

The S0 record consist 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+**
  - Four-character 2-byte address field, zeros in this example.

- **48**
  - ASCII B, D, and R - "BDR".

- **44+**
  - The checksum.
The first S1 record is explained as follows:

<table>
<thead>
<tr>
<th>S1</th>
<th>S-record type S1, indicating that it is a code/data record to be loaded/verified at a 2-byte address.</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Hexadecimal 13 (decimal 19), indicating that 19 character pairs, representing 19 bytes of binary data, follow.</td>
</tr>
<tr>
<td>00+</td>
<td>Four-character 2-byte address field; hexadecimal address</td>
</tr>
<tr>
<td>00</td>
<td>0000, where the data which follows is to be loaded.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>OPCODE</th>
<th>INSTRUCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>285F</td>
<td>MOVE.L (A7) +,A4</td>
</tr>
<tr>
<td>245F</td>
<td>MOVE.L (A7) +,A2</td>
</tr>
<tr>
<td>2212</td>
<td>MOVE.L (A2),D1</td>
</tr>
<tr>
<td>226A0004</td>
<td>MOVE.L 4(A2),A1</td>
</tr>
<tr>
<td>24290008</td>
<td>MOVE.L FUNCTION(A1),D2</td>
</tr>
<tr>
<td>237C</td>
<td>MOVE.L #FORCEFUNC,FUNCTION(A1)</td>
</tr>
<tr>
<td>0</td>
<td>(The balance of this code is continued in the code/data fields of the remaining S1 records, and stored in memory location 0010, etc.)</td>
</tr>
<tr>
<td>2A</td>
<td>The checksum of the first S1 record.</td>
</tr>
</tbody>
</table>

The second and third S1 records each also contain S13 (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:

<table>
<thead>
<tr>
<th>S9</th>
<th>S-record type S9, indicating that it is a termination record.</th>
</tr>
</thead>
<tbody>
<tr>
<td>03</td>
<td>Hexadecimal 03, indicating that three character pairs (3 bytes) follow.</td>
</tr>
<tr>
<td>00</td>
<td>The address field, zeros.</td>
</tr>
<tr>
<td>FC</td>
<td>The checksum of the S9 record.</td>
</tr>
</tbody>
</table>

Each printable character in an S-record is encoded in hexadecimal (ASCII in this example) representation of the binary bits which are actually transmitted.
Intel Hex Format

This format consists of symbol table information, data specifications for loading memory, a module starting address record (optional) and a terminator record. The format contains no information regarding the initial contents of any registers other than CS and IP: therefore, all other registers (in particular segment registers must be loaded explicitly by the programmer).

The records in the file appear in this order:

$$
symbol records - 0 or more
$$

$$
data records and segment base address records - 0 or more, any order
starting address record (optional) terminator record
$$

Symbol Record

As many symbol records as needed may be contained in the object module. A variable number of symbols per line is generated, depending on the lengths of the symbols; records are packed as tight as may be. A module may contain no symbol records. A sample record is shown below.

APPLE 00000H LABEL1 0DOC3H MEM 0FFFH ZEEK 01947H FIFTH 00005H

Segment Base Address Record

This record defines the segment base address relative to which the load addresses in subsequent data records are specified. The address in this record is 16 bits, which are the upper bits of a 20-bit address; the lowest 4 bits are presumed to be zero. This segment base address has nothing to do with any of the Loader segment addresses, base addresses, load addresses, etc. Segment base addresses are generated internally by the Loader, are not under the user’s control, and are generally of no concern to the user. The segment base address is presumed to be zero before any segment base address records are encountered.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
: 0 2 0 0 0 0 2 address checksum

Column 1 contains “:”, indicating the start of a record.
Column 2 and 3 contain "02", indicating there are 2 bytes of data in this record (the address).
Columns 4, 5, 6 and 7 contain "0000".
Columns 8 and 9 contain "02", identifying this record as a segment base address record.
Columns 10, 11, 12 and 13 contain the segment base address. Column 10 is the most significant digit and column 13 is the least significant.
Columns 14 and 15 contain a checksum, calculated as described below under Data Record.
## Data Record

This record specifies data bytes that are to be loaded into memory.

```
1 2 3 4 5 6 7 8 9 10 11 ... 41 42 43
: byte load 0 0 data data...data checksum
```

- Column 1 contains `:`, indicating the start of a record.
- Column 2 and 3 contain the count of the number of data bytes contained in this record. Column 2 is more significant.
- Columns 4, 5, 6 and 7 contain the address at which the first data byte is to be loaded. This address is a 16-bit offset from the current segment base address (see segment base address record). Column 4 is most significant, and column 7 is least significant.
- Columns 8 and 9 contain "00", identifying this record as a data record.
- Columns 10 through 41 (or fewer if not 16 data bytes) contain up to 16 bytes of data. Each byte occupies two columns, the leftmost being the more significant digit. The leftmost byte is loaded into the address specified by columns 4 through 7 (plus the segment base address); subsequent bytes are loaded into subsequent (higher) addresses.
- The last two columns contain a checksum. This is the two’s complement of the sum (modulo 256) of all bytes in the record (except the colon and the checksum itself).

## Starting Address Record

This record specifies the starting execution address of the object module. It contains startup values for the CS and IP registers.

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
: 0 4 0 0 0 0 0 3 CS IP checksum
```

- Column 1 contains `:`, indicating the start of a record.
- Column 2 and 3 contain "04", indicating there are 2 bytes of data in this record (the CS and IP values).
- Columns 4, 5, 6 and 7 contain "0000".
- Columns 8 and 9 contain "03", identifying this record as a starting address record.
- Columns 10, 11, 12 and 13 contain the 16 bit value to be loaded into CS.
- Columns 14, 15, 16 and 17 contain the 16 bit value to be loaded into IP.
- Columns 18 and 19 contain a checksum, calculated as described above under Data Record.
There are jumpers in both the 8018x pod and 80C18x pod which can be changed to specify choices in clock and chip select circuitry.

### 8018x Pod Jumpers

The five jumpers in the 8018x pod control whether chip selects are allowed to the target in pause mode and whether the target clock should bypass the conditioning circuitry in the pod.

#### Accessing the Jumpers

To access the jumpers, remove the screws which hold the pod cover on, and then remove the pod cover. The jumper and pin numbers are written on the board. Push the appropriate jumper to the setting you want.

#### Setting the Jumpers

There are five jumpers on the 80186 pod.

- **JP1**
  - Determines the state of the DT/R~ signal being asserted to the target during pause mode.
  - **JP1 1-2** DT/R~ asserted low to target during pause mode.
  - **JP1 2-3** DT/R~ asserted high to target during pause mode.

- **JP2**
  - Determines whether or not chip selects (UCS, LCS, MCS0-3, PCS0-1) are allowed out to the target in pause mode.
  - **JP2 1-2** Allows chip selects to go out to target in pause mode.
  - **JP2 2-3** Allows chip selects to go out to target ONLY in run mode or Peek/Poke cycles.

- **JP3**
  - Determines whether or not chip selects (PCS2-6) are allowed out to the target in pause mode.
  - **JP3 1-2** Allows chip selects to go out to the target in pause mode.
  - **JP3 2-3** Allows chip selects to go out to the target ONLY in run mode or peek/poke cycles.
Determines whether or not the target clock bypasses the clock conditioning circuitry in the pod. The conditioning circuitry may be bypassed if the target clock is generated by an IC in order to decrease the "clock-in to clock-out" delay. If the the clock is generated by a crystal the conditioning circuitry should be used.

<table>
<thead>
<tr>
<th>JP4 and JP5</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JP4 1-2 with JP5 1-2</td>
<td>All of target clock conditioning circuitry is used.</td>
</tr>
<tr>
<td>JP4 1-2 with JP5 2-3</td>
<td>Target clock bypasses part of the clock circuitry but still uses U57 (74HC04).</td>
</tr>
<tr>
<td>JP4 2-3 with JP5 1-2</td>
<td>All of clock conditioning circuitry is bypassed. Target clock goes through K1 relay and directly to the pod CPU.</td>
</tr>
<tr>
<td>JP4 2-3 with JP5 2-3</td>
<td>Invalid</td>
</tr>
</tbody>
</table>
80C18x and 80C18xEB Pod Jumpers

Accessing the Jumpers

80C18x
Remove the snap-on pod cover.

80C18xEB
Remove the screws on the bottom of the pod which hold the pod cover on. Remove the pod cover.

CAUTION
All ES 1800 emulator boards contain static-sensitive components. These procedures should be carried out by someone who is familiar with accepted static control practices. The ES 1800 pod cover should be removed and handled only at a static-free work station.

Failure to follow these precautions may result in permanent damage to your equipment. Check to ensure that the ES 1800 emulator is turned off before beginning this procedure.

Setting the Jumpers

The jumper numbers are written on the board. Place the shunt on the appropriate jumper setting.

80C18x, 80C18xEB Pod Jumper JP1

With the 80C18x and 80C18xEB processors, the T4 (status inactive) portion of the CPU bus cycle may be extended longer than the normal one clock cycle via the insertion of "idle" states (Ti cycles) in the CPU. The state insertion is internal to the CPU and is not user-controllable.

The leading (rising) edge of the ALE signal is specified by Intel as the rising edge of the CLKOUT signal immediately preceding T1. Since T4 can be extended internally, the ES 1800 emulator cannot determine in advance whether the next clock cycle will be a T1 or a Ti cycle. Therefore, you can shunt pins 2-3 on jumper JP1 to generate the ALE signal at the first T4 ("early" ALE), or shunt pins 1-2 on jumper JP1 to generate a "late" ALE signal after the status line goes active (indicating that the subsequent clock cycle will be a T1 cycle).

Figure C-1 shows the pin positions for the JP1 jumper. Your 80C18x or 80C18xEB pod is shipped from the factory with pins 1 and 2 of the JP1 jumper shunted together, as shown in the first drawing.

The latter method (late ALE) results in the leading edge of the ALE signal being somewhat later than specified by Intel; however, the trailing (falling) edge of the ALE signal is unaffected by the jumper position, and is as specified by Intel. With the JP1 jumper shunted for early ALE generation, the ALE signal may be longer than usual if the CPU inserts Ti cycles before the next T1 cycle.
NOTE

If you are using the emulator for the 80C18xEB microprocessor, its adaptor board also has a jumper JP1. Be sure you are setting the correct jumper for your purpose.

Figure C-1: Jumper 1 Pin Positions

80C18x, 80C18xEB Pod Jumpers JP3 and JP4

The 80C18x and 80C18xEB probes are shipped configured for 3rd harmonic crystal clock generation using the circuit layout described in the Intel manuals for the 80C186/C188 and the 80C186EB/C188EB. Jumpers JP3 and JP4 may be reconfigured to allow slower clocks (XTAL fundamental) or target system generated clock input.

JP3 Use to select external crystal or target system generated clock.

JP4 Significant only when external crystal is selected by JP4 - selects between fundamental and 3rd-overtone crystal configurations.

JP3 2-3 Target system generated clock

JP3 1-2 with JP3 2-3 3rd overtone crystal (24 MHz and above) (default)

JP4 1-2 Fundamental crystal (below 24 MHz)

Figure C-2 shows the pin positions for these jumpers.
**80C18x, 80C18xEB Pod Jumper JP5**

In some operating systems (such as Intel's RMX86 operating system), the processor is frequently halted during normal operation between interrupts. The emulator recognizes these halts and reports an error message each time. To avoid numerous "No Bus Cycle" error messages, you can set jumper JP5 to positions 1-2. This will cause the emulator to ignore the "No Bus Cycle" error messages.

With pins 2 and 3 of the jumper shunted together, the emulator properly reports any "No Bus Cycle" errors. This is the default factory setting.

**Figure C-3: Jumper 5 Pin Positions**
80C18x, 80C18xE8 Pod Jumpers JP6 through JP9

These jumpers should be left in the factory default position.
80C186EB/C188EB Adapter Board Jumper

In addition to the pod jumpers, jumper JP1 on the adapter board allows you to switch from an 80C186EB to an 80C188EB. The default position is set for the 80C186EB. The adapter board is the smaller board containing the 80C186EB chip, connected to the main pod board.

Accessing the Jumpers

To access the jumpers, remove the screws which hold the pod cover on. They are located on the bottom of the pod. Remove the pod cover. The jumper and pin numbers are written on the board.

CAUTION

All ES 1800 emulator boards contain static-sensitive components. These procedures should be carried out by someone who is familiar with accepted static control practices. The ES 1800 pod cover should be removed and handled only at a static-free work station.

Failure to follow these precautions may result in permanent damage to your equipment. Check to ensure that the ES 1800 emulator is turned off before beginning this procedure.

Setting the Jumpers

There is only one jumper on the EB adapter board. The figures below show the location of the jumper on the board and pin settings. The default position is between pins 1 and 2, for the 80C186EB. To emulate the 80C188EB microprocessor, move the shunt to connect pins 2 and 3.

CAUTION

Changing the jumper setting on the adapter board may loosen some connections if force is applied. When moving the shunt, make sure the adapter board remains well-seated.
Figure C-4: EB Adapter Board.

PIN 700-14500-00 Rev 1

Figure C-5: Jumper JP1 Settings

C186EB

C188EB
Applied Microsystems corporation offers a variety of applications notes on ES 1800 emulators which explain in more detail how to use the emulator for specific purposes.

If you would like copies of any of the Application Notes listed in this index, please contact your local sales office or representative, or the Applications department of Applied Microsystems Corporation.

If you have ideas for additional application notes you would like to see, please let us know:

Applications Department

<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
<th>Equipment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES-001</td>
<td>Downloading and Uploading to and from the Host Computer</td>
<td>ES 1800</td>
</tr>
<tr>
<td>ES-002</td>
<td>Two New Commands: COM, DIA</td>
<td>ES 1800, ESL Version 2.3</td>
</tr>
<tr>
<td>ES-003</td>
<td>Bus Error Display of ADDRESS and STATUS</td>
<td>ES 1800/68000/10</td>
</tr>
<tr>
<td>ES-004</td>
<td>How to Simplify Design Integration of uP Based System Using the Event Monitor System</td>
<td>ES 1800</td>
</tr>
<tr>
<td>ES-005</td>
<td>Production Test Uses for Emulation</td>
<td>EM and ES Series</td>
</tr>
<tr>
<td>ES-006</td>
<td>How to Use the Applied Microsystems ES 1800 Emulator to Determine the Duration of a Subroutine</td>
<td>ES 1800</td>
</tr>
<tr>
<td>ES-007</td>
<td>Selectively Tracing Using the Breakpoint System</td>
<td>ES 1800</td>
</tr>
<tr>
<td>ES-008</td>
<td>ES 1800/68000/08/10 ITR and PPT</td>
<td>ES 1800/68000/08/10</td>
</tr>
<tr>
<td>ES-009</td>
<td>How to Break on Execution as Opposed to Prefetch</td>
<td>ES 1800/68000/08/10/20</td>
</tr>
<tr>
<td>ES-010</td>
<td>Use of the ES 1800 “COM” Command</td>
<td>ES 1800/68000/08/10/20</td>
</tr>
<tr>
<td>ES-011</td>
<td>Using the COM Command to Simulate a Terminal I/O Device</td>
<td>ES 1800/68000/08/10</td>
</tr>
<tr>
<td>ES-012</td>
<td>Helpful Things to Know about the ES 1800</td>
<td>ES 1800</td>
</tr>
<tr>
<td>ES-013</td>
<td>Operating the ES 1800 68020 at 16.67 MHz</td>
<td>ES 1800/68020</td>
</tr>
<tr>
<td>ES-014</td>
<td>GenePak 8087 Emulation Software</td>
<td>GenePak</td>
</tr>
<tr>
<td>ES-015</td>
<td>How to Assemble Code and Descriptor Tables in 80286 Protected Mode</td>
<td>ES 1800/80286</td>
</tr>
<tr>
<td>ES-016</td>
<td>Running the 68020 Emulator with a Motorola VME-133 Board</td>
<td>ES 1800/68020</td>
</tr>
<tr>
<td>Number</td>
<td>Title</td>
<td>Equipment</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ES-018</td>
<td>Pinpointing an Overlay Memory Chip Failure on Boards with 512K Max Overlay</td>
<td>ES 1800 Overlay Boards: 700-11272, 700-11278, 700-11275, 900-11277</td>
</tr>
<tr>
<td>ES-019</td>
<td>68020A Timing Specifications</td>
<td>ES 1800/ 68020A</td>
</tr>
<tr>
<td>ES-020</td>
<td>ES 1800 Training Manual</td>
<td>ES 1800</td>
</tr>
<tr>
<td>ES-021</td>
<td>Pinpointing an Overlay Memory Chip Failure on High Speed Overlay Boards</td>
<td>ES 1800 Overlay Boards: 700-1160X-XX</td>
</tr>
<tr>
<td>ES-022</td>
<td>68010 Timing Specifications</td>
<td>ES 1800/ 68010</td>
</tr>
<tr>
<td>ES-023</td>
<td>The 80286: Protect Mode Tools</td>
<td>ES 1800/ 80286, VALIDATE/Soft-Scope 286</td>
</tr>
<tr>
<td>ES-024</td>
<td>80286 Timing Specifications</td>
<td>ES 1800/ 80286</td>
</tr>
<tr>
<td>ES-025</td>
<td>Cross Triggering Multiple Emulators</td>
<td>ES 1800, EL 800</td>
</tr>
<tr>
<td>ES-026</td>
<td>Reducing Memory Usage in MCC68K/DOS</td>
<td>MCC68K/DOS</td>
</tr>
<tr>
<td>ES-027</td>
<td>GPVS Software Utility</td>
<td>ES 1800, with GeneProbe and VALIDATE/Soft-Scope</td>
</tr>
<tr>
<td>ES-028</td>
<td>80186 Timing Specifications</td>
<td>ES 1800/ 80186</td>
</tr>
<tr>
<td>ES-029</td>
<td>Incremental Linking with LOD68K/DOS</td>
<td>LOD68K/DOS: rev 6.3b and previous</td>
</tr>
<tr>
<td>ES-030</td>
<td>Commonly Asked Questions on VALIDATE/XE and XRAY</td>
<td>VALIDATE/XE and XRAY</td>
</tr>
<tr>
<td>ES-031</td>
<td>Understanding the Z8002 NMI Cycle and the Emulator</td>
<td>ES 1800/ Z8000</td>
</tr>
<tr>
<td>ES-032</td>
<td>Using the FSI (Force Special Interrupt) Action</td>
<td>ES 1800</td>
</tr>
<tr>
<td>ES-033</td>
<td>Using the UNIX tip program to control an ES 1800</td>
<td>ES 1800</td>
</tr>
<tr>
<td>ES-034</td>
<td>68020-25 MHz AC Probe tip Timing Specifications</td>
<td>ES 1800/68020-25</td>
</tr>
<tr>
<td>ES-035</td>
<td>Connecting Multiple SCSI Emulators to a Sun Workstation</td>
<td>ES 1800</td>
</tr>
<tr>
<td>ES-036</td>
<td>Connecting Multiple SCSI Emulators to a PC</td>
<td>ES 1800</td>
</tr>
<tr>
<td>ES-037</td>
<td>Speeding SCSI Downloads</td>
<td>ES 1800</td>
</tr>
<tr>
<td>ES-038</td>
<td>ES 1800/80C186/C188 Timing Specifications</td>
<td>ES 1800/80C18x</td>
</tr>
<tr>
<td>ES-039</td>
<td>Using the Overlay Speed Setting</td>
<td>ES 1800/68000 family</td>
</tr>
<tr>
<td>ES-040</td>
<td>ES 1800/68000-16MHz AC Probe Tip Timing Specifications</td>
<td>ES 1800/68000-16MHz</td>
</tr>
<tr>
<td>ES-041</td>
<td>Setting Up ES 1800/683032 Chip Selects</td>
<td>ES 1800/68302</td>
</tr>
<tr>
<td>Number</td>
<td>Title</td>
<td>Equipment</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>ES-042</td>
<td>Programming the 8018x/80C18x Peripheral Control Block</td>
<td>ES 1800/8018x/C18x</td>
</tr>
<tr>
<td>ES-043</td>
<td>Comparison of Source-Level Debug Tools for Intel 16-Bit Microprocessors</td>
<td>ES 1800/intel</td>
</tr>
<tr>
<td>ES-044</td>
<td>Statistical Performance Analysis</td>
<td>ES 1800</td>
</tr>
<tr>
<td>ES-046</td>
<td>ES 1800/68302 Emulator Hardware Specifications</td>
<td>ES 1800/68302</td>
</tr>
</tbody>
</table>
This appendix shows the RS-232C serial cable connections between the ES 1800 and various host computers. Serial communication between two Data Terminal Equipment (DTE) devices requires the use of a null modem cable. The following figures show the wiring diagrams for cables supplied with ES 1800 emulators.

**PC 25-Pin Serial Cable**

*Figure E-1. ES 1800 to PC/XT*

* Note that pins 6, 8, and 20 are not used and are unaffected by the cable configuration.
PC 9-Pin Serial Cable

Figure E-2. ES 1800 to PC/AT
Sun 25-pin Serial Cable

Figure E-3. ES 1800 to Sun 3/50 or 3/60

Figure E-4. ES 1800 to Sun 3/80 (ttya) *

* Also works for Sun 3/50 and Sun 3/60
Figure E-5. ES 1800 to Sun 3/80 (ttyb)
# 8-9
$ 8-9
% 8-9
* 4-56, 4-61, 7-7
/ 4-56, 4-61, 7-6
: 4-58
<Backspace> 8-24
<ctrl-q> 8-24
<ctrl-r> 8-24
<ctrl-s> 8-24
<ctrl-x> 8-24
<ctrl-z> 4-56, 4-61, 7-72
<Delete> 8-24
<esc><esc> 8-24
? 2-5
@ 4-52, 4-54, 7-2
\ 8-9
- 4-56, 7-8, 8-5
' 4-56, 7-4
80C18X
  unique registers 4-25
80C18X specific features
  ignore halt errors 7-72
  interrupts during pause 7-70
  refresh during pause 7-100
  targets with attached CPUs 2-4
80C18xEB specific features
  unique registers 4-17

-A-
Absolute address 7-68
Absolute time 6-2
Absolute value 8-13
Acknowledge char 7-116
Actions 1-10
  break 7-16
  CNT 7-31
  definition 4-33

RCT 7-106
TGR 7-152
TOC 7-153
TOT 7-154
TRC 7-157

Address
  absolute 7-68
  branch to 7-64
  comparators 4-36
  odd 4-36
  registers 7-68

ALE signal jumper C-3, C-4
Alphanumeric value 8-6
ALT 7-92
AND 4-35, 7-169, 8-12
Application notes D-1
ARDY 7-18
ASM 4-52, 7-9
Assemble line 4-52, 4-53
Assembler 7-9
directives 7-9

-B-
BAS 4-7, 4-47, 4-57, 7-12
Base
  default 4-7, 7-12, 7-45
  display 4-12, 4-49
  registers 8-8
Base definition symbols 8-9
Baud rate 3-14, 4-4, 7-116
  emulator ports 3-3, 4-5
BKX 4-29, 7-13
Blank lines 7-108
Block data
  verify 7-162
Block move 7-14
  verify 7-163
BMO 4-52, 7-14
Index, continued

BNC connector 3-5, 7-152

Boards
  control 3-1
  emulation 3-3
  Future Domain 3-14
  MCB controller 2-2, 3-2
  RAM overlay 2-2
  SCSI 2-2, 3-2, 3-14
  trace and break 2-2, 3-3

Break
  on execution 4-29, 7-13
  on NMI 4-40
  on odd address 4-36

Break emulation 1-10, 4-29, 4-30, 6-24, 7-16

Breakpoints 2-9, 4-29
  disabling 4-29
  enabling 4-29
  run until 7-103
  setup 4-32-4-46
  setup example 4-41-4-45
  while running COM 7-37

Bringing up hardware 1-13

BRK 4-29, 4-32, 4-41, 7-16

BTE 7-19

BTO register 7-20

BUS 4-50, 7-18

Bus cycles 7-55

Bus status 5-3, 7-18

Bus timeout enable 7-19

Bus timeout register 7-20

BYM 4-52, 7-21, 7-167

Byte mode 7-21, 7-167, 8-22

-C-

Cables 2-3, 3-7, 3-17

CCT 4-3, 4-5, 7-23, 7-156

CD 7-92

CDH 7-24

CES 4-32, 4-57, 4-64, 7-25

Chassis 3-1

Checksums 4-9

Chip select 7-98, 7-105

Chip select circuitry 4-13

Chip select registers 4-15, 4-16

CK 7-26

Clear
  commands 4-57, 4-64
  CPU registers 4-7, 7-29
  DMA halt 7-24
  macros 7-30
  memory map 4-57, 7-28
  WHEN/THEN 4-32, 7-25

CLK 7-27

CLM 4-6, 4-57, 7-28

Clock
  internal 1-9
  internal/external 7-26
  read target 7-27
  target clock frequency 5-3

CLR 4-7, 4-47, 4-57, 4-64, 7-29

CMC 4-56, 7-30

CNT 4-40, 4-41

Code space 7-92

Colon operator 4-58

COM 4-50, 7-34

Commands
  arguments 7-68
  clear 4-57
  command line 8-5
  commonly used 8-17, 8-18
  configure system 4-5, 4-27
  delay execution 7-166
  ESL 8-5
  exceptions 8-5
  extending 8-5
  language overview 1-5
  memory 4-52
  mnemonics 8-6
  port dependent 4-5
  repeating 4-61, 7-6, 7-159, 8-5, 8-23
  run mode 4-29, 4-53
  single character 8-5
  symbols 4-56
  terminator sequence 7-116
# Index, continued

## Communication
- emulator 4-3
- establishing 1-8
- parameter setup 4-6
- SCSI 1-8, 4-5
- serial 1-8, 4-3
- target programs 7-34
- with host 4-4

## Communications
- help 8-16

## Computer control
- 1-3

## Computer port control
- 4-3, 4-9, 7-23

## Configuration
- menus 4-5, 4-27
- pins 3-14
- system 1-2

## Control boards
- 3-1

## Control characters
- 8-24

## Copy system variables
- 7-74

## Count events
- 7-31
  - reset 7-106
  - toggle counter 7-153

## Count limit comparator
- 4-40

## Count occurrences
- 6-20

## Counter overflow
- 6-6, 6-7

## Counter/timer use
- 3-5, 7-152

## CPU registers
- 4-49
  - clear 4-7
- CPY 4-50, 7-38
- CRC 7-39
- CRE 7-39
- CRO 7-39
- CRT 7-61
- CRT length 7-116
- CS:IP 4-11, 4-49
- CSEG 7-10
- CTL 4-40, 7-31
- CTS 7-40

## Customer service
- ii, 2-9

## Cyclic redundancy check
- 5-3, 7-39, 7-135

## -D-

## Data
- buffering 4-4
- comparator registers 4-37
- downloading 7-51
- enable 7-50
- printing 4-50, 7-38
- requirements 3-15
- serial formats B-1
- upload 7-160
- verify 7-52

## Data length
- 4-52, 7-21, 7-167
  - byte 4-52
  - word 4-52

## Data space
- 7-92

## DB
- 4-7, 4-52, 7-10, 7-41

## DB-25 connectors
- 4-3

## Debuggers
- high level 1-16

## Debugging mechanical systems
- 4-42

## Decoding
- memory and I/O 4-13

## Default base
- 7-45

## Default base register
- 4-7, 8-8

## Definitions
- 4-2

## DEL
- 4-56, 7-43

## Delete
- symbol or section 7-43, 7-102

## DES
- 4-32, 7-44

## DFB
- 7-45

## DIA
- 4-48, 7-46

## Diagnostic functions
- 1-13, 5-1, 7-120

## Diagnostics
- complete RAM test, looping 7-125
- complete RAM test, single pass 7-123
- cyclic redundancy check 7-135
- read data over entire range 7-134, 7-144
- read from address 7-127, 7-137
- simple RAM test, looping 7-124
- simple RAM test, single pass 7-121
- toggle data at address 7-126, 7-136
- write alternate patterns 7-129, 7-139
- write data then read 7-132, 7-142
write data to address 7-128, 7-138
write incrementing value 7-133, 7-143
write pattern then rotate 7-130, 7-140

DIS 4-7, 4-52, 7-48

Disassemble 7-48
memory 4-7
single step 4-61, 7-7, 7-159
trace memory 7-59
trace page 7-61

Display
base 4-12, 7-45
bus status 5-3, 7-18
character string 7-46
event specifications 7-44
improve readability 7-108
macros 7-79
memory block 7-41
memory map 7-49
PCB registers 2-12, 2-17
raw trace bus cycles 7-55
registers 4-7
revision dates 7-109
sections 7-115
symbols 7-147
trace 2-9, 4-49

DM 4-6, 7-49, A-5
DMA channels 4-13
DMA controllers 2-10, 4-15, 7-50
DMA halt 7-24
DME 4-15, 7-50
DNL 4-5, 4-6, 4-9, 4-10, 7-51
DNV 7-52
Don’t care values 4-12, 4-37, 4-39, 7-67, 8-9

Download 3-14, 4-6, 4-9
corrupt 7-165
ersors 4-10
from computer port 4-9
from terminal port 4-9
hex format files 3-14
port control 4-10
procedures 7-51
record format 7-119

returning control to emulator 4-10
symbols 4-11
verify data 4-6

Download speed 1-8
DR 4-7, 4-47, 7-53
DRAM 7-100
DRT 4-47, 4-49, 7-55, 7-148, A-5
DT 4-47, 4-49, 7-59, 7-148
DTA 7-92
DTB 4-47, 4-49, 7-61, 7-148, 7-166
DTF 4-47, 4-49, 7-61, 7-148
Dumb terminal setup 1-2
DW 7-10
Dyadic operator 8-15
Dynamic memory 7-50
Dynamic RAM refresh 4-13
Dynamic trace 1-12, 4-49
capture enable 7-148
with event system 4-33

EEPROM

groups 4-63
initialize 2-6, 7-114
load from 4-7, 4-57
save configuration 4-63, 7-87, 7-114
save to 4-7

Elapsed time 6-12
A to B 6-12
in range 6-12, 6-15
inter-module 6-12
measurement 6-1
measurement examples 6-12
out-of-module 6-12

Emulation 1-5, 1-6, 4-1
break 1-10, 4-29, 4-30, 6-24, 7-16
starting 4-29

Emulation board 3-3

Emulator
control boards 3-1
front panel 3-1, 3-5
hardware error messages A-4
rear panel 3-5
specifications 3-21
Emulator setup 2-2-2-3
Enable chip selects 2-10, 7-98
Enable data 7-50
END 4-52, 7-10
Enter program 2-7
EQU 7-10
Equation 8-6
emulator hardware A-4
ESL 8-26
received on computer port 4-10
target hardware A-1
target software A-5
while running COM 7-37
Errors
ESL 2-5, 8-25
no memory in header 2-4
syntax 8-25
ES Driver 1-9, 1-16, 4-45
communication 4-5
control software 1-3
ES Language 1-5, 8-1-8-30
commands 8-5
error messages 8-26
syntax 8-1
Escape sequence 7-116
ESL 1-5, 2-5, 8-1-8-30
control 1-2
errors 2-5
prompts 8-21
revisions 6-4
Event comparators 4-36
Event monitor system 1-10, 4-32-4-46,
8-11, 8-18
address comparators 4-36
clear WHEN/THEN 4-32, 7-25
comparator registers 4-34, 4-36
count events 7-31
data comparators 4-37
define action list 4-40
examples 4-41-4-45
groups 4-40, 7-69, 8-11
interrupts 7-64
LSA comparators 4-37
registers 4-25
reset counter 7-106
setup 4-32, 6-9
speed 6-25
status comparators 4-38
status mnemonics 4-38
structure 4-33
toggle counter 7-153
trace events 7-154, 7-157
trigger signal 7-152
WHEN/THEN 4-40, 7-169, 8-11
with dynamic trace 4-33
with software debuggers 4-45
Event specifications
display 7-44
use of parentheses 4-34
Events 1-10
definition 4-33
Exit line assembler 4-52
Expression 8-6
Extended Tek Hex format B-7

-F-
Fan 3-1
Fan filter
cleaning 3-17
FIL 4-52, 7-62
Files
closing 7-160, 7-161
opening 7-161
viewing 7-160
Fill operator 7-62
FIN 4-52, 7-63
Find memory pattern 7-63
Firmware check 7-135
FLX 4-11
FLX register 7-54
Forced special interrupt 7-66
Formats 7-119
Extended Tek Hex B-7
Intel Hex B-18
Index, continued

Intel Intellec  B-4
    MOS  B-2
Motorola Exorcisor  B-3
Motorola S-record  B-14
    serial data  B-1
Signetics  B-5
Tek Hex  B-6
FSI  4-41, 7-64
FSX  7-66
Fuses  2-2, 3-6

-G-
    GD  4-57, 7-67
GeneProbe  1-16
    with event monitor system  4-46
General purpose registers  1-13, 7-67, 7-68
Global data length  7-21, 7-167
GR  4-57, 7-68
GRO  4-41
Ground  2-3, 3-1

-H-
    Handshake
        hardware  3-16
        software  3-16
    Hanging pod  3-9
Hard copy  7-38, 7-90
Heat problems  2-4, 3-1
Help  2-5, 8-16
Help menu  8-17
High level debuggers  1-16
HLT  A-2
Host computer  1-3, 4-3
    communication with  4-4
    configuration  4-5
Host control  1-3

-I-
    I/O address space  4-55
I/O mode  4-55, 8-22
        enter  4-7
        entering  7-83
exiting  7-170
    pointer  7-73
I/O simulation  4-50
IDP  4-17, 7-70
IDX  4-56
Ignore halt errors  7-72
IHE  7-72
ILG  7-81
Illegal statement  2-5, 8-1
Indirection  8-7
Initialize
    EEPROM  2-6
    PCB registers  2-9, 2-14
    system  2-6
Installation
    LSA  3-12
    time stamp module  6-3
INT0  7-18
INT1  7-18
INT2/INTA0  7-18
INT3/INTA1  7-18
Integer  8-6
Intel Hex format  B-18
Intel Intellec format  B-4
Internal clock  1-9
Interrupt controller  4-13
Interrupt controller registers  2-11, 4-16
Interrupt latency  6-6, 6-17
Interrupts
    force special interrupts  7-64
    special interrupt register  7-64
    step through  4-29, 7-145
Interrupts during pause  7-70
IOP  4-52, 7-73
Isolate problem  1-11

-J-
Jumpers
    8018X  C-1
    80C18X, 80C18XEB  C-3
    ALE signal jumper  C-3, C-4
    J1  2-2, 3-2
    pod  2-1
Index, continued

- timer enable C-5
  -L-

  LCC socket  3-7
  LCS- signals  2-4
  LD  4-7, 4-47, 4-57, 7-74
  LDV  4-7, 4-12, 4-29, 4-47, 7-75
  LIM  4-56
  Limit register  4-56
  Line assembler  4-52, 4-53, 7-9, 8-22
  exiting  7-170
  Line assembler prompt  8-21
  LMCS register  4-15, 4-16
  Load
    overlay memory  7-76
    registers  4-7, 4-49
    reset vectors  4-7, 4-29, 7-75
    setup from EEPROM  4-57
    variables  7-74
  Log in banner  8-19
  Logic State Analyzer (LSA)  1-11, 3-5,
    3-12, 6-3
  LOV  4-6, 4-47, 4-48, 4-52, 7-76
  Low byte  4-37
  LSA port  3-5
    parameter setting  6-7
  LSA timing  3-12
  LST  7-86

- M-

  M  4-52, 7-77, 7-111
  MAC  4-56, 7-79
  Macros  1-13, 4-62
    clearing  7-30
    commands  4-56
    defining and using  7-8
    displaying  7-79
    saving  4-62
    truncation  4-63
    using registers  4-63, 7-67, 7-68
  Mainframe  3-1
  Maintenance  3-17
  MAP  4-6, 4-47, 4-48, 7-80

  Map overlay  7-80
  MCB controller board  2-2, 3-2
    switch setting  3-3
  Mechanical systems
    debugging  4-42
  Memory
    alternate overlay/target  7-90
    assembler  7-9
    block display  7-41
    block move  7-14
    clear overlay map  7-28
    commands  4-52
    disassemble  4-7
    disassembler  7-48
    display map  7-49
    download to overlay  7-51
    enable overlay  7-92
    fill  7-62
    find data pattern  7-63
    illegal  7-81, A-5
    load target to overlay  7-76
    map overlay  4-8, 7-80
    modify  4-51, 4-54
    overlay  4-7, A-2
    overlay speed  4-8, 7-93
    overview  1-11
    read only  7-80, A-5
    read/write  7-2, 7-80
    scrolling  4-54
    scroll through  4-54
    trace  1-11, 4-48
    verify overlay  7-164

  Memory block
    display  4-7
  Memory disassembler  8-22
  Memory map
    clear  4-6, 4-57
    display  4-6
    set  4-6
  Memory mode  4-54, 8-22
    entering  7-77
    exit  4-52
    exiting  7-170
Index, continued

modifying data 7-78
pointer 7-85
scrolling 7-86
Memory mode prompt 8-21
Microprocessor registers 4-19, 7-53
MIO 4-7, 4-14, 4-52, 7-83, 7-111
MMCS register 4-15
MMP 4-52, 4-54, 7-85
MMS 7-92
Modes
memory 4-54
ONCE 2-4
pause 4-2
run 4-2
special ESL 8-22
transparent 4-2, 7-156
Modifying program 1-12, 4-51
MOS format B-2
Motorola Exorcisor format B-3
Motorola S-record format B-14
MPCS register 4-15
Multiple users 4-63
Multiplex lines 4-33

-N-
NMI 4-16, 7-18
break 4-40
NOT 4-34, 7-169, 8-12
Null modem cable 2-3
Null target 1-9, A-3
Numbers
ESL 8-8
signed 8-12
NXT 7-86

-O-
Object module format 1-15
Odd address
break on 4-36
jump to 4-37
OFF 7-87
OFF -1 4-57
ON 7-87
ON/OFF 4-7, 4-14, 4-47, 7-87
save menu 4-27
ONCE mode 2-4
Operator precedence 8-7, 8-13
Operators
dyadic 8-15
ESL 8-7
indirection 8-10
precedence 4-34
single argument 8-14
OR 4-35, 7-169, 8-12
ORG 7-10
Oscilloscope use 1-13, 4-42, 5-2, 7-152, A-3
OVE 4-6, 7-81, 7-92
Overflow counter 6-7
Overlay
display map 7-49
run program from 4-48
Overlay memory 1-8, 4-7, A-2
boards 3-3
enable 4-6, 7-92
load 4-6
map 2-7, 4-6, 4-7, 4-8, 7-80, 7-81
size options 4-7
speed 4-6, 4-8, 7-93
verify 4-6
wait states 4-8, 7-93
OVL 7-57
OVS 4-6, 4-8, 7-93

-P-
PACS register 4-15
Parentheses 7-169
ESL 8-6
indirection 8-10
WHEN/THEN 8-12
Parentheses in event specifications 4-34
Parity 7-116
Parts 3-19
Patch program 4-8, 4-53, 7-3, 7-64
Pause mode 1-5
definition 4-2
interrupts 7-70
PCB 4-14
peripherals 4-14
refresh 7-100
PCB 4-7, 4-47, 7-95
common problems 4-14
during pause 4-14
relocation 4-13
PCB registers 4-11, 4-13-4-26, 7-95
8018x, 80C18x 4-19
80C18X 4-25
80C18xEB 4-21
default location 4-13
display 2-12, 2-17, 4-7
enhanced mode 4-25
initialize 2-9, 2-14
master mode 4-24
pause-to-run 4-13
run-to-pause 4-13
slave mode 4-24
PCS 7-98
Peeks 7-99, 7-127, 7-137
definition 4-2
Performance analysis 3-11
Peripheral control registers
initialize 2-9, 2-14
Peripheral control registers during pause 4-14
PIA 7-70
Pin configurations 3-14
PLCC adapter 3-7
Pod 3-7
Pod connection 3-5
Pod jumpers 2-1
Pokes 7-99, 7-128, 7-138
definition 4-2
POL register 4-16
Port parameters 4-5
Ports 3-5, 3-14
baud rate 3-14
commands 4-5
configuration 4-3, 7-116
control 4-5, 4-10
copying data to 7-38
download data 7-51
LSA 3-5
parameter setup 4-3
port control 3-14, 7-23, 7-149, 7-156
SCSI 3-5, 3-14
serial 3-14
terminal control 7-149
upload/download 3-14
POS register 4-16
Power 3-21
Power controller 4-13
Power supply 2-3, 3-1
Power-up sequence 2-4
no target 2-5
with target 2-4
PPT 7-99
PRE 4-14, 7-10, 7-100
Prefetch 4-36, 7-13, 7-66
Print session 7-90
Probe tip 3-17
connecting to target 3-7
Problems
isolating 4-47
Program
entering 2-7
Prompts 8-21
Prototype hardware 1-13
PUR 4-56, 4-57, 7-102

-Q-

Question mark 2-5

-R-

RAM
overlay board 2-2
testing 2-7, 5-2
Range 8-9
ESL 8-9
Raw trace 7-55
RBK 4-29, 7-103, 7-166
RBV 4-29, 7-104
RCS 4-15, 7-105
RCT 4-40, 4-41, 7-31
Index, continued

RDY 7-107
Read chip select 7-105
Read serial status 7-111
Read/write memory 7-2
Ready signal 7-107
Real time 1-5
Reentrant code 4-43
Refresh controller 4-13
Refresh during pause 7-100
Registers 1-12, 8-10
  address 4-36
  BTO 7-20
  checking 4-11
  chip select 4-15, 4-16
  clear 4-7, 7-29, 7-53
  comparator 4-34
  count limit 4-40
  CPU 4-49
  display 4-7, 7-53
  display base 4-12, 4-49, 7-12
  dynamic RAM refresh 4-14
  event monitor system 4-25
  general ES1800 4-26
  general purpose 1-13, 4-57, 4-63, 7-67, 7-68
  interrupt controller 4-16
  load 4-49, 7-53
  microprocessor 4-19
  MMP 4-54
  overlay memory 4-8
  PCB 4-24, 4-25, 7-95
  reset status 4-29, 7-112
  run mode 4-12
  save 4-12, 4-49
  serial controller 4-17
  set base 7-12
  status 4-38
  target PCB 4-19, 4-21
  values 4-12
Relative time 6-2
Repairs ii
Repeat command line 7-6, 7-7, 8-23
Repeat commands 1-13
Reset A-3
  pod microprocessor 4-29, 7-112
Reset character 7-116, 7-166
Reset vectors
  load 4-7, 4-29
  run 7-104, 7-110
RET 4-48, 7-108
Return authorization number ii
REV 7-109
Revision dates 7-109
RNV 4-29, 7-110
RO 7-80
RS232 4-3
RSS 7-111
RST 4-29, 7-112
RUN 4-29, 7-113
  commands (chart) 4-30
Run
  breakpoints disabled 4-29
  from overlay 4-48
  halt emulation 4-30
  target program 4-29, 7-103, 7-104, 7-110, 7-113
Run mode 1-5, 4-12, 4-29
  commands 4-53
  definition 4-2
  prompt 8-21
Run program 1-9, 6-10
RW 7-80

-S-

SAV 4-7, 4-47, 4-57, 7-114
Save
  ON/OFF 4-27, 4-63, 7-87
  parameters 7-119
  registers 4-7, 4-12, 4-49
  setups 1-13
  switches 4-7, 4-27, 7-87
  system variables 7-114
Scope loops 5-2
Scroll
  direction 4-54
  through memory 7-41
trace buffer 4-49
SCSI board 3-14
SCSI communication 1-8, 2-2, 3-2, 4-5
SCSI port 3-5, 3-14
SEC 4-56, 7-115
Sections 4-57
  define 4-56, 7-4
  deleting 7-43, 7-102
  display 7-115
SEG 7-57
Serial communication 1-8, 4-3
Serial controller registers 4-17
Serial ports 3-5
Service 2-6, 2-9
SET 4-3, 4-5, 4-6, 4-57, 7-116
Set command 6-7
SET menu 7-116
Setup 7-116
  emulator 2-2-2-3
  port parameters 4-3
  save 1-13, 7-114
  system 1-2
  target system 2-3, 4-6
SF 0 7-121
SF 1 7-123
SF 11 7-133, 7-143
SF 12 7-134, 7-144
SF 13 7-135
SF 2 7-124
SF 24 7-126, 7-136
SF 25 7-127, 7-137
SF 26 7-128, 7-138
SF 27 7-129, 7-139
SF 28 7-130, 7-140
SF 29 7-132, 7-142
SF 3 7-125
SF 31 7-133
SF 32 7-134, 7-143, 7-144
SF 4 7-126, 7-136
SF 5 7-127, 7-137
SF 6 7-128, 7-138
SF 7 7-129, 7-139
SF 8 7-130, 7-140
SF 9 7-132, 7-142
Shortcuts 1-12, 4-56
SIA 7-64
Signetics format B-5
Signing
  ESL 8-12
Simulate I/O 4-50
Single step 2-8, 4-29, 4-50, 7-6, 7-7, 7-146, 7-159
Single-argument operators 8-14
Soft shutdown routine 4-42
Soft switches 4-7, 4-27-4-28
  80186/188 7-88
  80C186/C188 7-89
  80C186EB/C188EB 7-90
Soft-Scope 1-17, 4-46
Software debuggers
  with event monitor system 4-45
Software options 1-15
Special characters 8-24
Special functions 1-13, 5-1, 7-120, 8-22
  help 8-16
Special interrupts 7-64
Special modes 8-22
Speed
  overlay memory 7-93
SRDY 7-18
S-records
  creation B-16
  format B-14
  types B-15
STA 7-92
Stand-alone operation 1-3
Status comparators 4-38
Status mnemonic table 4-39
Status translation table 4-39
Step, single 2-8, 4-29, 4-50, 7-6, 7-7, 7-146, 7-159
STI 4-29, 7-145
Stop and step target system 4-29, 7-146
Stop bits 7-116
Stop program 2-8, 4-50
STP 4-29, 7-146
Switches 4-27–4-28, 6-8, 7-87
break on instruction 4-29, 7-13
dynamic trace capture enable 7-148
FSI on instruction execution 7-66
help 8-16
internal/external clock 7-26
internal/external ready signal 7-107
interrupts during pause 7-70
refresh during pause 7-100
step through interrupts 4-29, 7-145
SYM 4-56, 7-147
Symbolic debugging 1-12
Symbolic references 8-8
Symbols 4-57
define 4-56, 7-4
deleting 7-43, 7-102
display 7-147
downloading 4-11
maximum limit 4-57
symbolic debugger 1-13
tables 4-58
uploading 7-161
System
initializing 2-6
operation 1-5
reset 7-166
setup 1-2, 4-5, 4-27
testing 2-6
System variables 7-114

-T-
T4 state 4-33
TAR 7-57
Target
bus cycle 4-2
clock 7-26
communication with 7-34
cyclic redundancy check 7-39
definition 4-2
display memory string 7-46
download to 7-51
hardware errors A-1
load into emulator 7-76
null 1-9
PCB registers 4-19, 4-21
problems 4-47, 4-48
read clock 7-27
reset 2-4
run program 4-29, 7-103, 7-104, 7-110, 7-113
software error messages A-5
system peeks 7-127, 7-137
system pokes 7-128, 7-138
system setup 2-3
VCC 8-19
Target environment setup 4-6
TCE 4-47, 7-148
TCT 4-3, 4-5, 7-149, 7-156
TE 4-14, 7-150
Tek Hex format B-6
Temperature 3-21
Terminal
dumb 4-3
control 1-3
setup 1-2
Terminal port control 4-3, 4-9, 7-149
Terms 4-2
TEST 7-18
Test register 7-159
Test run of system 2-6
TGR 4-41, 6-9, 7-152
event monitor system 6-8
external 6-8
TGT 7-81
Thumbwheel switch 2-2, 3-2
Time base
maximum 6-2
switch 6-7
Time measurement maximums 6-7
Time stamp 6-10
convert value 7-40
counter 6-8
Time stamp information
saving in file 6-11
viewing 6-10
Time stamp module 1-14, 3-11, 6-1–6-25
software 6-4
TGR inputs 6-5
using 6-5
Timer enable jumper C-5
Timers 2-11, 2-15, 4-14, 7-150
Timing
LSA 3-12
timer 3-12
TOC 4-41, 7-31, 7-153
Toggle data at address 7-136
TOT 4-41, 7-154
TRA 4-3, 4-5, 4-6, 7-156
Trace
disassemble memory 7-59
disassemble page 7-61
display 2-9
display bus cycles 7-55
dynamic 1-12, 4-49
events 7-154, 7-157
memory 1-11, 4-48
memory, buffer size 4-48
subroutine 7-69
Trace and break board 2-2, 3-3
Tracing peeks and pokes 7-99
Transparent mode 4-3, 7-156, 8-22
definition 4-2
terminating 4-3, 4-4, 4-5, 4-6
exit 4-4
TRC 4-41, 7-157
Trigger signal 7-152
Trigger timing 3-12
Troubleshooting 3-20
TST 4-56, 4-61, 7-159

-U-

UCS- signals 2-4
Unary operator 8-12
UPL 4-5, 7-160
Upload 3-14
data 7-160
hex format files 3-14
record format 7-119
record length 7-119
symbols 7-161
UPS 4-5, 7-161
Users
specifying 7-116

-V-

VALIDATE
communication 4-5
Soft-Scope 4-46
with event monitor system 4-46
VALIDATE/Soft-Scope 1-17
Variables
loading 7-74
VBL 4-52, 7-162
VBM 4-52, 7-163
Vectors
load reset 7-75
Vent 3-1
Verify
block data 7-162
block move 7-163
code 4-6
data 7-52, 7-165
overlay memory 7-164
serial data 4-6
VFO 4-6, 7-164
VFY 4-6, 7-165
Voltage
configuring for 2-2

-W-

WAI 4-48, 4-49, 7-166
Wait states
overlay memory 1-5, 4-8, 7-93
Warranty ii
WDM 4-52, 7-21, 7-167
WHEN/THEN 1-10, 4-29, 4-32, 4-40, 7-25,
7-169
conflicting 4-41
syntax 4-34
Word mode 7-21, 7-167, 8-22
Applied Microsystems Corporation

Applied Microsystems Corporation maintains a worldwide network of direct sales offices committed to quality service and support. For information on products, pricing, or delivery, please call the nearest office listed below. If you are unsure which office to contact, call 1-800-426-3925 for assistance.

CORPORATE OFFICE
Applied Microsystems Corporation
5020 148th Avenue Northeast
P.O. Box 97002
Redmond, WA 98073-9702
(206) 882-2000
1-800-426-3925
Customer Support
1-800-ASK-4AMC
TRT TELEX 185196
Fax (206) 883-3049

EUROPE
Applied Microsystems Corporation Ltd
AMC House
South Street
Wendover
Aylesbury, Bucks
HP22 6EF England
44 (0) 296-625462
Telex 265871 REF WOT 004
Fax 44 (0) 296-623460

GERMANY
Applied Microsystems GmbH
Dammstrasse 6
W-6453 Seligenstadt
Germany
06182/9203-0
Fax 06182/9203-15

JAPAN
Applied Microsystems Japan, Ltd.
Nihon Seimei
Nishi-Gotanda Building
7-24-5 Nishi-Gotanda
Shinagawa-Ku
Tokyo T141, Japan
3-3493-0770
Fax 3-3493-7270

U.S. REGIONAL SALES OFFICES

Western Region
Applied Microsystems Corporation of Washington
3333 Bowers Avenue
Suite #220
Santa Clara, CA 95054
(408) 727-5433
Fax (408) 727-9011

Applied Microsystems Corporation of Washington
25909 Pala Place
Suite #280
Mission Viejo, CA 92691
(714) 588-0585
Fax (714) 588-1476

Central Region
Applied Microsystems Corporation
14643 Dallas Parkway
Suite 230, LB-76
Dallas, Texas 75240
(214) 991-6344
Fax (214) 991-4581

Eastern Region
Applied Microsystems Corporation of Washington
6 Cabot Place
Stoughton, MA 02072
(617) 341-3121
Fax (617) 341-0245