Applied
Microsystems
Corporation

ES 1800 Satellite Emulator Reference Manual For 68000 Family Microprocessors

Applied
Microsystems
Corporation

ES 1800 Satellite Emulator Reference Manual For 68000 Family Microprocessors

P/N 920-00069-04
August 1988

Table of Contents

Table of Contents

Preface

PREFACE

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

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

The ES1800 Emulator is intended for use in developing, debugging, and testing Motorola 68000/68008/68010/68020 microprocessor-based systems. This manual assumes that the user is familiar with the terminology and capabilities of the Motorola series microprocessor.

Unpacking and Inspection

Your 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 units are present. If you find any damage, file a claim with the carrier and notify Applied Microsystems Corporation (Customer Service 1-800-426-3925).

STANDARD EQUIPMENT

- Emulator chassis with power cord
- Main control boards and pod assembly
- *ES1800 Emulator Reference Manual for 68000 Series Microprocessors*

OPTIONAL EQUIPMENT

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

Warning

Service

If the ES1800 unit needs to be returned for repairs, Applied Microsystems Customer Service will issue a return authorization number. To obtain the necessary return authorization number and shipping information, call 1-800-426-3925, and ask for Customer Service. After the expiration of the warranty period, service and repairs are billed at standard hourly rates, plus shipping to and from your premises.

Limited Hardware Warranty

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

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

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

Hardware Extended Warranty

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

Hardware Service Agreements

SERVICE AGREEMENTS are available for purchase at any time for qualified Applied Microsystems Corporation manufactured products. The SERVICE AGREEMENT covers the repair of electrical and mechanical parts for defects in materials and workmanship. For information, contact your local sales office.

SECTION 1

Table of Contents

Introduction

INTRODUCTION

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

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

Information in this manual applies to Motorola 68000/68008/68010/68020 microprocessors only. For more complete information on your particular chip, refer to the reference manuals: *16 Bit Microprocessors Users Manual* (for the 68000/08/10 microprocessors) or the *MC68020 32 Bit Microprocessors Users Manual* (for the 68020 microprocessor) published by Motorola Corporation.

How to Use This Manual

This manual is your guide to using the Applied Microsystems Corporation's ES1800 Emulator for Motorola 68000/08/10/20 microprocessors. For your first time using the ES1800, read through the Introduction and Getting Started sections and refer to the Hardware section to make sure your hardware is set up correctly.

Once you are familiar with the Emulator, the sections on the ES language, system commands and target commands provide information on all of the available commands. The comprehensive Index and Appendix A: ES Language Mnemonics are useful for finding specific information in the manual.

The manual is organized as follows:

Section 1: Introduction introduces Applied Microsystems Corporation's ES1800 Emulator for the 68000/08/10/20 microprocessors. It explains emulation, set-up, and configuration requirements, and provides an overview of the features of the ES1800.

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

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

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

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

Section 6: Target Commands is a reference to commands that directly control the target system. It is divided into sections on running the target program, memory commands, the line assembler, the memory disassembler, memory mode, and diagnostics.

Section 7: 68000/08/10 Event Monitor System explains the powerful breakpoint and control system, including the structure of the system, breaking emulation, counting events, using special interrupts, and tracing events. (This chapter pertains to the 68000, 68008, and 68010 microprocessors.)

Section 8: 68020 Event Monitor System explains the powerful breakpoint and control system, including the structure of the system, breaking emulation, counting events, using special interrupts, and tracing events. The different trace modes that pertain only to the 68020 microprocessor are covered, as well as setup shortcuts.

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

System Setup

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

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

The stand alone environment (refer to diagram on page 1-6) consists of the Emulator and a dumb terminal or equivalent connected to the terminal port. This setup can debug target systems with software already installed, or short hand-entered routines. The stand alone configuration is common in manufacturing test and service facilities.

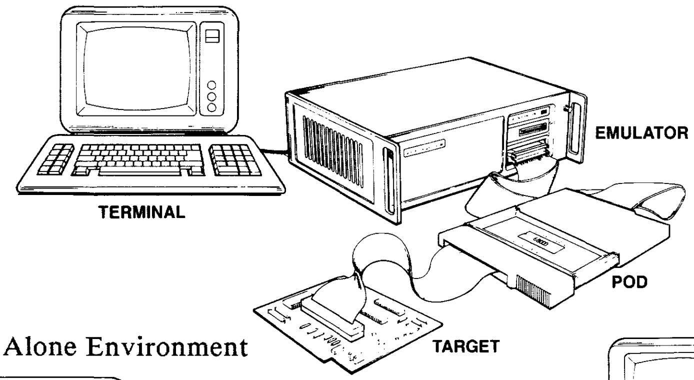
The Emulator can also use data stored in a host development system by setting up a hosted environment (refer to diagram on page 1-6). The Emulator is still under the direct control of the CRT but can load data from the host system's data files.

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

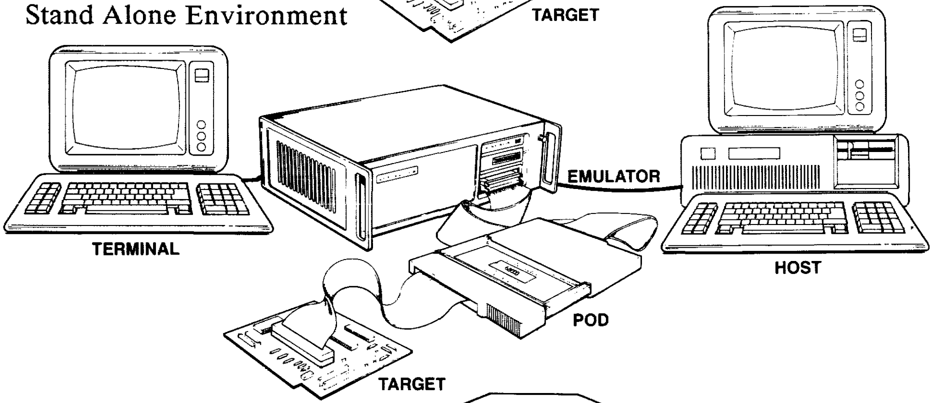
Software for driving the Emulator is available from Applied Microsystems Corporation for the IBM PC and compatibles, SUN, APOLLO and VAX.

The Emulator can also be totally controlled by your host system. This hosted software environment (refer to diagram on page 1-6) requires special host resident software. Drivers and High Level Debuggers are available from Applied Microsystems for most languages and host systems.

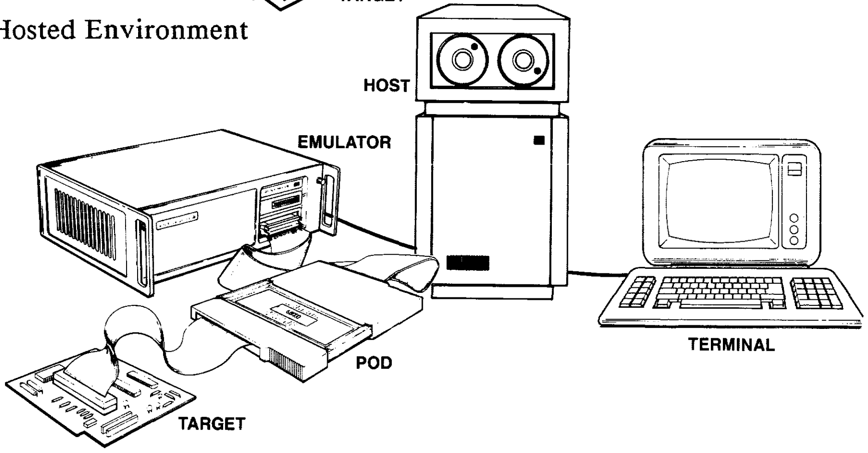
Figure 1. Environments



Stand Alone Environment



Hosted Environment



Hosted Software Environment

System Operation

OVERVIEW

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

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

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

ES LANGUAGE

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

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

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

REAL TIME

Since the pod processor is identical to your target microprocessor, the target system runs in real time. No wait states are inserted by the Emulator during run mode. Additionally, the Emulator's internal clock provides a completely self-contained microprocessing environment that allows you to execute your software without connecting the Emulator to your target system, using the null target.

TRACE MEMORY

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

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.

OVERLAY MEMORY

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

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

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

EVENT MONITOR SYSTEM

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

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

OPTIONAL SYMBOLIC DEBUGGER

The symbolic debug option allows you to assign frequently used values to symbol names that are easily remembered. Features include:

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

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

OPTIONAL LOGIC STATE ANALYZER (LSA)

LSA inputs can qualify event specifications in the Event Monitor System. In the simplest form, specific bit patterns at the LSA inputs can cause a breakpoint. The LSA comparator can detect complex event specifications as well. The LSA provides an additional means of monitoring information from other parts of the target system, beyond the normal recording of trace data performed by the Emulator. This is useful when monitoring (1) buffers suspected of failure, (2) decode logic, (3) memory management circuit translations, and (4) asynchronous external events.

DIAGNOSTIC FUNCTIONS

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

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

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

SECTION 2

Table of Contents

Getting Started

GETTING STARTED

Introduction

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

Emulator Set-up

1. Refer to page 3-1 and verify that proper grounding and power requirements have been met.
2. Remove the front cover of the Emulator by turning the thumbscrews counterclockwise. The pod and LSA pod may need to be unplugged in order to do this.
3. Verify that the main control board and the memory control board are in the top two slots of the Emulator chassis.
4. Verify that the trace/break board is in the third bus slot of the Emulator chassis.
5. If you are using overlay memory, verify that the overlay memory master, and/or master and slave boards, if needed, are inserted.

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

7. Verify that all boards are firmly seated in their motherboard connectors.

8. Set the thumbwheel switch on the main control board for your particular system variables (see page 3-5).

   System default variables in switch position 0 are:

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

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

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

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

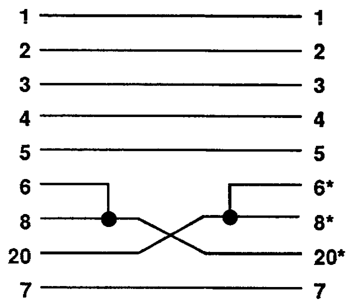
11. OPTIONAL: Connect Logic State Analyzer pod (see page 3-11).

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

13. Verify that the RS232C baud rates and data requirements are set the same on both the Emulator and the terminal (see pages 3-5).

NOTE: the 68020 microprocessor requires different pod connections depending on the trace memory mode you are intending to use (refer to page 3-8)

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



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

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

15. If you plan to use dynamic RAM, read the pages on the ON-OFF switches CAS and TAD. CAS controls whether the address strobe is active in pause mode (page 5-13), and TAD controls whether the emulator address bus is tri-stated during pause mode (page 5-24).

Target System Set-up

1. Check that the target has a 48 pin package dual in-line socket for the 68008 chip, a 64 pin package for the 68000/68010 chips, and a 114 pin grid array for the 68020 chip.
2. Check that a good ground exists at the microprocessor socket using an ohmmeter.
3. Validate the power supply at the microprocessor socket. It should read 5V ±10%.
4. Check for a valid clock signal at the target microprocessor socket.
5. Turn off target system power and Emulator power.
6. Plug in the probe tip (see page 3-8).

Power-On Sequence

TARGET SYSTEM PRESENT

1. Turn on the target system.
2. Turn on the Emulator.
3. Reset the target system (see page 6-14).

NO TARGET SYSTEM

1. Verify that the pod is connected to the Emulator (see page 3-8).
2. Remove the conductive rubber probe tip protection and install Null Target Software Simulation Tool (see page 3-10).
3. Power-up the Emulator.
4. The log-in banner is displayed (see page 4-25).

When you power-up the Emulator, all registers, maps, event clauses, and system variables are either cleared or set to default values. Examine the **SET** and **ON** menus (see pages 5-3 and 5-9) and configure the system as desired. Your special setup can then be stored in EEPROM (see page 5-25). By setting the rotary switch on the controller board to the proper position, your special setup can be autoloaded on power-up.

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

Test Run of System

Use this test guide after the system configuration is correct and the ES prompt is displayed ($\boxed{>}$).

A system test run consists of the following 9 steps:

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

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

If you encounter difficulty with the ES1800 hardware, call the Applied Microsystems Corporation Customer Service hot line at 1-800-426-3925 for assistance.

1. INITIALIZE THE EMULATOR

Enter the following to initialize the Emulator:

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

2. MAP RAM MEMORY

Map all of the overlay memory available to the Emulator.

```
>MAP 0 TO XXXX
```

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

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

For example, to map 64K, enter:

```
>MAP 0 to 0FFFF
```

For more information, refer to page 5-55.

3. TEST RAM

Test all overlay memory installed by entering:

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

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

If RAM test continues to fail, reseat all PAL and RAM chips on the overlay memory.

For more information, refer to Diagnostic Functions, page 6-49.

4. ENTER PROGRAM

Enter a short program by invoking the line assembler and entering 68000 op codes (see page 6-34).

```
 >ASM 400
**** 680XX LINE ASSEMBLER ****
 000400>NOP
 000402>/
 000404>/
 000406>/
 000408>/
 00040A>BRA $400
 00040C>X
```

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

5. VERIFY THE PROGRAM

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

```
>PC = 400
>STP; DT
>/
>/
>/
>/
>/
```

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

For more information on the **STP** command, refer to page 6-8.

6. RUN THE EMULATOR

Enter **RUN**.

```
 >RUN
R>
```

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

7. STOP THE PROGRAM

Enter **STP** to stop.

```
R>STP
```

The Emulator should stop running and display the PC register value.

8. DISPLAY THE TRACE BUFFER

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

```
>DRT
```

The display should show sequence numbers between 0 and 20, and address values between 400 and 40E. Then enter **DTB**.

```
>DTB
```

The display should show a disassembled trace of the program with NOPs and BRA $000400.

9. SET A BREAKPOINT

Set a breakpoint to verify that the Event Monitor System will halt execution when a defined condition is met. In this case, the Emulator executes 100 (hex) bus cycles, then breaks.

Enter:

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

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

If a break does not occur:

1. Enter **STP**.
2. Set PC to 400.
3. Enter **DES 1** and verify that you have entered the WHEN/THEN statement and comparator values as shown above.
4. Type **RBK** again.

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

SECTION 3

Table of Contents

Hardware

HARDWARE

Emulator Chassis Assembly

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

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

WARNING!

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

SYSTEM GROUNDS

The ES1800 Emulator has three grounding systems:

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

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

WARNING!

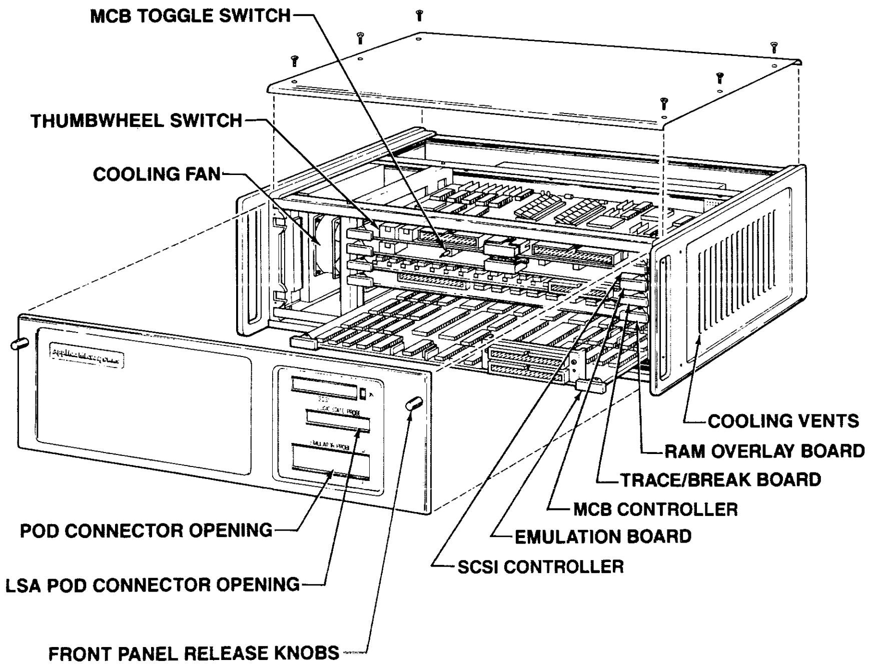
Failure to ground the system properly may create a shock hazard.

Emulator Control Boards

Removing the front panel of the Emulator chassis exposes the chassis card cage as shown on page 3-4. This panel is opened by turning two knobs, located in the upper corners of the front panel, counterclockwise.

Main Control Board	The main control board holds the controlling 6809 CPU for the Emulator, the EEPROM, two serial ports, and RAM. The 16-position thumbwheel switch located on this board determines the system variables and serial line baud rates for autoloading on power-up. Refer to page 3-5, for each switch position setup. Switch position 0 autoloads default system variables.
Memory Control Board	The memory control board holds the memory management logic and optional symbolic debug memory. The three-position toggle switch located below the main control board thumbwheel switch must be in the center position. If the toggle switch is in either of the other two positions the Emulator will not work properly.
Trace/Break Board	The trace/break board holds trace memory, the Event Monitor System, and the logic state analyzer (LSA) interface.
Overlay Memory Board(s)	The overlay memory board set is optional and can hold 32K, 64K, 128K, 256K or 512K of memory. 512K of memory requires a slave board.
Emulation Board	There are six different emulation boards depending on the target microprocessor you are using.

Figure 2. Control Boards



Emulation Board
Thumbwheel Switch Settings

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

----

*Factory Default Parameters

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

Emulator Chassis Rear Panel

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

Serial Ports

The two serial ports are RS 232C ports labeled $\boxed{\text{TERMINAL}}$ and $\boxed{\text{COMPUTER}}$. Pins are discussed on page 3-12.

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

Trigger Output

The ES1800 Emulator provides a TTL trigger strobe output controlled by the Event Monitor System. The trigger output is available at a BNC connector on the rear panel of the chassis and on a clip lead attached to the optional Logic State Analyzer (LSA) pod. Refer to Section 7 (68000/08/10) or Section 8 (68020) for information on Event Monitor System actions.

The trigger can be used for such things as:

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

Power Switch

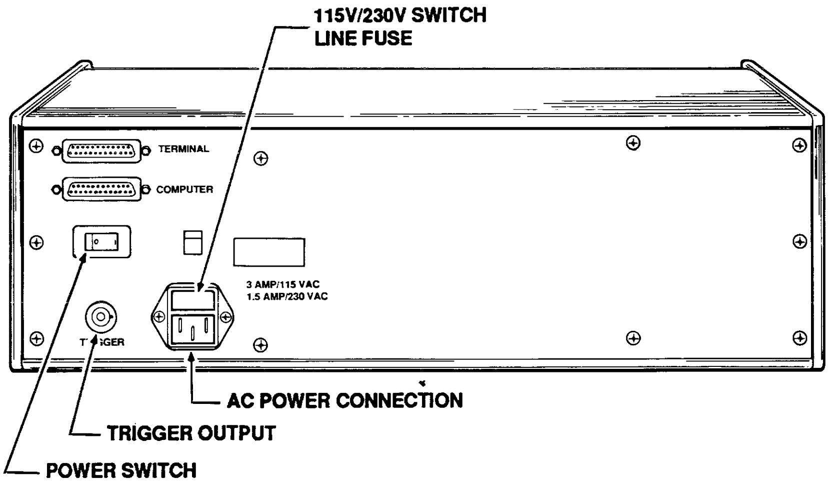
Before powering up, two items should be checked:

1. Proper grounding of power cable (see page 3-1).
2. Proper power-up sequence of Emulator, target system, and/or peripheral equipment. (See Power-On Sequence, page 2-5.)

Line Fuse

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

Figure 3. Rear Panel



Pod Assembly

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

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

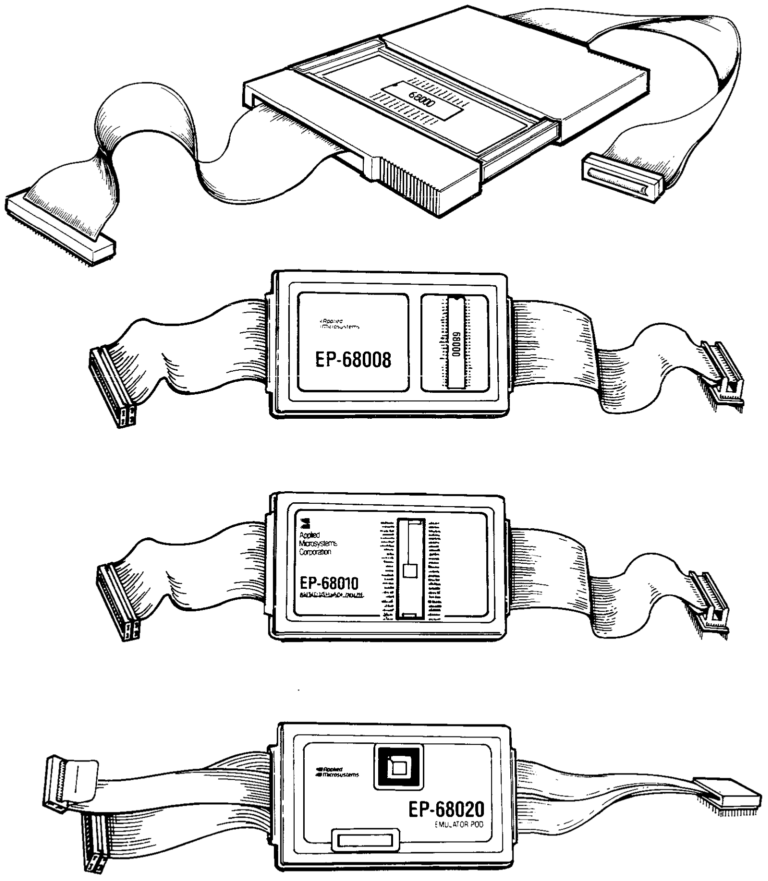
68020:

The 68020 microprocessor has four trace memory modes. Each mode requires a different Emulator/pod assembly configuration.

The 68020 pod has three ribbon cables that connect to the emulator chassis. All three cables are located on the same side of the 68020 pod. The bottom two ribbon cables for all four trace modes, always connect to the Emulator chassis' emulation board. For modes 1, 2, and 3 the top ribbon cable connects to the Emulator chassis trace and break board. For mode 0 this top cable is not used and in its place the LSA pod assembly is connected to the trace and break board. The lower six LSA bits are available in mode 3 if you choose to use them. To do so, the LSA pod assembly connects to the front panel of the 68020 pod.

Refer to page 8-14 for more information on 68020 trace modes.

Figure 4. Pod Assemblies



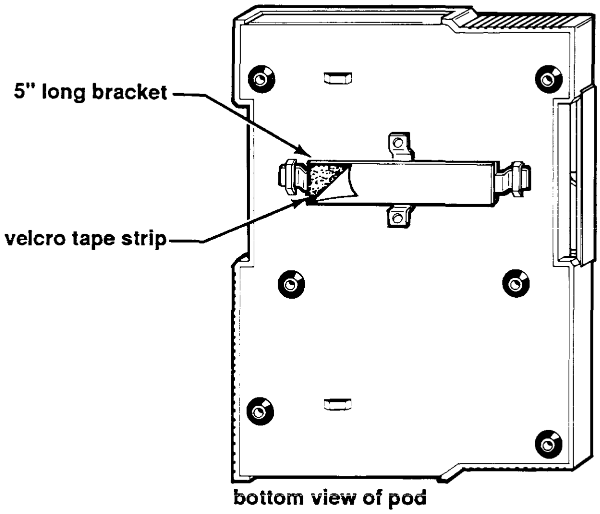
Saving Desk Space

To save limited desk or table space, the ES 1800 68000 16.67 MHz pod 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 have to bend the bracket slightly). Figure 1-1 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 1-1. You can now attach the 68000 pod to any surface that adheres to velcro, such as many types of office partitions.

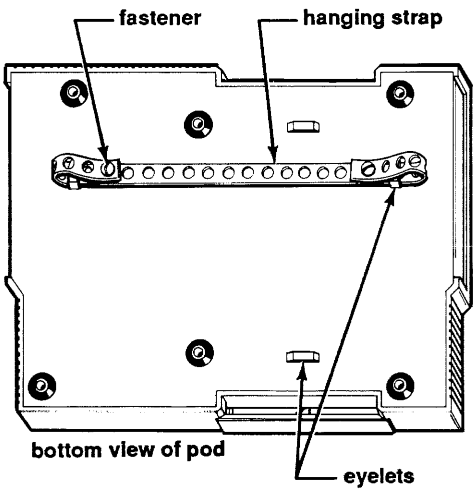
Figure 3-1: Velcro Tape Support



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 1-2 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 3-2: Hanging Strap Support



Null Target Software Simulation Tool

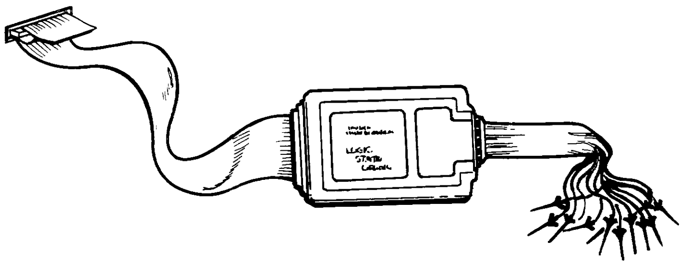
The Null Target Software Simulation Tool is a device that is about the size and shape of a microprocessor chip. It is to be plugged onto the end of your pod assembly probe tip adapter. This device allows you to run the ES1800 Emulator without having to be connected to a target system.

Logic State Analyzer (LSA) Assembly

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

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

Figure 5. Logic State Analyzer Pod Assembly



Serial Ports

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

Baud Rate	Baud rates and data lengths for each port are independent. Refer to the **SET** command (page 5-3) for available baud rates on each port.
Port Control	Only one port can be the controlling port. Either port can give control to the other port.
Upload/Download	The Emulator accepts commands to begin uploading/downloading from either port. However, the Emulator will upload/download hex format data files only through the computer port.

Your system configuration determines which port should be in control. Refer to page 1-4.

PIN CONFIGURATIONS

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

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

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

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

DATA REQUIREMENTS

Stop Bits

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

Parity

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

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

Hardware Handshake

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

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

Software Handshake XON XOFF

The ES1800 uses normal flow control codes to control software handshaking. The default values are XON CTRL Q and XOFF CTRL S. These values can be changed by the user (see page 5-4).

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

Although the user will never overfill the input buffer from a controlling terminal, a controlling computer is quite capable of doing so. The input buffer for the computer port is 64 characters deep. When eight characters have been placed in the computer input buffer, the XOFF character is transmitted. Allowing two character times for skew, the computer must transmit no more than 54 characters until the next XON from the ES1800.

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

Maintenance

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

CABLES

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

PROBE TIP ASSEMBLY

The probe tip assembly is the small header assembly that plugs into the target system microprocessor socket. The most obvious area to inspect is the adapter, as the pins can be broken during insertion or extraction.

The adapter can be protected by installing a microprocessor socket (male-female) onto the adapter. If a pin is then broken on the socket, it is easier to replace because of its common usage.

CLEANING THE FAN FILTER

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

1. Unplug the ES 1800.

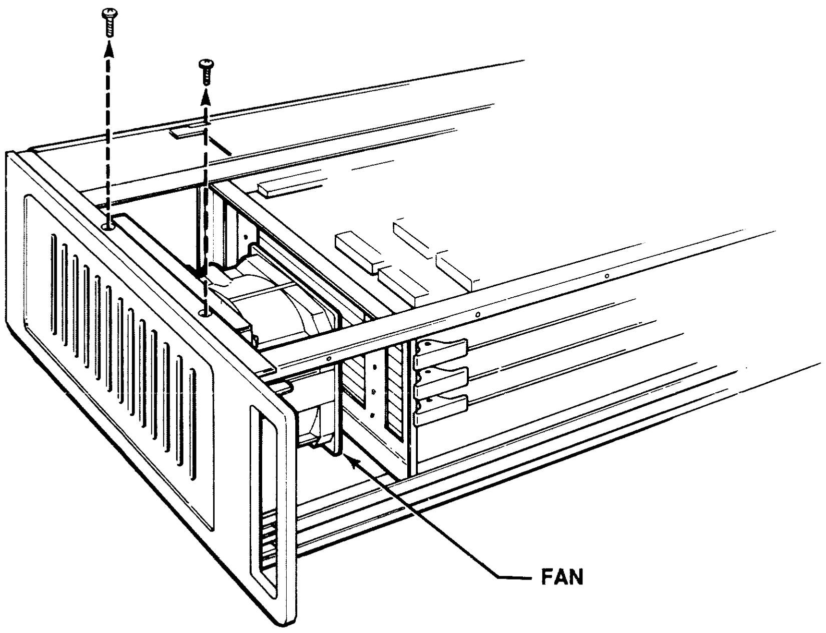
WARNING!

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

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

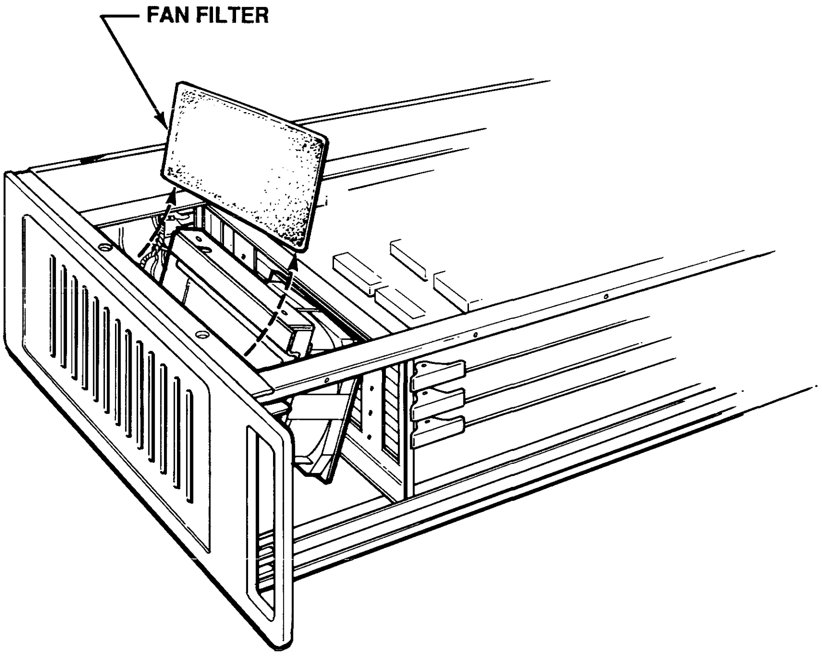
4. Unscrew the two screws at the top of the chassis which hold the fan in place.

ES 1800 Fan Mounting



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

ES 1800 With Fan Titled 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:

```
48 pin adapter (68008 chip)
64 pin adapter (68000/68010 chips)
114 pin grid adapter (68020 chip)
Short cable set
Long cable set
```

Troubleshooting

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

The most common problems encountered are listed below. We recommend that you contact Customer Service at Applied Microsystems Corporation (1-800-426-3925) if you experience any problems that do not fall within this range of items. Before you call our service department, display your software revision number by typing **REV** (page 5-135). You will be asked for this when you call.

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

Troubleshooting

SYMPTOM	POSSIBLE CAUSES
Target system runs erratically	1. Faulty interconnect cables
	2. Broken pin on adapter
	3. Emulator and target system not compatible
	4. LDV not executed before RUN (vector not loaded).
Emulator will not communicate over RS232	1. Baud rate set incorrectly.
	2. Target system requires "null" modem cable (pin 2 and pin 3 of RS232 connector reversed).
	3. For terminal operation, thumbwheel switch located on the top card is not in the "0" position or the cable is not properly attached to the terminal port in the back of the ES1800.
	4. Cable not going to correct port of the terminal or PC.
	5. Toggle switch located on the second card from the top in the ES1800 not in the middle position.
	6. Power LED not on.
	7. Cards not seated properly.

ES1800 Emulator Specifications

INPUT POWER

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

ENVIRONMENTAL

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

PHYSICAL

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

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

SECTION 4

Table of Contents

ES Language

ES LANGUAGE

Structure of the ES Language

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

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

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

ES Language Syntax

Language Element	Example
Command Line	
[Repeat] Command Statement [;Command Statement] ... <RETURN>	
Single Character Instant Command	
Repeat	
<*>	`*STP;DT`
<*><Repeat limit>	`*9 STP;DT`
Repeat Limit:	
Decimal number only (1 to 2^{32}-1)	`87651234`
Command Statement	
Command Mnemonic	`DTB`
Command Mnemonic <Expression>	`MM PC + 4`
Command Mnemonic <Expression List>	`SET #20,#14`
Assignment Command	`D0 = 0FA9`
Expression	`2 * GR5`
Event Monitor System Control Statement	`WHEN AC1 THEN BRK`
Single Character Instant Command	
</>	
<,>	
<.>	
<?>	
Command Mnemonic	
<1 or more alpha characters>[1 or more decimal characters]	
Expression	
[Unary Operator] Ivalue	`-2473`
Ivalue <Operator> Expression	`2 - 3F6C90`
<@> Expression	`@240;@@A7`
<(> Expression <)>	`2 * (-2 + 3)`

ES Language Syntax *(cont)*

Language Element	Example
Ivalue:	
Symbol	`'main`
Nvalue	
Symbol:	
<'><1 or more printable characters><space or return>	
Nvalue:	
Number	`7FA36`
Register Name	`A0`
Register Name:	
<1 - 3 Alpha characters>[0 - 2 decimal digits]	
Number:	
[Base]<1 or more digits>	`%0101001`
Base:	
<%> (binary)	
<\\> (octal)	
<#> (decimal)	
<$> (hexadecimal)	
Expression List	
Expression <,> Expression [,Expression list]...	`1,PC,2+2,-6`
Assignment Command	
Svalue <=> Expression	`A6 = @0FFFF0`
<@> Expression <=> Expression	`@(A6 + 5) = GD3`
Svalue:	
Symbol	`'Test_result`
Register Name	`MMP`

ES Language Syntax *(cont)*

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

ES Language Syntax *(cont)*

Language Element	Example
Action:	
<BRK>	
<TRC>	
<TOT>	
<CNT>	
<TOC>	
<RCT>	
<TGR>	
<FSI>	
<GRO Group>	`GRO 3`
Unary Operator	
<ABS>	`ABS GD3`
<!>	`!0AA`
<->	`-3`
Operator	
Mul.op	
Add.op	
Shft.op	
<&>	`GD4 & 0FF`
<^>	`DC2.3 ^ 0FF00`
Mul.Op	
<*>	`2 * 3`
</>	`0FAC / %01001`
<MOD>	`GD5 MOD 7`
Add.op	
<+>	`GR0 + A7`
<->	`@(A7 - 4)`
Shft.op	
<<<>	`DC1 << 3`
<>>>	

Notes on ESL

Command Line

A command line is created by entering one or more characters after the ESL prompt (see page 4-27 for a description of the various prompts). One or more command statements can be placed on a single command line. Multiple command statements must be separated by a semicolon. The command line is limited to 76 characters and must be terminated with a return. The only way to extend command lines is by using macros (see page 5-117).

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

Repeat

If an asterisk ($\boxed{*}$) is the first character on the command line, the entire command line will be repeated 2^{32} times. 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 (see page 6-42), entering a $\boxed{\text{RETURN}}$ on an empty line causes the next location to be read. Entering a value followed by $\boxed{\text{RETURN}}$ will cause that value to be written to memory. The line assembler (see page 6-34), memory disassembler (page 6-39), and main help menu (page 4-20) 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 ($\boxed{/}$) will cause the previously entered command line to be repeated.

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

This example will single step three times.

The comma ($\boxed{,}$) executes macro 1 and the period ($\boxed{.}$) executes macro 2. However, if you are in memory mode, the period moves you to the next higher memory address while the comma moves you to the next lower address.

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

A $\boxed{?}$ entered at any other time (i.e., not after an error), causes a two-page help menu to be displayed. A $\boxed{\text{RETURN}}$ moves you from the first page to the second. Any other character terminates the help menu.

Command Mnemonics

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

Expression

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

Parentheses may be used to alter the normal precedence of operations. The Emulator recognizes parentheses just as they are treated in algebraic equations. As many levels of parentheses as needed can be used. 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) = PC + #100 * (D0 >> 4) +0AF34
```

This example retrieves the value of the D0 register, shifts it right 4 bit positions (divide by 2^4), multiplies the result by 100 decimal, adds 0AF34 and the contents of the PC register, and writes the result to the location 3 bytes above the address in GD0.

A more common and useful example might be:

```
ASM PC + 4
```

This computes the address PC + 4, and starts up the line assembler at that address. The expression:

```
'interrupt + 1A6
```

by itself will add 1A6 to the current value of the symbol interrupt and display the result. If you 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 page 4-27 for more information on BYT/WRD/LWM modes.)

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

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

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

Symbols

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

Numbers

The ES1800 has a default base register. It is assumed that numbers entered without a leading base character are being entered in the default base. Generally, the default base is hexadecimal (factory default). See page 5-85 for more information in changing the default base register.

There are three different types of numbers. An integer is a 32 bit signed value.

A don't care is a 32 bit value with a 32 bit mask. For each bit set in the mask, the corresponding bit position in the value is ignored during Event Monitor comparisons. Don't cares can be entered in two ways. `1234 DC 0FF0` is explicit. `1XX4` is equivalent to `1FF4 DC 0FF0`. Don't cares are useful for setting the Event Monitor System Event Comparators (see pages 7-2, 8-2).

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

Ranges, don't cares, and integers are not generally interchangeable. Certain registers can only hold certain data types. All registers can hold integers. Address type registers cannot be loaded with don't care values. Status and data registers cannot be loaded with range values. See page 5-68 for a list of all registers and their data types.

Base

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

Expression List

Lists are required by a few commands. They can also be used for implicit evaluation. For example, in pause mode, entering the three numbers `%010011010, #128, \77347` causes the Emulator to display their equivalent in the default display base (usually hexadecimal). Lists are limited to nine elements. Lists are used in memory mode 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 `@A7 = 0` are legal and useful.

The following example assigns the value on the right ($47FF) to the A0 register.

```
A0 = $47FF
```

The Emulator displays nothing in response to this entry.

Verify by:

```
>A0
$000047FF
```

In the following example the expression $121 + $4 is calculated.

```
A0 = $121 + $4
```

The register A0 is then assigned the value $125. Your console screen will not display this entry. Verify by:

```
 >A0
$125
```

Registers are grouped into three types: integer only, don't care, and range. Any register can be assigned an integer value. Don't care registers can be loaded with don't care values or integers but not ranges. Range registers can be loaded with integers or ranges but not don't care values. See page 5-68 for a list of all registers and their data types.

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, word, or long 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 $\boxed{\text{GD4}}$. The command interpreter accesses the target system location pointed at by $\boxed{\text{GD4}}$, adds six to the value stored there, and displays the final results.

In the second example, the Emulator displays the value stored in the sixth location above the address pointed to by $\boxed{\text{GD4}}$.

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

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

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

Event Monitor System Control Statement

Event Monitor System statements describe combinations of target program conditions and the corresponding actions to be taken if the conditions are met, not 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 (68000/08/10) or Section 8 (68020), 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: `3 WHEN AC1 THEN BRK` is functionally the same as `WHEN AC1.3 THEN BRK`; both use group 3. You cannot mix group numbers within a single EMS control statement.

Event

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

Disjunctive

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

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

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

Conjunctive

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

```
AC1 AND DC1 OR DC2
```

This event is equivalent to `AC1 AND DC1` in one statement and `DC2` in another. If you are looking for two different data values at an address, 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 `$FFFFFFFF`. Internal math however, is signed only for the `+`, `-`, `*`, `/` operations; `-5+3` is `$FFFFFFFE`, while `-1 >> 1` is reduced to `$7FFFFFFF`.

ABS converts a signed number to its absolute value.

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

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

Operator

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

Mul.op:		
	*	Multiply
	/	Divide
	MOD	Modulo
Add.op:		
	+	Add
	-	Subtract
Shft.op:		
	>>	Right shift
	<<	Left shift
&		Logical AND
^		Logical OR

*Modulo (*MOD*)*

The result of this operation is the remainder after the value on the left has been divided by the value on the right. For example, `>29 MOD 4` results in 1.

Results of Single-Argument Operators

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

Results of Dyadic Operator Combinations

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

Help

There are two pages of help information available. The 68020 help menu differs from the 68000/08/10 menu. Enter a $\boxed{\text{?}}$ 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 $\boxed{\text{RETURN}}$ at the end of the first page to move to the second page. The menus are shown on the next four pages.

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

Software Switches	Enter either $\boxed{\text{ON}}$ or $\boxed{\text{OFF}}$ to display the current settings and definitions of all software switches, (see page 5-9).
Communications Set-up	Enter $\boxed{\text{SET}}$ to display the current configuration settings and possible values (see page 5-3).
Special Diagnostic Functions	Enter $\boxed{\text{SF}}$ to display a list of the available special functions (RAM/ROM tests, scope loops, etc.) (see page 6-50).

First Page of Help Menu (68000/08/10)

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

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

MEMORY - REGISTER COMMANDS:            DR - DISPLAY ALL CPU REGISTERS
 DB X TO Y - DISPLAY BLOCK             FILL X TO Y, Z - FILL BLOCK WITH Z
 BMO X TO Y, Z - BLOCK MOVE TO Z       LOV/VFO X TO Y - LOAD/VERIFY OVERLAY
 MMS = ALT, COD, DAT, STA              DEFINES STATUS LINES FOR MEMORY ACCESS
 X - EXIT MEMORY MODE                  M X - VIEW/CHANGE MEMORY AT X

MEMORY MAPPING:                        OVE = DC, DAT
 MAP X TO Y :RO  :RW  :TGT  :ILG       DM/CLM - DISPLAY/CLEAR MEMORY MAP

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

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

Second Page of Help Menu (68000/08/10)

```
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
CL      -       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
CL.2 - 24;AC2.2 = $F000 LEN $400          WHEN CL.2 OR AC2.2 THEN BRK
```

First Page of Help Menu (68020)

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

TRACE HISTORY:
DRT X TO Y - DISPLAY TRACE RANGE      DRT (X)-DISPLAY PAGE RAW TRACE (FROM X)

MEMORY - REGISTER COMMANDS:           DR - DISPLAY ALL CPU REGISTERS
 DB (.BWL) X TO Y - DISPLAY BLOCK     CLR - CLEAR ADDRESS AND DATA REGISTERS
 FIL X TO Y,Z - FILL BLOCK WITH Z     VBL X TO Y,Z - VERIFY BLOCK CONTAINS Z
 BMO X TO Y,Z - BLOCK MOVE TO Z       LOV/VFO X TO Y - LOAD/VERIFY OVERLAY
 VBM X TO Y,Z - VERIFY BLOCK MOVE     X - EXIT MEMORY MODE
 M (X) - VIEW/CHANGE MEMORY (AT X)    .B,.W,.L - DATA LENGTH; BYTE, WORD, LONG
 MMS/MMD = SCO-SC7,SP,SD,UP,UD,CPU    STATUS FOR MEMORY ACCESS; SOURCE/DEST

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
 BUS - VIEW HARDWARE STATUS LINES     SZ (.BWL) - SET DEFAULT DATA LENGTH
 ON/OFF - VIEW/ALTER SWITCHES         SF - VIEW/EXECUTE SPECIAL FUNCTIONS
 MAC/CMC - DISPLAY/CLEAR MACROS       DIS/ASM (X) DIS/ASSEMBLE FROM/TO MEMORY
 LD/SAV (X) - LOAD/SAVE 0=SETUP,1-REGS,2-EVENTS,3=MAP,4=SWITCHES,5=MACROS
```

Second Page of Help Menu (68020)

```
MEMORY MAPPING:                          OVE = SCO-SC7+SP+SD+UP+US+CPU; OVS = 2-5
MAP X TO Y :RO  :RW  :TGT   :ILG         DM/CLM - DISPLAY/CLEAR MEMORY MAP

EVENT MONITOR SYSTEM
DES/CES (X) - DISPLAY/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
                          + SP/SD/UP/UD/SCO-SC7 + IPO .. IP7 + BER _ AV + IP
LSA     -       16 LOGIC STATE LINES, MAY INCLUDE DON'T CARE BITS
CL      -       COUNT LIMIT, ANY NUMBER 1 TO 65,535

STEP 1 - ASSIGN EVENT DETECTORS          STEP 2 - CREATE EVENT SPECIFICATIONS
AC1 = $1234;S1 = SP + RD                 WHEN AC1 AND S1 THEN GROUP 2
AC1.2 = $4576+14*6;DC2.2 = $5600 DC $FF  WHEN AC1 AND NOT DC2 THEN CNT
CL.2 - 24;AC2.2 = $F000 LEN $400         WHEN C<.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.

```
COPYRIGHT 1985
APPLIED MICROSYSTEMS CORPORATION
SATELLITE EMULATOR 68XXX VX.XX
USER = ___ SW= ___
#___K AVAILABLE OVERLAY
```

Satellite Emulator 68XXX	The microprocessor type is that of the target system. Refer to the Motorola reference manual for more information.
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 Emulator main control board (refer to page 3-5).
#____K AVAILABLE OVERLAY	The amount of overlay memory indicated depends on the amount installed in the system. This can be 32, 64, 128, 256 or 512 K of memory.
>No Target VCC	The console screen displays a NO TARGET VCC (see error message page B-3) 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 page 3-10 Null Target System Simulation Tool, for using the ES1800 Emulator without a target system.

Prompt

The pause mode prompt $\boxed{>}$ indicates that the Emulator is not running, is in a pause mode and is ready to receive instructions. Make sure that the $\boxed{>}$ 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.

Check for proper connection of the cable between the CRT and the Emulator.

Check the cable connecting the pod to the Emulator. 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 Emulator chassis. Check for loose boards.

If an error message appears, refer to the given message in Appendix B.

Prompts

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

>	The standard, or pause mode prompt from ESL consists of a space character followed by a right arrow.
R>	During emulation, the run mode prompt is displayed. Most ESL commands are still valid.
$12345678 $00 > *$12345678 $00 R>* *$12345678 $0000 >* *$12345678 $0000 R>*	In memory mode, the prompt includes the memory address and the data contained there. Depending on whether byte mode, word mode, or long word mode (BYM, WDM, LWM) has been chosen, the data will be a byte, a word or a long word. The "run" prompt ($\boxed{R>}$) may also be present during memory mode.
$000000 $FF> *$000000 $FFFF>* *$000000 $FFFFFFFF>*	These three memory mode prompts will display to indicate whether the system is in an 8 bit, 16 bit or 32 bit mode, respectively, for displaying your address and data.
**** 680XX LINE ASSEMBLER **** *000100 >*	The 68000/08/10 line assembler displays a 16 bit address prompt. The 68020 line assembler displays a 32 bit address. This prompt contains an $\boxed{R}$ if you are assembling during emulation.

Special Modes

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

Byte Mode/
Word Mode/
Long Word Mode

The **BYM**, **WDM** and **LWM** commands select byte, word and long word mode operation. The mode selected determines whether 8, 16 or 32 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 and long word modes.

More compatible with the Motorola style of value display is the **SZ** command. To designate globally what memory address and data pattern the system will display enter the **SZ** mnemonic, specify the mode with a dot operator and press return: `SZ <.B,.W,.L>`

Memory address and data patterns will globally display in the specified mode: `SZ.B` displays in byte mode. `SZ.W` displays in word mode. `SZ.L` displays in long word mode.

You can temporarily override the byte, word and long word address and data display prompts by keying in the dot operators (`.B`, `.W`, `.L`) 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. `DB.L` means a block of memory is displayed in long word mode.

The dot operators may also be used with the indirection operator (@) e.g. @.B(183+1).

When the indirection operator is used (@), long word mode is the dot operator default extension regardless of the system default mode. To change this you must specify .B or .W.

Line Assembler

The 68000/08/10 line assembler has a single 16 bit address prompt. The 68020 line assembler has a 32 bit 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.

Transparent Mode

No characters are generated by the ES1800. 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 create an indefinitely repeating command that does not display anything. Terminate such commands with the reset character (default is CTRL Z).

Special Characters

DELETE *BACKSPACE*	Either character deletes a character just entered on a command line.
CTRL X	Deletes an entire command line.
CTRL R	Redisplays the current command line (for hardcopy terminals).
CTRL Z	The default reset character. The reset character can be changed by the user (see page 5-3).
ESC ESC	The default transparent mode escape sequence, used to terminate transparent mode. The transparent mode escape sequence can be changed by the user (see page 5-3).
CTRL S	The XOFF character. When issued from the keyboard, the screen display stops scrolling, allowing the user to view the information. The XOFF character can be changed by the user (see page 5-3).
CTRL Q	The XON character. Restarts the screen display after an XOFF is issued. The XON character can be changed by the user (see page 5-3).

Errors

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

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

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

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

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

ES Language Error Messages

1,2,3	EXPRESSION HAS NO MEANINGFUL RELATION TO REST OF COMMAND. Often caused by entering symbols out of context. $\boxed{\text{DR}}$ and $\boxed{\text{BRK}}$ are both legal, but when entered together as $\boxed{\text{DR BRK}}$ cause this error message.
4	PARSE ERROR. . .CALL AMC..
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, a list of only 9 values can be entered on a single command line.
19	INVALID GROUP NUMBER . . . (NOT IN 1-4). There are only four event groups (1-4).

20,21,22,23	OPERATION INVALID FOR THESE ARGUMENT TYPES. Often caused by attempting arithmetic operations on incompatible variables.
24	BASE ARGUMENT MUST BE A SIMPLE INTEGER. Argument should be #0 to #16.
25	ILLEGAL OVS VALUE . . . (NOT IN 1-7). Overlay speed determines the number of wait states inserted for overlay accesses. Only 1-7 are allowed.
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.
28	ATTEMPT TO CHANGE CURRENT OVERLAY SEGMENT (USE 'CLM' FIRST). Overlay memory can only be mapped within a a 16 megabyte range.
29	ILLEGAL DESTINATION - SOURCE TYPE MIX. Caused by trying to store don't care data into a range variable or other similar operations.
30,31	RANGE START AND END ARGUMENTS MUST BE SIMPLE INTEGERS. Cannot use masked values or ranges.
32	RANGE END MUST BE GREATER THAN RANGE START. $\boxed{\texttt{6 len 1}}$ and $\boxed{\texttt{10 to 5}}$ are examples of invalid ranges.

33	RANGE START AND END ARGUMENTS MUST BE SIMPLE INTEGERS. Cannot use masked values or ranges.
34	READ AFTER WRITE-VERIFY ERROR. Data supposedly written to memory during a download operation was read back as a different value. The error message contains the locations and results of the comparison.
35	WARNING - DATA WILL BE LOST WHEN EMULATION IS BROKEN. Caused by assigning values to CPU registers during emulation. CPU registers are copied into internal RAM only when emulation is broken. The RAM contents are copied into the processor only when emulation is begun. The Emulator cannot access CPU registers during emulation. Thus, once emulation has been started the **DR** command shows the contents of the CPU registers as they were before emulation was begun. Changes can be made to these values, but the data will be rewritten when emulation is broken.
36,37,38	NO ROOM . . . BREAKPOINT CLAUSES TOO NUMEROUS OR COMPLEX. Too many WHEN/THEN clauses were entered. 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.
42	ILLEGAL SETUP SET VALUE. Consult **SET** menu for legal values (page 5-3).
43	"WHEN" CLAUSE REDUCED TO NULL FUNCTION. Caused by constructs such as `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 Emulator is warrantied for one year. 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.
64	ARGUMENT TOO LARGE. Caused by entering **DRT** argument that includes numbers greater than #2045.
65	ILLEGAL RANGE.
66	STATUS CONSTANTS CANNOT BE ALTERED. System constants (i.e., BYT, OVL) cannot be assigned values.
67	TOO MANY "WHEN" CLAUSES. You have tried to enter more WHEN/THEN clauses than the Event Monitor System can handle.
68	INVALID DATA FORMAT FOR SYMBOLS. Must use Extended Tektronix Hex.
70	CANNOT INITIALIZE VECTORS DURING EMULATION. LDV, RNV, and RBV can only be entered in pause mode.
71	MUST BE IN TRACE MODE 2 TO DISASSEMBLE TRACE. Refer to **SET** menu and rerun in trace mode 2 before using DTB(3).
72	INCOMPATIBLE EEPROM DATA. Previous data saved to EEPROM was not from an 680XX Emulator system.

74	COMMAND INVALID DURING EMULATION. Must be in pause mode.
75	INVALID RECORD TYPE. Download routine received invalid record type code.
76	NO SYMBOLIC DEBUG. The symbolic debug option is not installed in your system. Cannot assign symbol and section values.
78,79,80	TOO MANY SYMBOLS. Symbols exceeded available RAM. Purge symbols before downloading again.
81	SYMBOL OR SECTION PREVIOUSLY DEFINED. An attempt was made to redefine an existing symbol or section. Section definitions cannot overlap. Symbols should be purged before downloading.
82	SYMBOL NAME IN USE. Symbol name cannot be used more than once. You must delete a section before assigning it a new value.
83	TYPE CONFLICT WITH DEFINED SYMBOL. Please refer to Extended Tekhex specification, page C-7.
87	SECTION TABLE FULL. Too many symbolic section names have been defined.
88	INVALID ARGUMENT SIZE. Operand doesn't fit into destination register.
89	INVALID ADDRESSING MODE.

90	ARGUMENT OUT OF RANGE.
91	INVALID TRAP VECTOR NUMBER.
92	INVALID OP CODE for 68000/08/10..
93	INVALID CONTROL REGISTER.
94	ARGUMENT NOT SYMBOLIC. DEL Requires a symbolic argument.
95	ILLEGAL SPACE CODE.
255	UNKNOWN ERROR.

SECTION 5

Table of Contents

System Commands

Page

SYSTEM COMMANDS

Set-Up Commands

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

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

The **ON/OFF** menu contains switches that control emulation and the serial port copy switch. For example, the copy switch copies data to both serial ports for obtaining hard copy of your emulation session.

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

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

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

SET COMMAND

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

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

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

SYSTEM:    #1 USER = 0; [0,1]
           #2 RESET CHAR = $1A
           #3 XON, XOFF = $11,$13
           #4 TRACE MODE =2; [0,1,2,3] (68020 ONLY)

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

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

(continued)

SET *<parameter>*, *<exp>*	The value of the specified parameter is changed to *<exp>*. If you assign an illegal value to a variable, an error message is displayed, and the value is not changed.
	The table below shows the valid values for each SET variable. All arguments preceeded with a $ indicate that the value entered must be a 7-bit ASCII character.

Comments

In the commands that follow, "#" indicates a decimal number. It is not required for the numbers 0-9.

Parameters	Description	Reset Required
SET #1,#0	User 0	No
SET #1,#1	User 1	No
	Two users may save and load values to the EEPROM. This parameter indicates which user is active when executing the **SAV** and **LD** commands.	
SET #2,$n	Reset Character	No
	The reset character will reset the Emulator and the pod CPU. The system default is CTRL Z ($1A).	

Parameters	Description	Reset Required
SET #3,$n,$m	XON/XOFF Characters	No
	XON and XOFF control the screen scrolling. An XOFF will stop a scrolling display. XON will resume the scrolling. The system defaults are CTRL Q, CTRL S ($13, $11).	
SET #4,	Trace Mode (68020 only)	No
#0	Mode 0	
#1	Mode 1	
#2	Mode 2	
#3	Mode 3	
SET #10,#1	75 baud	Yes
#2	110 baud	
#3	134.5 baud	
#4	150 baud	
#5	300 baud	
#6	600 baud	
#7	1200 baud	
#8	1800 baud	
#9	2000 baud	
#10	2400 baud	
#11	3600 baud	
#12	4800 baud	
#13	7200 baud	
#14	9600 baud (default)	
#15	19200 baud	
	The terminal port baud rate	
SET #11,#1	1 stop bit (default)	Yes
#2	2 stop bits	
	The number of stop bits for the terminal port	

(continued)

Parameters	Description	Reset Required
SET #12,#0	No Parity (default)	Yes
#1	Even Parity	
#2	Odd Parity	
	The parity for the terminal port	
SET #13,#n	CRT length (default: 24 lines)	No
	The maximum number of lines displayed for commands that use paging	
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	

Parameters	Description	Reset Required
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.	
SET #23,$n,$m	Transparent Mode Escape Sequence	No
	When entered from the computer port, transparent mode is exited. The default sequence is ESC, ESC ($1B,$1B).	
SET #24,$n,$m,$o	Command terminator sequence	No
	The default sequence is RETURN, null, null ($0D, $00, $00).	
SET #25,#n	Upload record length	No
	The maximum length for an upload record. (The default length is 32 bytes of data.)	
SET #26,#0	Intel (default)	No
#1	MOS	
#2	Motorola	
#3	Signetics	
#4	Tektronix	
#5	Extended Tekhex	

(continued)

Parameters	Description	Reset Required
SET #27,$n	Acknowledge character The acknowlege character is sent when a valid record is received when downloading in computer control. The default is $06.	No

Comments

Some **SET** parameters require the system to be reset, and will 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 will show the new value you have assigned the variable, even though it is not currently in effect.

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

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

See Serial Communication (page 5-29) for information on communicating with a host computer.

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

```
 >ON
     ES SWITCH SETTINGS
LD/SAV 4:  LOAD/SAVE SWITCH SETTINGS IN EEPROM
EXAMPLES:  >ON BTE + CPY + SLO
           >OFF FST + PPT

VALUE   NAME      DESCRIPTION

OFF     PPT       TRACE PEEK/POKE CYCLES
ON      BTE       BUS TIMEOUT ENABLE
OFF     FTO       FAST BUS TIMEOUT
OFF     DBP       DISABLE BUS ERRORS ON PEEKS/POKES
ON      SLO       SLOW INTERRUPT
OFF     FST       FAST INTERRUPT
OFF     CAS       CONTINUOUS ADDRESS STROBE WHILE PAUSED
OFF     TAD       TRI-STATE ADDRESS BUS WHILE PAUSED
OFF     SPD       VIEW BUS TIMING INFO INSTEAD OF IPLS
OFF     IM        INTROSPECTIVE MODE
ON      ECS       CONTINUOUS ECS/OCS WHILE PAUSED (68020 ONLY)
OFF     CDS       CACHE DISABLE (68020 ONLY)
OFF     CPY       COPY DATA TO TERMINAL & COMPUTER PORTS
ON      TCE       TRACE CAPTURE ENABLE (DYNAMIC TRACE ONLY)
 >
```

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

(continued)

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

Comments

Some ON/OFF switches cannot be set during run mode. See the switch command pages for specific information.

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

BTE	Bus timeout enable
CAS	Continuous address strobe while paused
CDS	Cache disable (68020 only)
CPY	Copy data to terminal and computer ports
DBP	Disable bus errors on peeks/pokes
ECS	Continuous ECS/OCS while paused (68020 only)
FST	Fast interrupts
FTO	Fast bus timeout
IM	Introspective mode
PPT	Trace peek/poke cycles
SLO	Slow interrupt
SPD	View bus timing info instead of IPLS
TAD	Tri-state address bus while paused
TCE	Trace capture enable (Dynamic Trace only)

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

You can save all of the current switch settings in EEPROM for later use with a **SAV** (to save all variables and settings) or **SAV 4** (to save just switch settings) command (see page 5-25).

The saved switches can be loaded automatically at power-up or manually after the system is up and running. To load automatically, set the thumbwheel switch (see page 3-5) before turning on the Emulator. To load manually, enter **LD** (to load all variables and settings) or **LD 4** (to load just the switch settings) (see page 5-27).

Examples

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

```
>ON CPY
>
```

BUS TIME OUT ENABLE

Command	Result
ON BTE	With **BTE** switch enabled, the Emulator generates a bus error and breaks emulation if a bus cycle is not terminated in a reasonable period of time (refer to page 5-20). A bus error message is displayed on the screen.
OFF BTE	When **BTE** is off, the Emulator does *not* terminate a bus cycle. Default: **OFF**

Comments

When a bus error is detected with **BTE** either ON or OFF, the entire target system stack of registers is saved. These registers are saved in the Emulator under specific names and may be examined and/or modified. (See page 5-70). **BTE** is always enabled for peeks/pokes while paused, regardless of switch setting.

CONTINUOUS ADDRESS STROBE

Command	Result
ON CAS	If **CAS** is ON, the address strobe continues to be active in the target system while paused.
OFF CAS	If **CAS** is OFF, the address strobe is active only during run mode. Default: **OFF**

Comments

Use this software switch to allow the address strobe to go to the target system while the Emulator is not in run mode.

CACHE DISABLE (68020)

Command	Result
ON CDS	When the cache disable is ON, the CDIS pin in the emulator is asserted. The microprocessor will not use its cache memory. This is necessary if the trace disassembler is going to be used.
OFF CDS	When **CDS** is OFF, the CDIS signal in the target system is used. Cache memory will operate normally with internal cache cycles replacing some of the external bus cycles during small loops. Default: **OFF**

Comments

You may want to enable the **CDS** switch if you are disassembling trace memory (see page 5-103).

COPY DATA TO BOTH PORTS

Command	Result
ON CPY	Send all data to both the terminal and computer ports. Data sent to the controlling port is echoed to the other port (non-controlling port).
OFF CPY	Only send data from the Emulator to the controlling port.
	Default: **OFF**

Comments

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

See Serial Communications, page 5-29, for more information on the terminal and computer ports.

DISABLE BUS ERRORS ON PEEKS/POKES

Command	Result
ON DBP	When **DBP** is ON, the bus error signals coming from a target system are ignored by the Emulator during peeks/pokes. However, the Emulator will still generate a bus error when one is detected by the Emulator's built in watchdog circuit.
OFF DBP	When **DBP** is off, target system bus errors are detected and will display on your console screen. Default: **OFF**

Comments

Use this software switch to disable bus errors generated by the target if using overlay memory mapped to a target area that will generate bus errors when accessed.

EXTERNAL CYCLE START (68020)

Command	Result
ON ECS	When **ECS** is ON, the Emulator sends both the external cycle start (**ECS**) and the operand cycle start (**OCS**) signals to the target system while in pause mode.
OFF ECS	When **ECS** is OFF, the Emulator does not send the external cycle start (**ECS**) or the operand cycle start (**OCS**) signal to the target system while in pause mode. Default: **ON**

Comments

This software switch setting is operable only on the 68020 microprocessor.

The **ECS** and **OCS** signals will always operate during run mode.

INTERRUPT ENABLE

Command	Result
ON SLO	When **SLO** (slow interrupt enable) is ON, interrupts will not be enabled immediately upon going into run mode. A delay of approximately 160 clock cycles will elapse before interrupts are enabled. Default: **ON**
ON FST	When **FST** (fast interrupt enable) is ON, interrupts will be enabled the moment the Emulator begins executing the target system program. Default: **OFF**

Comments

The **FST** switch will take precedence over the **SLO** switch when both are in the ON switch setting.

Switch setting matrix:

SLO	FST	RESULTS
ON	ON	Interrupts immediately enabled
ON	OFF	Interrupts delayed
OFF	ON	Interrupts immediately enabled
OFF	OFF	Interrupts generated by the target system will be inhibited from reaching the Emulator.

FAST TIME OUT

Command	Result
ON FTO	When **FTO** is on, **BTE** requires 2240 clock cycles.
OFF FTO	When **FTO** is off, **BTE** requires 35,840 clock cycles. Default: **OFF**

Comments

FTO should be set appropriately for the target system's bus error timeout.

INTROSPECTIVE MODE

Command	Result
ON IM	When **IM** is ON, the Emulator recognizes its own internal memory space as a target system. This allows the **DNL** command to download to this internal memory space just as it would a target system.
OFF IM	The Emulator recognizes the pod assembly and target microprocessor connected to its emulation board as the target system.
	Default: **OFF**

Comments

This command is valid only in pause mode.

PEEK POKE TRACE

Command	Result
ON PPT	This will cause the system while in the pause mode to trace all target reads (peeks), and writes (pokes) cycles to the target system.
OFF PPT	In this software switch position trace memory will only trace those memory cycles in the target system. Peeks and pokes, while in the pause emulation mode, are not traced. Default: **OFF**

Examples

To trace peeks and pokes to a particular memory location:

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

All peeks, pokes and memory cycles at the specified memory location will be traced by trace memory. To display what has happened at this location (5550) during the memory diagnostics, use the **DRT** command.

VIEW BUS SPEED INFORMATION

Command	Result
ON SPD	When **SPD** is ON, the **IPL** column will display a number that relates to the access time of devices on the target system bus.
OFF SPD	When **SPD** is OFF, the state of the interrupt lines will be displayed. Default: **OFF**

Comments

Use this software switch setting to view bus timing information instead of the state of the interrupt lines (IPL) from the target system.

The **DRT** command contains a column labeled **IPL** that displays the state of the interrupt lines from the target system.

If the **SPD** switch is ON, this column is labeled **SPD**.

If one access displays a 4 and another a 5, the later bus cycle took one more clock cycle to complete than the former. Access times greater than or equal to 10 cycles display as $\boxed{+}$.

The **IPL/SPD** column is not displayed when in trace mode 2 (68020 only).

TRI-STATE ADDRESS BUS

Command	Result
ON TAD	When **TAD** is ON, the Emulator address bus is tri-stated while paused. This will tri-state the address bus any time the Emulator is not emulating and doing peeks and pokes.
OFF TAD	When **TAD** is OFF, addresses generated during pause mode are output by the Emulator address bus to the target system. Default: **OFF**

DYNAMIC TRACE CAPTURE ENABLE

Command	Result
ON TCE	This command starts trace acquisition. With TCE on, the DT, DTB, DTF, and DRT commands work only in pause mode.
OFF TCE	This command stops trace acquisition to allow examination of your trace memory. With TCE off, you can observe trace without stopping emulation.

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. Type OFF TCE to read the trace (after the buffer is frozen). Type ON TCE to restart trace acquisition.

(continued)

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. Also, the Event Monitor System counters will not increment.

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.

SAVE SYSTEM VARIABLES IN EEPROM

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

Comments

This command is valid only in pause mode.

A **SAV** operation may take up to two minutes.

DO NOT INTERRUPT THE PROCESS!

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

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

(continued)

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

0 - SET menu	5-3
1 - Contents of Emulator registers	5-71
2 - Event Monitor System WHEN/THEN Statements	7-1, 8-1
3 - Overlay map	5-55
4 - Software switch settings	5-9
5 - Macros	5-117

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

When you first receive your Emulator, you should execute a **SAV** command with no parameter. This initializes EEPROM, so that subsequent **LD** commands will work properly with the 68000 series Emulator board and pod.

Examples

```
>SAV 1
```

The current values of all the Emulator registers are saved in EEPROM.

LOAD SYSTEM VARIABLES FROM EEPROM

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

Comments

This command valid only in pause mode.

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

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

You may load the following variable categories from EEPROM:

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

(continued)

Examples

```
   >LD 3
```

The overlay memory map in the EEPROM is copied into internal RAM. Use the **DM** command to verify the new map.

Serial Communications

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

USING A HOST COMPUTER

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

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

DATA BUFFERING AND BAUD RATE

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

XON and XOFF characters can be used to control either output port on the ES1800. These characters are user definable. The problem described in the above paragraph can happen in the reverse direction. If the ES1800 is uploading data to the host, it may be able to overrun the host's ability to receive characters. While lowering baud rates may help, there are probably commands available on your host to solve the problem. You should also make sure that the host does not echo characters sent to it while uploading data. If the characters are echoed, the ES1800 will quickly send an XOFF to the host while continuing to send normal upload characters. The host

(continued)

system will then probably send an XOFF to the ES1800 because the host's buffers are full. The result of this situation is that both systems will lock up waiting for the other to send an XON. See your system administrator or call Applied Microsystems Corporation Customer Service at 1-800-426-3925 for help.

COMMUNICATION WITH THE HOST COMPUTER

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

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

PORT DEPENDENT COMMANDS

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

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

DOWNLOAD FROM TERMINAL PORT

When the ES1800 receives a download command (**DNL**), it always expects data records to arrive at the computer port. If the download command is entered from the terminal port, the ES1800 automatically enters transparent mode to allow you to send commands to your host system. You normally enter a command that causes your host system to copy the formatted object file to your terminal (see page 5-7 for object file formats). The proper procedure is to enter the command to your host system but not terminate it (i.e., do not press the RETURN key). Instead, enter the two-character transparent mode escape sequence. When transparent mode terminates, control returns to the download process. The download routine then sends the user definable command terminator sequence to your host system (see page 5-3). Your host system responds by sending the data records from your 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 are written into target memory. When the End of File (EOF) record is received, the download process terminates and a normal ESL prompt is displayed.

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

DOWNLOAD FROM COMPUTER PORT

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

TRANSPARENT MODE

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

Comments

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

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

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

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

Refer also to Serial Communications (page 5-29).

Examples

```
>TRA
```

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

TERMINAL PORT CONTROL

Command	Result
TCT	The terminal port becomes the controlling port.

Comments

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

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

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

For port selection on power-up refer to page 3-5.

COMPUTER PORT CONTROL

Command	Result
CCT	The computer port becomes the controlling port.

Comments

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

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

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

If there is a host attached to the computer port and you type a CCT from a terminal connected to the terminal port, the host system takes control of the Emulator. The host system must be able to handle incoming data at high rates. Both hardware and software handshakes are supported (see page 3-14).

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

(continued)

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.

  In most cases this process returns control to the terminal. If not:

- Turn the Emulator off and then on.

This command can be executed from the computer port. For port selection on power-up refer to page 3-5.

DOWNLOAD OPERATIONS

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

Comments

You can choose the destination of the downloaded file:

- Target memory
- Overlay memory

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

Verify also that the data format of the host system file matches that being used by the Emulator. Refer to **SET** menu set parameter #26 for verification of Emulator format. Use transparent mode (**TRA**) to verify host system format and the address in the file. (See page 5-32.)

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.

(continued)

DOWNLOADING UNDER TERMINAL PORT CONTROL

After typing **DNL**, the system automatically enters transparent mode, allowing communication with the host system. When you are ready to download the file, enter a command that causes the host system to display a file to the terminal, but *in place of a* RETURN, enter the transparent mode escape sequence (see page 5-32).

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

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

RETURN CONTROL TO EMULATOR

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

1. An end of file record is received. If an end of file record is not recognized by the Emulator, control will *not* be returned to the Emulator terminal port. This can be caused by:
   - Using a RETURN instead of the proper escape sequence to terminate the command line to the host computer.
   - Selecting the incorrect data format.
2. An Emulator reset is executed (factory default is CTRL Z).
3. An error is detected.

DOWNLOADING UNDER COMPUTER PORT CONTROL

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

```
>DNL
```

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

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

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

- All good records are acknowledged with an $\boxed{\texttt{ACK \$06}}$.
- All error messages from bad records are received on the computer port; therefore the host program that is controlling the Emulator will need to be able to interpret error messages.
- Records are not echoed.

SYMBOLIC DOWNLOAD

The download command accepts symbolic definition records as well as data records when the symbolic debug option is used and the Emulator download format variable is set to 5 (Extended Tekhex). (See **SET** parameter #26, page 5-7.)

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

(continued)

Errors

CHECKSUM ERROR IN THE DATA RECORD

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

READ AFTER WRITE VERIFY ERROR

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

Problem	What to Check
Emulator does not return a prompt after file has been sent.	1. Serial data format - **SET** menu. 2. No end of file (EOF) record. 3. You entered a $\boxed{\text{RETURN}}$ instead of the transparent mode escape sequence after entering the host copy command.
Read-after-write verify error.	1. Target hardware problem. 2. Overlay memory not mapped in download range. Address is indicated by misverify message.

Checksum error.	1. Improperly formatted record sent by host. 2. Noisy serial data lines. 3. Host computer is not responding to XON/XOFF protocol.
Display of data does not commence after entering transparent mode escape sequence.	1. Host not responding to user defined command terminator sequence - **SET** menu.

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

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

VERIFY SERIAL DATA

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

Comments

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

Use this command if you suspect a file you downloaded was corrupted. If downloaded data is being corrupted by your program, you can detect it by mapping overlay as **RO** (read only) (see page 5-55).

This command is also useful for determining differences between object files. Follow instructions for downloading a file on page 5-37.

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

Comments

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

If uploading to a file on a host system, transparent mode should be entered first in order to open a file to store the uploaded data records. (Refer to your host system instructions on ASCII files.)

Example

UNIX	**CAT** >Filename
VMS	**TYPE** >Filename

(Create or EDT are also acceptable)

CPM	**PIP** A:Filename = RDR:

After this is done, enter the transparent mode escape sequence, and the upload command.

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

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

(continued)

If your 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 a host. In this case, there is no need to enter transparent mode before uploading. Just be sure the peripheral is ready to receive data.

Refer also to Serial Communications, page 5-29.

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

Comments

Extended Tekhex restricts the number and range of characters that can be used for symbol names. When formating symbols for upload, the Emulator truncates symbol names to 16 characters and substitutes $\boxed{\%}$ for characters not allowed by Tekhex.

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

If uploading to a file on a host system, transparent mode should be entered first in order to open a file to store the uploaded data records. (Refer to your host system instructions on ASCII files.)

Example

UNIX	**CAT** >Filename
VMS	**TYPE** >Filename

(Create or EDT are also acceptable)

CPM	**PIP** A:Filename = RDR:

After this is done, key in the transparent escape sequence, and begin the uploading.

(continued)

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

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

Refer also to Serial Communications, page 5-29, and Symbols, page 5-126.

COMMUNICATION WITH TARGET PROGRAMS

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

Comments

This command is only useful during run mode. It affects real time operation and requires special target code. **COM** mode uses two bytes at the specified address. The byte at *<address>* is used for characters sent from the target to the controlling port. The byte at *<address>* + 1 is used for characters being sent to the target program. This command makes use of 7 bit ASCII characters, with the eighth bit of each byte used for handshaking.

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

To receive a character from the Emulator, the target examines the byte at *<address>* + 1. If the MSB of this byte is cleared, the Emulator has not yet transmitted a new character. If the MSB is set, the character is new. If the controlling port of the ES1800 is a terminal, the target program should

(continued)

echo the character back by immediately copying the character 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.

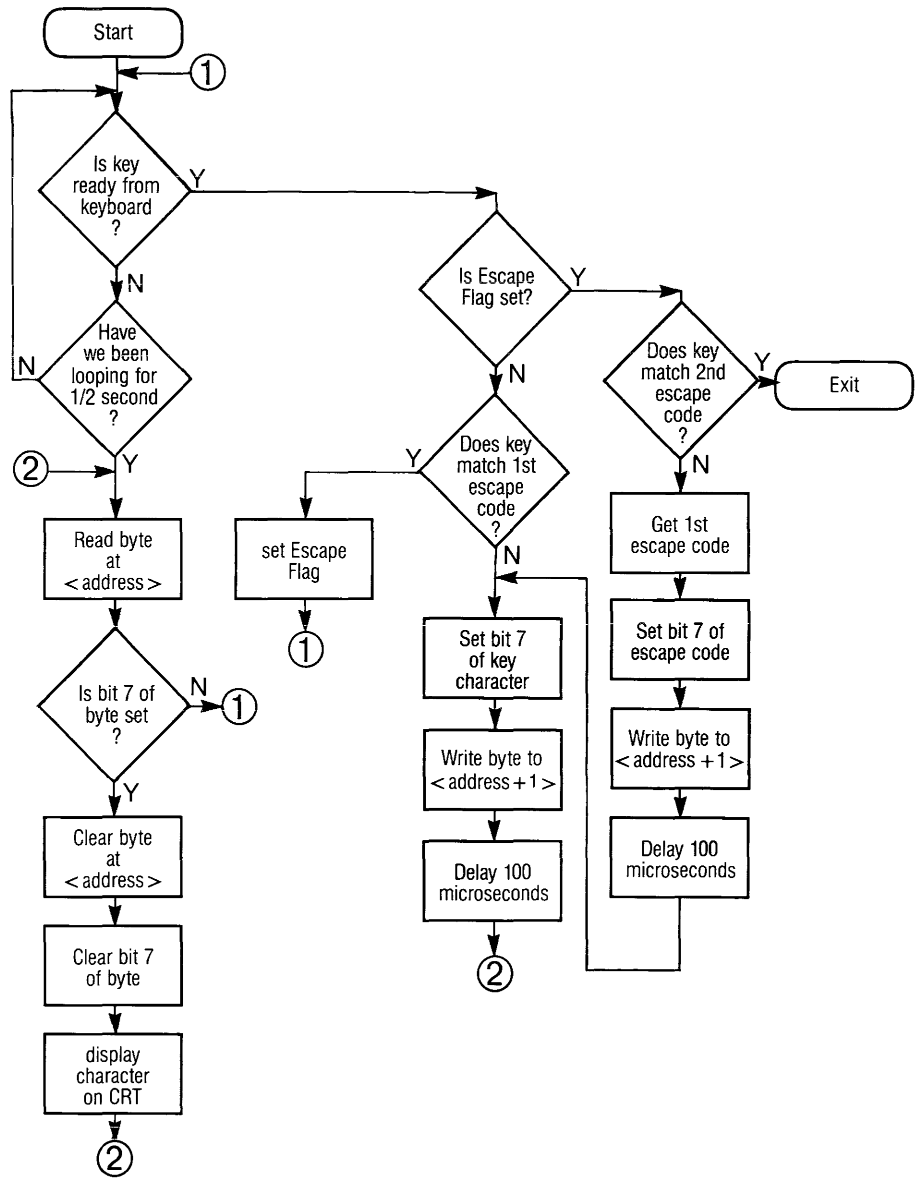
The **COM** routine does not check the byte at *<address>* + 1 to see if the target program has received it. Generally, the target program will be substantially faster than the **COM** routine and will always receive one character before the **COM** routine can transmit the next.

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

The following flow diagram summarizes the **COM** process.

Figure 6. Flow Chart

COM Routine Processing


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

(continued)

Examples

One good example of using the **COM** command is to simulate a serial I/O port when debugging code before target hardware is available. The target program is downloaded into overlay memory and emulation is entered by using the **RUN** command. 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 terminal.

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

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

DISPLAY CHARACTER STRING

Command	Result
DIA *<address>*	Read and display characters from target memory starting at the specified address. The DIA routine terminates when it reads $00 from target memory. Affects real time operation when entered in run mode. See page 6-1.

Comments

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

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

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

(continued)

Examples

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

This example sets a breakpoint in your target error routine. When the breakpoint occurs, a message pointed at by the D0 register is displayed. If the D1 register is zero, the process stops. Otherwise, the ES1800 immediately begins emulation and waits for another breakpoint and message.

```
  >AC1 = 'Error_stop
  >WHE AC1 THE BRK
  >* RBK;WAI;DIA D0;TST = D1
```

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 your program is making illegal accesses, and patch target PROM code quickly and easily.

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

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

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

Overlay memory accesses occur in real time, no wait states are added by the Emulator.

DISPLAY MEMORY MAP

Command	Result
DM	Display the memory map currently in effect.

Comments

This command is valid only in pause mode.

Examples

Default map at power up (68000/08/10):

```
 >DM
MEMORY MAP:
OVERLAY ENABLED FOR (OVE=) UD+UP+SD+SP, SPEED (OVS)=0
MAP $000000 TO $FFFFFF:TGT
 >
```

Default map at power up (68020):

```
 >DM
MEMORY MAP:
OVERLAY ENABLED FOR (OVE=) UD+UP+SD+SP, SPEED (OVS)=1
MAP $00000000 TO $FFFFFFFF:TGT
 >
```

Command	Result
MAP *<range>*	The specified range is mapped and assigned the default attribute type, RW.
MAP *<value>*	A 2K byte block is mapped surrounding the specified value. The block is assigned the default attribute type, RW.
MAP *<range><attribute>*	The specified range is mapped and assigned the specified attribute type.
MAP *<value><attribute>*	A 2K byte block is mapped surrounding the specified value. The block is assigned the specified attribute.

Attributes

RW — Memory mapped with this attribute will respond like normal overlay memory. 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 will define RW partitions.

(continued)

RO	Memory mapped as RO acts like read only memory to your target program. If your program attempts to write to this memory, the ES1800 will abort run mode and display MEMORY WRITE VIOLATION. The contents of RO overlay cannot be altered by a running target program. You can always modify memory mapped as RO (in pause mode) even though your target program (run mode) cannot. The same comments about speed given in the paragraph on RW, apply to memory mapped as RO.
ILG	Memory mapped as illegal can be used to mark address ranges that should not be accessed by your target program. Any access to an address range mapped as ILG will cause the ES1800 to abort run mode and display MEMORY ACCESS VIOLATION. Memory mapped as ILG does not use up available overlay memory.
TGT	Accesses in address ranges mapped with this attribute are ignored by the ES1800. Memory that is not explicitly mapped is defaulted to TGT.

Comments

Overlay memory is mapped in segments of 2K bytes. If you specify an address or a range to be mapped as RW or RO, the mapping outline will allocate the minimum number of 2K segments that will completely enclosed the address(es) of interest (see overlay memory page 5-63).

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

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

MAPPING THE 68020

Two issues need to be considered when using overlay memory with the 68020 microprocessor. The first issue concerns addressing. The second issue deals with wait states.

ADDRESSING

The overlay memory in the ES1800 Emulator is limited to 24 address bits even though the 68020 microprocessor uses 32 address bits.

This limits the acceptable mapable range to 16 megabytes. Overlay memory may be mapped anywhere in the four gigabyte addressable area of the 68020 in 2K increments, provided the mapped area is within a 16 megabyte window. If an attempt is made to map overlay memory outside of the 16 megabyte window (called a segment) the following error will appear:

```
ATTEMPT TO CHANGE CURRENT OVERLAY SEGMENT (USE CLM FIRST)
```

The **CLM** (clear memory map) command must be issued prior to mapping outside the previously specified address range. (See page 5-62).

When mapping memory for the 68020 the most significant address byte must be the same for every 2K byte segment mapped (except for memory mapped TGT).

```
MAP $1F000000 TO $1F0007FF :RW
MAP $1F100000 TO $1F101FFF :ILG
MAP $1F200000 TO $1F2007FF :RO
```

(continued)

WAIT STATES:

The minimum amount of wait states for the overlay memory is 1. The OVS (overlay speed) will automatically default to 1. However, if the target systems clock speed is 8 megahertz, 0 wait states will be utilized. If a slower speed is required to match slower hardware in the target system, type in "**OVS=X**" where X is 1 through 7. If the OVS value is set to any other value, the following error message will occur:

```
ILLEGAL OVS VALUE...(NOT IN 1-7)
```

The reason wait states may be inserted is because the 68020 uses a 25 percent shorter bus cycle.

The overlay memory is treated by the 68020 as a 16 bit port. This could increase execution time if overlaying a 32 bit bus.

The addresses in the Emulator for the overlay memory are treated as segmented addresses. This is due to the internal 24 bit bus not being large enough for the 68020's architecture. Therefore, 8 extra bits are being stored on the pod and appear to the Emulator as a segment for internal purposes. However, all accesses to the target area are 32 bit. When accessing target memory, typically, the Emulator will not insert any wait states. If the target is running over 8 megahertz, and the target area that is being accessed is in the same segment as an area of overlay memory that is mapped, one wait state may be inserted.

Examples

If 0 to 7FF is mapped as RW, and an access to any area of target memory that is in the bottom 16 megabyte range (800 to FFFFFF) occurs, a wait state may be inserted.

Again, the reason we may insert a wait state is due to the 68020 bus cycles being executed considerably faster than the 68010. Due to the segmented logic being located in the pod, the Emulator will know relatively quickly that an overlay memory access is going to be made. All we need to complete the cycle is to have the ˜OME signal come back. This indicates the address has been set up and the memory is ready. Then, DSACK will be required to terminate the cycle. If ˜OME is received in time, a wait state will not be inserted. If the access is in the target area, we will not rely on the ˜OME signal unless there is a "match" in the segment logic early in the bus cycle.

The segmented address only affects the overlay memory operation.

The following command sequence might reflect a common 68000/08/10 mapping.

```
Command                            Comments
---------------------------        -------------------------------------
 >CLM                              Clear map to all :TGT
 >MAP 'ram-start LEN 2000          Map some RAM to work with
 >MAP 'rom-start LEN 4000:RO       Map ROM
 >MAP $4000 to $DFFF:ILG           No accesses expected in this range
 >DM                               Display map

 MEMORY MAP:
 OVERLAY ENABLED FOR (OVE=) UD+UP+SD+SP, SPEED (OVS)=0
 MAP $000000 TO $003FFF :RW
 MAP $004000 TO $00DFFF :ILG
 MAP $00E000 TO $00FFFF :RW
 MAP $010000 TO $FFFFFF :TGT
```

(continued)

The following command sequence might reflect a common 68020 mapping.

```
Command                               Comments
----------------------------          -----------------------------------
 >CLM                                 Clear map to all :TGT
 >MAP 'ram-start LEN 2000             Map some RAM to work with
 >MAP 'rom-start LEN 4000:RO          Map ROM
 >MAP $4000 to $DFFF:ILG              No accesses expected in this range
 >DM                                  Display map

 MEMORY MAP:
 OVERLAY ENABLED FOR (OVE=) UD+UP+SD+SP, SPEED (OVS)=1
 MAP $00000000 TO $00003FFF :RW
 MAP $00004000 TO $0000DFFF :ILG
 MAP $0000E000 TO $0000FFFF :RW
 MAP $00010000 TO $FFFFFFFF :TGT
```

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

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

If you don't have target memory to work with, and you still want to compare two programs, you can use a trick of overlay memory allocation. This particular example assumes you have 128K or more of overlay memory.

```
Command                            Comments
----------------------------       -------------------------------------
 >CLM                              Clear previous map
 >GR0 = 1000 LEN 8000              Will save some typing
 >MAP GR0                          Map 32K bytes for code space
 >DNL                              Download first program into overlay
  (Download commands and records)
 >MAP GR0:TGT                      Unmap code space (The data is still
                                   in Overlay RAM)
 >MAP GR0 + 10000                  Remap but at higher address range.
                                   The first program now "exists" again
                                   but in a higher address range.
 >MAP GR0                          Now map more overlay at the normal
                                   range
 >DNL                              Download second program.
  (Download commands and records)
                                   Now you have a copy of both programs.
 >MAP GR0:TGT;MAP GR0 + 20000      Relocates second program out of the way
 >MAP GR0 +10000:TGT;MAP GR0       Relocates first program back to normal
                                   address range.
```

CLEAR MEMORY MAP

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

Comments

This command clears all addresses from the overlay map.

This command is valid only in pause mode.

OVERLAY MEMORY ENABLE

Command	Result
OVE = SP + SD	Both supervisor program and supervisor data space are decoded by the overlay memory.
OVE = SP	Only supervisor program status space accesses are decoded by overlay memory.
OVE = SD	Only supervisor data status space accesses are decoded by overlay memory.
OVE = ALL	Enables overlay for all spaces. (68000/08: spaces 1, 2, 5, 6, 7) (68010/20: spaces 0-7)
OVE = SC*<0,3,4>*	Enables overlay memory for a specific space (0, 3, 4) as additional memory space (68010/20).

Comments

This command allows you to utilize overlay memory as SP, SD, UP, UD, CPU and ALL.

(continued)

Status Space	Mnemonic	Description
0		Unnamed (68010/20)
1	UD	User data
2	UP	User program
3		Unnamed (68010/20)
4		Unnamed (68010/20)
5	SD	Supervisor data
6	SP	Supervisor program
7	CPU	Target microprocessor space

Overlay memory will respond to an access only if a mapped address and the current OVE status match the cycle being executed. For more information about the eight status spaces, see the Raw Trace section and the *16 Bit Microprocessor Users Manual* or the *MC68020 32 Bit Microprocessor Users Manual.*

SP is code space. The processor encodes it as supervisor program status.

SD is data space. The processor encodes it as supervisor data.

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

To display the value of the current status being used for memory mode access, use the MMS command (page 5-81).

LOAD OVERLAY MEMORY

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

Comments

This command valid only in pause mode.

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

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

Use the **VFO** command (page 5-66) to verify the memory move.

VERIFY OVERLAY MEMORY

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

Comments

This command is valid only in pause mode.

OVERLAY MEMORY SPEED

Command	Result
OVS = *<0-7> (<1-7> 68020)*	Overlay memory will either supply a DTACK or not, according to the specified parameter.

Comments

Use this register to determine whether overlay memory will return a DTACK to the CPU.

DISPLAY

The current value of OVS is displayed when the memory map is displayed.

OVS may be loaded with one of seven parameters:

```
0 = No DTACK supplied by overlay memory
    (this does not apply to the 68020)
1 = No delay, address strobe returned to 680XX as DTACK
2 = +1 cycle delay
3 = +2 cycles delay
4 = +3 cycles delay
5 = +4 cycles delay
6 = +5 cycles delay
7 = +6 cycles delay
```

When overlaying PROM, it may be necessary to set MMS = <space code> + OVO. This will allow the overlay to operate independently of the system. A DTACK is automatically supplied when this is in effect.

(continued)

Registers

The following is a complete list of all the registers in the Emulator. These registers can be logically divided into four groups. Please note that there are separate lists of registers for the 68000/08, 68010, and 68020 microprocessors.

- microprocessor registers
- general Emulator registers
- event system registers
- bus error registers

Each register accepts one or two of three value types:

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

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

68000/08/10/20 Target Microprocessor Registers

Name	Description	Type	Length (bits)
PC	program counter register	Int	24
USP	user stack pointer register	Int	24
SSP	supervisor stack pointer register	Int	24
SR	status register	Int	10
A0	address register #0	Int	32,16
A1	address register #1	Int	32,16
A2	address register #2	Int	32,16
A3	address register #3	Int	32,16
A4	address register #4	Int	32,16
A5	address register #5	Int	32,16
A6	address register #6	Int	32,16
D0	data register #0	Int	32,16,8
D1	data register #1	Int	32,16,8
D2	data register #2	Int	32,16,8
D3	data register #3	Int	32,16,8
D4	data register #4	Int	32,16,8
D5	data register #5	Int	32,16,8
D6	data register #6	Int	32,16,8
D7	data register #7	Int	32,16,8

Additional 68010 Target Microprocessor Registers

Name	Description	Type	Length (bits)
VBR	vector base register	Int	24
SFC	source function code register	Int	3
DFC	destination function code register	Int	3

(continued)

Additional 68020 Target Microprocessor Registers

Name	Description	Type	Length (bits)
CACR	cache control	Int	32
CAAR	cache address	Int	32
MSP	master stack pointer	Int	32
ISP	interrupt stack pointer	Int	32
PC	program counter	Int	32
VBR	vector base register	Int	32
USP	user stack pointer	Int	32

General Emulator Registers

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

Event Monitor System Registers

Name	Description	Type	Length (bits)
AC1.1-AC1.4	Address Comparator	Range	24
AC2.1-AC2.4	Address Comparator	Range	24
DC1.1-DC1.4	Data Comparator	DC	16
DC2.1-DC2.4	Data Comparator	DC	16
S1.1-S1.4	Status Comparator	DC	16
S2.1-S2.4	Status Comparator	DC	16
CL.1-CL.4	Count Limit Comparator	Int	16
LSA.1-LSA.4	Logic State Comparator	DC	16
SIA	Special Interrupt Address	Int	32

Each register has a separate display base. The display base is viewed and changed with the **BAS** command (see page 5-79). Display bases are often changed for registers such as the Event Monitor LSA comparators, which you might like to see in binary, and the CL register, which you might want to see in decimal.

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

See *Event Monitor System* (Chapter 7 for the 68000/08/10 or Chapter 8 for the 68020) for Event Monitor System register descriptions.

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

(continued)

68000/08/10 BUS ERROR REGISTERS

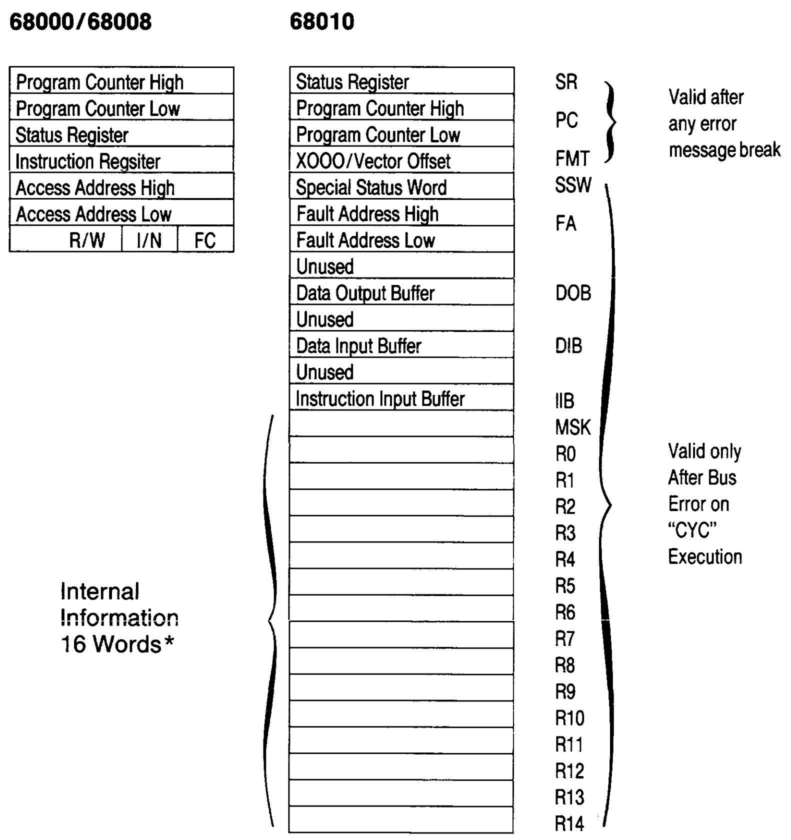
The Emulator can send a bus error to the MPU for two different reasons:

- When executing the **CYC** command, a bus error is forced so the long stack of information will be available to the firmware that simulates bus cycles.
- When the **BTE** switch is ON, emulation will be aborted by an internally generated bus error if the target system holds the address strobe asserted for a sufficient length of time.

In either case, the entire stack of registers is saved in registers with specific names. The registers may be examined and/or modified.

The registers named MSK and R0 -R14 contain internal information that is not documented by Motorola. Modification of any of these registers may result in unpredictable operation of the 68010 MPU.

Figure 7. 68000/08/10 Bus Error Registers



* These registers contain internal information that is not documented by Motorola. Modification of these registers may result in unpredictable operation of the 68010/68020 MPU.

(continued)

68020 BUS ERROR REGISTERS

The ES1800 uses the same bus error stack format internally as the 68020 chip does externally in the target sytem when the **BTE** switch is ON and an internal bus error occurs.

This information is available to the user by keying in the register name. Please refer to the tables that follow.

These registers are internal representations of the conditions listed above and do not reflect normal bus error information in the target system.

The first three registers in both the short and long bus cycle are always available after each break in emulation. The remainder are only available when the three conditions occur:

- the **BTE** switch is ON
- an internal bus error occurs
- register name is keyed in

Figure 8. 68020 Bus Error Registers

68020
Short Bus Cycle
Fault Stack Frame

Status Register	SR
Program Counter High	PC
Program Counter Low	
FMT/Vector Offset	BFMT
Internal Register 1	IR1
Special Status Word	SSW
Instruction Pipe C	IPS
Instruction Pipe B	
Fault Address High	FA
Fault Address Low	
Internal Register 2 High	IR2
Internal Register 2 Low	
Data Output Buffer	DOB
Data Output Buffer	
Internal Register 3 High	IR3
Internal Register 3 Low	

68020
Long Bus Cycle
Fault Stack Frame

Status Register	SR
Program Counter High	PC
Program Counter Low	
FMT/Vector Offset	BFMT
Internal Register 1	IR1
Special Status Word	SSW
Instruction Pipe C	IPS
Instruction Pipe B	
Fault Address High	FA
Fault Address Low	
Internal Register 2 High	IR2
Internal Register 2 Low	
Data Output Buffer	DOB
Data Output Buffer	
Internal Register 3 High	IR3
Internal Register 3 Low	
Internal Register 4 High	IR4
Internal Register 4 Low	
Stage B Address High	SBA
State B Address Low	
Internal Register 5 High	IR5
Internal Register 5 Low	
Data Input Buffer	DIB
Data Input Buffer	
Internal Register 6H	IR6
Internal Register 6L	
Internal Register 7H	IR7
Internal Register 7L	
Internal Register 8H	IR8
Internal Register 8L	
Internal Register 9H	IR9
Internal Register 9L	
Internal Register 10H	IRA
Internal Register 10L	
Internal Register 11H	IRB
Internal Register 11L	
Internal Register 12H	IRC
Internal Register 12L	
Internal Register 13H	IRD
Internal Register 13L	
Internal Register 14H	IRE
Internal Register 14L	
Internal Register 15H	IRF
Internal Register 15L	
Internal Register 16H	IRG
Internal Register 16L	

DISPLAY/LOAD MICROPROCESSOR REGISTERS

Command	Result
DR	**DR** causes the system to display the contents of all the microprocessor registers.
CLR	The **CLR** command clears registers A0-A6 and D0-D7 to zero.
LDV	Load the reset vectors into the stack pointer and program counter. The reset vectors can also be loaded by the **RNV** and **RBV** commands. These load the vectors and enter run mode (page 6-5).

Comments

This command is valid only in pause mode.

If keyed in while running, the values will be as they were before entering run emulation. They will not accurately reflect current values.

Refer to the Motorola *16 Bit Microprocessors Users Manual* or the *MC68020 Microprocessors Users Manual* for the default power-up values of the microprocessor registers. Register values may be saved to and loaded from EEPROM.

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

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

The display of the SR (and CCR in the 68020) register is different from that of the other CPU registers. The flags are more conveniently decoded by using an alpha character to indicate whether the flag was set or cleared by a particular instruction cycle. If the flag is clear, you see a ⊡ as a place holder. If set, the following characters describe the flag.

X	Trace on execution (SR,68020)	O	Overflow
M,I	Master/Interrupt state		
0-7	Interrupt priority mask		
T	Trace mode		
S,U	Supervisor/User state		
Z	Zero		
N	Negative		
X	Extend (CCR)		
C	Carry		

(continued)

Examples

```
(68000/08 microprocessor)

     - 0 -    - 1 -    - 2 -    - 3 -    - 4 -    - 5 -    - 6 -    - 7 -
D = 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A = 00000000 00000000 00000000 00000000 00000000 00000000 00000000

    PC = 000000       SSP = 000000      USP = 0000000     SR = TS7XNZVC
```

```
(68010 microprocessor)

     - 0 -    - 1 -    - 2 -    - 3 -    - 4 -    - 5 -    - 6 -    - 7 -
D = 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A = 00000000 00000000 00000000 00000000 00000000 00000000 00000000

PC=000000 VBR=000000 SFC=0 DFC=0 SSP=000000 USP=000000 SR=TS7XNZVC
```

```
(68020 microprocessor)

     - 0 -    - 1 -    - 2 -    - 3 -    - 4 -    - 5 -    - 6 -    - 7 -
D = 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A = 00000000 00000000 00000000 00000000 00000000 00000000 00000000

    SR  = XSI7                          SFC = 0             MSP = 00000000
    CCR = XNZVC        CAAR = 00000000  DFC = 0             ISP = 00000000
    PC  = 00000000     CACR = 00000000  VBR = 00000000      USP = 00000000
```

SET/DISPLAY REGISTER DEFAULT BASE

Command	Result

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 Emulator operates correctly, but the results may be confusing. If you set the base value to a value other than hexadecimal, decimal, octal,

(continued)

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

Refer to the default base command, **DFB** (page 5-85), for displaying the system global default base.

Examples

```
>BAS SR
DEFAULT: #16
>BAS SR=#8
>BAS SR
#8
```

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

The value of GD3 will always be displayed in binary until you change its display base or the Emulator is powered down.

MEMORY MODE STATUS REGISTER

Command	Result
MMS	Display the value of the current status being used for memory accesses (peeks and pokes).
MMS = *<MMS memory space>*	Set the status space for memory accesses.
MMS = *<MMS memory space>* + TGO	Map overlay memory and examine target (with no impact on mapped memory).
MMS = *<MMS memory space>* + OVO	Map target system and examine overlay memory. OVO parameter results in no bus cycles entering the target system.
MMS = *<MMS memory space>* + NRM	Returns system to default mode after TGO or OVO has been specified.
MMD	MMD is used like MMS, but for block verifies and block moves.

(continued)

Comments

One of eight memory spaces can be selected to operate memory with:

SC0 =	unused	(used only by 68010 and 68020)
SC1 =	UD	(user data)
SC2 =	UP	(user program)
SC3 =	unused	(used only by 68010 and 68020)
SC4 =	unused	(used only by 68010 and 68020)
SC5 =	SD	(supervisor data)
SC6 =	SP	(supervisor program)
SC7 =	CPU	(interrupt acknowledge or CPU space cycle)

MEMORY MODE POINTER

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

Comments

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

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

The memory mode pointer is automatically modified when you scroll to a new address after entering memory mode. When you exit memory mode, the MMP reflects the last address examined. For more information on memory mode, see page 6-15.

(continued)

Examples

Set 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
```

DEFAULT BASE

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

Comments

Various operators tell the Emulator what base an input value is in. They are:

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

(continued)

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

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

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

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

GENERAL PURPOSE DATA REGISTERS

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

Comments

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

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

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

Examples

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

```
>GD4 = 5000
```

(continued)

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

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

You may choose to use a general purpose register instead of memory mode status mnemonics.

```
>GD6 = SP
>MMS = GD6
>GD1 = OVL+RD+IOA                    To set-up a breakpoint on an overlay
>S1 = GD1                            read.
```

GENERAL PURPOSE ADDRESS REGISTERS

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

Comments

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

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

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

Examples

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

```
>GR4 = 5000
```

(continued)

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

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

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

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

TEST REGISTER

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

Comments

See The Repeat Operator for more detailed information. (See page 5-122).

(continued)

Trace Memory

Trace memory commands deal with the display and disassembly of trace memory data. Refer to the Event Monitor System (Section 7 for 68000/08/10 or Section 8 for 68020) for sophisticated uses of trace memory.

Trace memory is 70 bits wide and 2046 bus cycles deep (101 bits wide for 68020). You may use some bus cycles as marks to identify start and stop points within the trace buffer. An unqualified trace contains all bus activity for the last 2046 bus cycles. Trace can be searched for a particular address, data or status.

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 happened. When used in conjunction with the trace disassembler, hardware and software problems may be found.

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 system is running. You can trace in target systems which require one or more signal lines active, such as targets using dynamic RAM. With targets using multiple multiprocessors, dynamic trace lets you examine trace from one processor without shutting down all the processors.

PERFORMANCE ANALYSIS (68020)

A timer provides a time signature for each bus cycle captured in trace memory. Each line in trace memory is "stamped" with a time relative to when emulation started.

Emulation

TRACE MEMORY MODES (68020)

The 68020 microprocessor is a 32 bit environment, while the trace and Event Monitor Systems are based on a 16 bit environment.The additional signals available with the 68020 microprocessor make it impossible to trace all 32 address, 32 data, and relevant status bits simultaneously.

To enhance compatibility of these two environments, the ES1800 has four trace memory modes. These four modes trace four combinations of address, data, and status bits giving you full coverage of the bits available with the 68020 microprocessor.

The four trace memory modes and their respective bits combination are:

Mode	Address	Data	Status	Timing	LSA
0	24	16	19	24	16
1	24	32	19	24	0
2	32	32	11	24	0
3	32	16	21	24	6

MODE SELECTION

To determine which mode you would like to run in, first become familiar with what is presently in program memory. Decide whether your program requires the tracing of more address bits than data bits, or more data bits than address bits. Select the mode that will be most effective for the program that is going to be traced.

When using the 68020 microprocessor it is required that prior to entering the run emulation mode you must designate which of the four trace memory modes you are going to use. Set parameter #4 on the SET menu (see page 5-5) is used to designate the mode.

To look at raw trace you need to know what bits are traced in each mode. For run mode to operate correctly with trace memory modes the comparator registers are required to be set up appropriately. (Refer to page 8-14 for more detailed information).

DISPLAY RAW TRACE BUS CYCLES

Command	Result
DRT	Display the last page of bus cycles recorded in trace memory.
DRT *<line number>*	Displays a page of the trace buffer starting with *<line number>*.
DRT *<range>*	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 page 5-3).

This command is valid only in pause mode, unless you have the dynamic trace feature.

See DST (68020), page 5-108 for an alternative way to display the raw trace bus cycles.

NOTE: The sequence numbers in **DT**, **DTB**, and **DTF** (instructions) correlate with the line numbers displayed in **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 a multiple bus cycle instruction has been executed. Also, the sequence number (SEQ #) may be repeated when two byte wide instructions were executed from continuous addresses.

Examples

```
                              68000/08/10

>DRT
LINE  ADDRESS   DATA  R/W        FC     IPL LSA  -  8  7  -  0
 #20  00100A  > 3080   R   OVL   SP      0  %11111111 %11111111
 #19  00100C  > B03C   R   OVL   SP      0  %11111111 %11111111
 #18  00100E  > 0039   R   OVL   SP      0  %11111111 %11111111
 #17  001010  > 6C02   R   OVL   SP      0  %11111111 %11111111
 #16  001012  > 5E40   R   OVL   SP      0  %11111111 %11111111
 #15  001014  > 5740   R   OVL   SP      0  %11111111 %11111111
 #14  001016  > 41FA   R   OVL   SP      0  %11111111 %11111111
 #13  001018  > 03F0   R   OVL   SP      0  %11111111 %11111111
 #12  00101A  > 3140   R   OVL   SP      0  %11111111 %11111111
 #11  00101C  > FF00   R   OVL   SP      0  %11111111 %11111111
 #10  00101E  > 1140   R   OVL   SP      0  %11111111 %11111111
  #9  001308  > 0037   R   OVL   SP      0  %11111111 %11111111
  #8  001020  > FFF7   R   OVL   SP      0  %11111111 %11111111
  #7  001022  > 60F0   R   OVL   SP      0  %11111111 %11111111
  #6  0013FF  >   37   W   OVL   SP      0  %11111111 %11111111
  #5  001024  > 4E71   R   OVL   SP      0  %11111111 %11111111
  #4  001004  > B03C   R   OVL   SP      0  %11111111 %11111111
  #3  001006  > 0039   R   OVL   SP      0  %11111111 %11111111
  #2  001008  > 6C02   R   OVL   SP      0  %11111111 %11111111
  #1  00100A  > 3080   R   OVL   SP      0  %11111111 %11111111
  #0  BREAK
```

(continued)

```
                                   68020
                               MODE 0 DISPLAY

>DRT
LINE  ADDRESS   DATA  R/W       FC  IPL   LSA  -  8  7  -  0    TIME
 #20  F0053C  > 51CB   R   TAR  SP   0    %11111111 %11111111  307.418 MS
 #19  F0053E  > FFF8   R   TAR  SP   0    %11111111 %11111111  307.418 MS
 #18  F00540  > 6024   R   TAR  SP   0    %11111111 %11111111  307.419 MS
 #17  F00542  > 4A03   R   TAR  SP   0    %11111111 %11111111  307.420 MS
 #16  F00534  > 007E   R   TAR  SP   0    %11111111 %11111111  307.421 MS
 #15  F00536  > 301A   R   TAR  SP   0    %11111111 %11111111  307.421 MS
 #14  F00538  > E309   R   TAR  SP   0    %11111111 %11111111  307.422 MS
 #13  F0053A  > 6506   R   TAR  SP   0    %11111111 %11111111  307.423 MS
 #12  F005BA  > 0000   R   TAR  SD   0    %11111111 %11111111  307.424 MS
 #11  F0053C  > 51CB   R   TAR  SP   0    %11111111 %11111111  307.424 MS
 #10  F0053E  > FFF8   R   TAR  SP   0    %11111111 %11111111  307.425 MS
  #9  F00540  > 6024   R   TAR  SP   0    %11111111 %11111111  307.426 MS
  #8  F00542  > 4A03   R   TAR  SP   0 IP %11111111 %11111111  307.427 MS
  #7  F00534  > 007E   R   TAR  SP   0 IP %11111111 %11111111  307.427 MS
  #6  F00536  > 301A   R   TAR  SP   0 IP %11111111 %11111111  307.428 MS
  #5  F00538  > E309   R   TAR  SP   0 IP %11111111 %11111111  307.429 MS
  #4  F0053A  > 6506   R   TAR  SP   0 IP %11111111 %11111111  307.429 MS
  #3  F005BC  > 0000   R   TAR  SD   0 IP %11111111 %11111111  307.430 MS
  #2  F0053C  > 51CB   R   TAR  SP   0 IP %11111111 %11111111  307.431 MS
  #1  F0053E  > FFF8   R   TAR  SP   0 IP %11111111 %11111111  305.431 MS
  #0  BREAK
```

```
                                    68020
                                MODE 1 DISPLAY
>DRT
LINE  ADDRESS   DATA      R/W         FC     IPL       DS   SZ   OCS    TIME
 #20  F0053C  > 51CBFFFF   R    TAR   SP      0        01   00    0     1.396 S
 #19  F0053E  > FFF8FFFF   R    TAR   SP      0        01   10    1     1.396 S
 #18  F00540  > 6024FFFF   R    TAR   SP      0        01   00    0     1.396 S
 #17  F00542  > 4A03FFFF   R    TAR   SP      0        01   10    1     1.396 S
 #16  F00534  > 007EFFFF   R    TAR   SP      0        01   00    0     1.396 S
 #15  F00536  > 301AFFFF   R    TAR   SP      0        01   10    1     1.396 S
 #14  F00538  > E309FFFF   R    TAR   SP      0        01   00    0     1.396 S
 #13  F0053A  > 6506FFFF   R    TAR   SP      0        01   10    1     1.396 S
 #12  F005B4  > 004AFFFF   R    TAR   SD      0        01   10    0     1.396 S
 #11  F0053C  > 51CBFFFF   R    TAR   SP      0        01   00    0     1.396 S
 #10  F0053E  > FFF8FFFF   R    TAR   SP      0        01   10    1     1.396 S
  #9  F00540  > 6024FFFF   R    TAR   SP      0        01   00    0     1.396 S
  #8  F00542  > 4A03FFFF   R    TAR   SP      0        01   10    1     1.396 S
  #7  F00534  > 007EFFFF   R    TAR   SP      0    IP  01   00    0     1.396 S
  #6  F00536  > 301AFFFF   R    TAR   SP      0    IP  01   10    1     1.396 S
  #5  F00538  > E309FFFF   R    TAR   SP      0    IP  01   00    0     1.396 S
  #4  F0053A  > 6506FFFF   R    TAR   SP      0    IP  01   10    1     1.396 S
  #3  F005B6  > 0056FFFF   R    TAR   SD      0    IP  01   10    0     1.396 S
  #2  F0053C  > 51CBFFFF   R    TAR   SP      0    IP  01   00    0     1.396 S
  #1  F0053E  > FFF8FFFF   R    TAR   SP      0    IP  01   10    1     1.396 S
  #0  BREAK
```

(continued)

```
                              68020
                          MODE 2 DISPLAY
>DRT
LINE  ADDRESS     DATA     R/W        FC   DS   SZ   OCS    TIME
 #20  FFF00506  > 00C0FFFF  R   TAR   SP   01   10   1     1.215 S
 #19  FFF00508  > 206DFFFF  R   TAR   SP   01   00   0     1.215 S
 #18  FFF0050A  > 041AFFFF  R   TAR   SP   01   10   1     1.215 S
 #17  FFF21C72  < FFFBFFFB  W   TAR   SD   01   00   0     1.215 S
 #16  FFF21C74  < 00400040  W   TAR   SD   01   10   1     1.215 S
 #15  FFF21C6E  < FFF2FFF2  W   TAR   SD   01   00   0     1.215 S
 #14  FFF21C70  < 04220422  W   TAR   SD   01   10   1     1.215 S
 #13  FFF0050C  > 2268FFFF  R   TAR   SP   01   00   0     1.215 S
 #12  FFF0050E  > 000CFFFF  R   TAR   SP   01   10   1     1.215 S
 #11  FFF2041A  > FFF2FFFF  R   TAR   SD   01   00   0     1.215 S
 #10  FFF2041C  > 0422FFFF  R   TAR   SD   01   10   1     1.215 S
  #9  FFF00510  > 4A29FFFF  R   TAR   SP   01   00   0     1.215 S
  #8  FFF00512  > 0007FFFF  R   TAR   SP   01   10   1     1.215 S
  #7  FFF2042E  > FFFBFFFF  R   TAR   SD   01   00   0     1.215 S
  #6  FFF20430  > 0040FFFF  R   TAR   SD   01   10   1     1.215 S
  #5  FFF00514  > 6A02FFFF  R   TAR   SP   01   00   0     1.215 S
  #4  FFF00516  > 6106FFFF  R   TAR   SP   01   10   1     1.215 S
  #3  FFFB0047  > 44FFFFFF  R   TAR   SD   10   01   0     1.215 S
  #2  FFF00518  > 4CDFFFFF  R   TAR   SP   01   00   0     1.215 S
  #1  FFF0051A  > 0300FFFF  R   TAR   SP   01   10   1     1.215 S
  #0  BREAK
```

```
                                  68020
                              MODE 3 DISPLAY
>DRT
LINE  ADDRESS      DATA  R/W        FC    IPL       LSA - O     TIME
 #20  FFF21C5E   < FFFB   W    TAR  SD     0  IP   %111111  500.183 MS
 #19  FFF21C60   < 0040   W    TAR  SD     0  IP   %111111  500.184 MS
 #18  FFF21C5A   < FFF2   W    TAR  SD     0  IP   %111111  500.184 MS
 #17  FFF21C5C   < 0422   W    TAR  SD     0  IP   %111111  500.185 MS
 #16  FFF21C56   < 0000   W    TAR  SD     0  IP   %111111  500.186 MS
 #15  FFF21C58   < 0000   W    TAR  SD     0  IP   %111111  500.186 MS
 #14  FFF21C52   < 0000   W    TAR  SD     0  IP   %111111  500.187 MS
 #13  FFF21C54   < 0000   W    TAR  SD     0  IP   %111111  500.188 MS
 #12  FFF21C4E   < 0000   W    TAR  SD     0  IP   %111111  500.188 MS
 #11  FFF21C50   < 0008   W    TAR  SD     0  IP   %111111  500.189 MS
 #10  FFF21C4A   < 0000   W    TAR  SD     0  IP   %111111  500.190 MS
  #9  FFF21C4C   < 00FE   W    TAR  SD     0  IP   %111111  500.190 MS
  #8  FFF21C46   < 0000   W    TAR  SD     0  IP   %111111  500.191 MS
  #7  FFF21C48   < 0003   W    TAR  SD     0  IP   %111111  500.192 MS
  #6  FFF21C42   < 0000   W    TAR  SD     0  IP   %111111  500.192 MS
  #5  FFF21C44   < 0003   W    TAR  SD     0  IP   %111111  500.193 MS
  #4  FFF21C3E   < 0014   W    TAR  SD     0  IP   %111111  500.193 MS
  #3  FFF21C40   < 000C   W    TAR  SD     0  IP   %111111  500.194 MS
  #2  FFF21C3A   < 0000   W    TAR  SD     0  IP   %111111  500.195 MS
  #1  FFF21C3C   < 0001   W    TAR  SD     0  IP   %111111  500.195 MS
  #0  BREAK
```

LINE	Line number 0 in the trace buffer indicates the last bus cycle prefetched or executed before the Emulator went into pause mode. The larger the LINE number the further back in the history of the program you are viewing. You can get a good idea of the relationship of bus cycles to instructions by matching the bus cycle LINE numbers in the DRT to the SEQ# in the disassembled trace.
ADDRESS DATA	The address displayed is where the bus cycle took place, along with the DATA written to, or read from, that address.

(continued)

	$\boxed{>}$ and $\boxed{<}$ are data direction indicators. They display whether data was read from an address ($\boxed{>}$) or written to an address ($\boxed{<}$). These same indicators are used in the trace disassembly.
R/W	Indicates whether the cycle was a read or a write.
TAR/OVL	TAR/OVL indicates whether the access was in the target memory area or in the Emulator's overlay memory (see **DM** command to determine what addresses are mapped).
RM (68020, Mode 3)	68020 RMC signal. Indicates the cycle was an indivisible read-modify-write cycle.
FC	Indicates the state of FC0-2.
V	Marks all cycles that occurred after a memory write violation or memory access violation.
B	Marks all bus cycles that occurred after a breakpoint.
BER	Indicates the BERR signal was asserted.
IPL	Indicates the interrupt level encoded on the IPL0-2 lines.
CD (68020, Mode 3)	Indicates the CDIS signal was asserted.
AV (68020, Mode 0,1,3)	Indicates the AVEC signal was asserted.
IP (68020, Mode 0,1,3)	Indicates the IPEND signal was asserted.
LSA	LSA 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 can cause event actions such as selective tracing, counting, or breaking emulation (refer to the Event Monitor System, Section 7 for 68000/08/10 or Section 8 for 68020).
DS (68020, Mode 1, 2)	Indicates the state of DSACK0 and DSACK1.
SZ (68020, Mode 1, 2)	Indicates the state of SIZ0 and SIZ1.
OCS (68020, Mode 1, 2)	Indicates whether the OCS signal was asserted.
TIME (68020)	Indicates the time elapsed since emulation was entered.
VM (68000/10)	Indicates the VMA signal was asserted.
VP (68000/08/10)	Indicates the VPA signal was asserted.

CLEAR TRACE MEMORY (68020)

Command	Result
CLT	All current trace memory is cleared. Applies to 68020 only.

DISASSEMBLE TRACE MEMORY

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

Comments

This command is valid only in pause mode, unless you have the dynamic trace feature.

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

(continued)

will give a listing similar to a program listing without the **STP;DT** line between each command.

68020

Disassemble trace (68020 microprocessor) requires the following:

1. You must be in trace mode 2 (Set parameter #4, page 5-3). An error message will display if you are not in trace mode.

2. The **CDS** switch must be enabled (ON).

Examples

These two commands used in conjunction will give the user output similar to a program listing.

```
>STP;DT
```

```
>DT 0
SEQ#       ADDR       OPCODE    MNEMONIC  OPERAND FIELDS AND BUS CYCLE DATA
---------------------------------------------------------------------------
0051       00000192   64EA      BCC.S     $0000017E
0047       0000017E   13C20000  MOVE.B    D2,$0000A001
                      A001                0000A001<3C
0044       00000184   D081      ADD.L     D1,D0
0043       00000186   64000008  BCC.W     $00000190
0039       00000190   D885      ADD.L     D5,D4
0038       00000192   64EA      BCC.S     $0000017E
0034       0000017E   13C20000  MOVE.B    D2,$0000A001
                      A001                0000A001<3C
0031       00000184   D081      ADD.L     D1,D0
0030       00000186   64000008  BCC.W     $00000190
0026       00000190   D885      ADD.L     D5,D4
0025       00000192   64EA      BCC.S     $0000017E
0021       0000017E   13C20000  MOVE.B    D2,$0000A001
                      A001                0000A001<3C
0018       00000184   D081      ADD.L     D1,D0
0017       00000186   64000008  BCC.W     $00000190
0013       00000190   D885      ADD.L     D5,D4
0012       00000192   64EA      BCC.S     $0000017E
0008       0000017E   13C20000  MOVE.B    D2,$0000A001
                      A001                0000A001<3C
>
```

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.

(continued)

MNEMONIC	The command used to invoke the instruction.
OPERAND FIELD	The assembly-language instruction.
BUS CYCLE DATA	The bus cycle transaction, if any, that occurred as a result of the instruction. This includes any information written to, or read from, memory or I/O locations. If several branches are taken within a short period of time the trace disassembler may not be able to decide which branches were taken. These instructions are tagged with question marks.

SELECT TIMER FREQUENCY (68020)

Command	Result
TIM = *<n>*	Set time stamp resolution: 1 - 0.1 MHz 2 - 1.0 MHz 3 - 10.0 MHz
TIM	Display time stamp resolution. Default: 10 MHz

Examples

```
>TIM
$00000003
```

DISPLAY RAW TRACE (68020)

Command	Result
DST	Display the last page of bus cycles recorded in trace memory with full timestamp resolution.
DST *<line number>*	Displays a page of the trace buffer starting with *<line number>*.
DST *<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 page 5-3).

This command is valid only in pause mode, unless you have the dynamic trace feature.

See DRT, page 5-94 for an alternative way to display the raw trace bus cycles.

Examples

```
                         TRACE MODE 0
>DST
LINE  ADDRESS   DATA R/W       FC   DS  SZ  OCS   TIME
 #20  F0053C >  51CB  R   TAR  SP   01  00   0    0.3074177 SEC
 #19  F0053E >  FFF8  R   TAR  SP   01  10   1    0.3074183 SEC
 #18  F00540 >  6024  R   TAR  SP   01  00   0    0.3074194 SEC
 #17  F00542 >  4A03  R   TAR  SP   01  10   1    0.3074201 SEC
 #16  F00534 >  007E  R   TAR  SP   01  00   0    0.3074207 SEC
 #15  F00536 >  301A  R   TAR  SP   01  10   1    0.3074214 SEC
 #14  F00538 >  E309  R   TAR  SP   01  00   0    0.3074221 SEC
 #13  F0053A >  6506  R   TAR  SP   01  10   1    0.3074227 SEC
 #12  F005BA >  0000  R   TAR  SD   01  10   0    0.3074235 SEC
 #11  F0053C >  51CB  R   TAR  SP   01  00   0    0.3074242 SEC
 #10  F0053E >  FFF8  R   TAR  SP   01  10   1    0.3074248 SEC
  #9  F00540 >  6024  R   TAR  SP   01  00   0    0.3074259 SEC
  #8  F00542 >  4A03  R   TAR  SP   01  10   1    0.3074266 SEC
  #7  F00534 >  007E  R   TAR  SP   01  00   0    0.3074272 SEC
  #6  F00536 >  301A  R   TAR  SP   01  10   1    0.3074278 SEC
  #5  F00538 >  E309  R   TAR  SP   01  00   0    0.3074286 SEC
  #4  F0053A >  6506  R   TAR  SP   01  10   1    0.3074292 SEC
  #3  F005BC >  0000  R   TAR  SD   01  10   0    0.3074300 SEC
  #2  F0053C >  51CB  R   TAR  SP   01  00   0    0.3074306 SEC
  #1  F0053E >  FFF8  R   TAR  SP   01  10   1    0.3074313 SEC
  #0  BREAK
```

(continued)

```
                              TRACE MODE 1
>DST
LINE  ADDRESS    DATA      R/W        FC   DS   SZ   OCS     TIME
 #20  F0053C >   51CBFFFF   R    TAR   SP   01   00   0      1.3955394  SEC
 #19  F0053E >   FFF8FFFF   R    TAR   SP   01   10   1      1.3955400  SEC
 #18  F00540 >   6024FFFF   R    TAR   SP   01   00   0      1.3955411  SEC
 #17  F00542 >   4A03FFFF   R    TAR   SP   01   10   1      1.3955418  SEC
 #16  F00534 >   007EFFFF   R    TAR   SP   01   00   0      1.3955424  SEC
 #15  F00536 >   301AFFFF   R    TAR   SP   01   10   1      1.3955431  SEC
 #14  F00538 >   E309FDFF   R    TAR   SP   01   00   0      1.3955438  SEC
 #13  F0053A >   4506FFFF   R    TAR   SP   01   10   1      1.3955444  SEC
 #12  F005B4 >   004AFFFF   R    TAR   SD   01   10   0      1.3955452  SEC
 #11  F0053C >   51CBFFFF   R    TAR   SP   01   00   0      1.1955459  SEC
 #10  F0053E >   FFF8FFFF   R    TAR   SP   01   10   1      1.3955465  SEC
  #9  F00540 >   6024FFFF   R    TAR   SP   01   00   0      1.3955476  SEC
  #8  F00542 >   4A03FFFF   R    TAR   SP   01   10   1      1.3955483  SEC
  #7  F00534 >   007EFFFF   R    TAR   SP   01   00   0      1.3955489  SEC
  #6  F00536 >   301AFFFF   R    TAR   SP   01   10   1      1.3955495  SEC
  #5  F00538 >   E309FFFF   R    TAR   SP   01   00   0      1.3955503  SEC
  #4  F0053A >   6506FFFF   R    TAR   SP   01   10   1      1.3955510  SEC
  #3  F005BC >   0056FFFF   R    TAR   SD   01   10   0      1.3955518  SEC
  #2  F0053C >   51CBFFFF   R    TAR   SP   01   00   0      1.3955524  SEC
  #1  F0053E >   FFF8FFFF   R    TAR   SP   01   10   1      1.3955531  SEC
  #0  BREAK
```

```
                           TRACE MODE 2
>DST
LINE  ADDRESS     DATA     R/W       FC   DS  SZ  OCS   TIME
 #20  FFF21C5E <  FFFBFFFB  W   TAR  SD   01  00  0     1.2638674 SEC
 #19  FFF21C60 <  00400040  W   TAR  SD   01  10  1     1.2638680 SEC
 #18  FFF21C5A <  FFF2FFF2  W   TAR  SD   01  00  0     1.2638688 SEC
 #17  FFF21C5C <  04220422  W   TAR  SD   01  10  1     1.2638694 SEC
 #16  FFF21C56 <  00000000  W   TAR  SD   01  00  0     1.2638700 SEC
 #15  FFF21C58 <  00000000  W   TAR  SD   01  10  1     1.2638707 SEC
 #14  FFF21C52 <  00000000  W   TAR  SD   01  00  0     1.2638714 SEC
 #13  FFF21C54 <  00000000  W   TAR  SD   01  10  1     1.2638720 SEC
 #12  FFF21C4E <  00000000  W   TAR  SD   01  00  0     1.2638728 SEC
 #11  FFF21C50 <  00080008  W   TAR  SD   01  10  1     1.2638734 SEC
 #10  FFF21C4A <  00000000  W   TAR  SD   01  00  0     1.2638741 SEC
  #9  FFF21C4C <  00FE00FE  W   TAR  SD   01  10  1     1.2638748 SEC
  #8  FFF21C46 <  00000000  W   TAR  SD   01  00  0     1.2638755 SEC
  #7  FFF21C48 <  00030003  W   TAR  SD   01  10  1     1.2638761 SEC
  #6  FFF21C42 <  00000000  W   TAR  SD   01  00  0     1.2638768 SEC
  #5  FFF21C44 <  00030003  W   TAR  SD   01  10  1     1.2638774 SEC
  #4  FFF21C3E <  00140014  W   TAR  SD   01  00  0     1.2638781 SEC
  #3  FFF21C40 <  000C000C  W   TAR  SD   01  10  1     1.2638788 SEC
  #2  FFF21C3A <  00000000  W   TAR  SD   01  00  0     1.2638795 SEC
  #1  FFF21C3C <  00010001  W   TAR  SD   01  10  1     1.2638801 SEC
  #0  BREAK
```

(continued)

```
                              MODE 3 DISPLAY
>DST
LINE  ADDRESS     DATA R/W        FC   DS  SZ  OCS    TIME
 #20  FFF21C5E <  FFFB  W   TAR   SD   01  00   0     0.5001831 SEC
 #19  FFF21C60 <  0040  W   TAR   SD   01  10   1     0.5001838 SEC
 #18  FFF21C5A <  FFF2  W   TAR   SD   01  00   0     0.5001844 SEC
 #17  FFF21C5C <  0422  W   TAR   SD   01  10   1     0.5001851 SEC
 #16  FFF21C56 <  0000  W   TAR   SD   01  00   0     0.5001857 SEC
 #15  FFF21C58 <  0000  W   TAR   SD   01  10   1     0.5001863 SEC
 #14  FFF21C52 <  0000  W   TAR   SD   01  00   0     0.5001870 SEC
 #13  FFF21C54 <  0000  W   TAR   SD   01  10   1     0.5001876 SEC
 #12  FFF21C4E <  0000  W   TAR   SD   01  00   0     0.5001883 SEC
 #11  FFF21C50 <  0008  W   TAR   SD   01  10   1     0.5001889 SEC
 #10  FFF21C4A <  0000  W   TAR   SD   01  00   0     0.5001895 SEC
  #9  FFF21C4C <  00FE  W   TAR   SD   01  10   1     0.5001902 SEC
  #8  FFF21C46 <  0000  W   TAR   SD   01  00   0     0.5001908 SEC
  #7  FFF21C48 <  0003  W   TAR   SD   01  10   1     0.5001915 SEC
  #6  FFF21C42 <  0000  W   TAR   SD   01  00   0     0.5001921 SEC
  #5  FFF21C44 <  0003  W   TAR   SD   01  10   1     0.5001927 SEC
  #4  FFF21C3E <  0014  W   TAR   SD   01  00   0     0.5001934 SEC
  #3  FFF21C40 <  000C  W   TAR   SD   01  10   1     0.5001940 SEC
  #2  FFF21C3A <  0000  W   TAR   SD   01  00   0     0.5001947 SEC
  #1  FFF21C3C <  0001  W   TAR   SD   01  10   1     0.5001953 SEC
  #0  BREAK
```

LINE	Line number 0 in the trace buffer indicates the last bus cycle prefetched or executed before the Emulator went into pause mode. The larger the LINE number the further back in the history of the program you are viewing. You can get a good idea of the relationship of bus cycles to instructions by matching the bus cycle LINE numbers in the DRT to the SEQ# in the disassembled trace.
ADDRESS DATA	The address displayed is where the bus cycle took place, along with the DATA written to, or read from, that address.

	$\boxed{>}$ and $\boxed{<}$ are data direction indicators. They display whether data was read from an address ($\boxed{>}$) or written to an address ($\boxed{<}$). These same indicators are used in the trace disassembly.
R/W	Indicates whether the cycle was a read or a write.
TAR/OVL	TAR/OVL indicates whether the access was in the target memory area or in the Emulator's overlay memory (see **DM** command to determine what addresses are mapped).
FC	Indicates the state of FC0-2.
DS (68020, Mode 1, 2)	Indicates the state of DSACK0 and DSACK1.
SZ (68020, Mode 1, 2)	Indicates the state of SIZ0 and SIZ1.
OCS (68020, Mode 1, 2)	Indicates whether the OCS signal was asserted.
TIME (68020)	Indicates the elapsed time since emulation was entered.

SEARCH RAW TRACE (68020)

Command	Result
STA <*address*>	Search raw trace for address. Enter address pattern that looks like raw trace.
STD <*data*>	Search raw trace for data. Enter data pattern that looks like raw trace.
STS <*status*>	Search raw trace for status. See page 8-17 for the position and meaning of the traced status bits.

Comments

The code only looks for the amount of information appropriate to the current trace mode. That is, in trace mode 0 it only looks for a 24 bit address pattern since that is all that is traced.

Examples

```
STA 1680                          address 1680 only
STA 16XX DC OFF                   addresses 1600 through 16FF
STS 50 DC %1111111110001111       all Supervisor Data cycles
STS 0 DC %1111111110001111        all writes
```

The ES 1800 response is to display all occurrences of the matched pattern in trace memory in the same format as for DRT (page 5-94). The first line is preceded by the raw trace header. If there are more lines than the current screen length, the display pauses at the bottom of the screen and waits for a key from the user. $\boxed{\text{RETURN}}$ continues the trace search. Any other key aborts the search.

DISASSEMBLE TRACE PAGE

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

Comments

This command is valid only in pause mode, unless you have the dynamic trace feature.

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

Refer also to the **DT** command, page 5-103, the **DRT** command, page 5-94, and the repeat command, page 5-125.

Macros

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

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

If the lengths of all ten macros exceeds the buffer length of 125 characters, the highest numbered macro is truncated.

EXAMPLE: If macros #1 to #8 are defined and in this process have used up all of the space in the buffer, then an attempt to define macro #9 and #0 would result in those macros remaining null. Also, if the length of any macro from #1 to #7 was increased after filling the buffer, then macro #8 would be truncated. If the increase was more than the size of macro #8, macro #8 would become null and macro #7 would be truncated.

There are no warnings when truncation or nullification takes place. If a number of long macros are defined, execute the MAC command to determine if the macros of the highest numbers are still intact. Using the general purpose registers in macros will help save the number of characters used.

Macros can be saved in the Emulator EEPROM. Refer to the **LD** and **SAV** (pages 5-27 and 5-25) commands for information on saving and reloading macros.

DISPLAY DEFINED MACROS

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

Examples

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

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

Comments

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

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

Examples

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

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

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

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

(continued)

Step and disassemble one instruction at a time.

```
    >_1= DB USP LEN 20;RET;DIS PC LEN 12
```

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

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

Macro number three is executed.

```
    >_3
```

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

Examples

Clear macro #1.

```
>_1=
```

(continued)

The Repeat Operators

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

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

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

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

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

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

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

```
>*STP;DT; TST=PC-$C324
```

If you are waiting for a RAM location to be cleared:

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

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

The TST register 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 registers 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.

Examples

The following examples show some interesting uses of the LIM and IDX registers.

If you need a decimal counter:

```
 >BAS IDX=#10
 >*3 IDX
#0
#1
#2
```

(continued)

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

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

REPEAT COMMAND LINE

Command	Result
/	Re-execute the previous command line. No RETURN is necessary.

Comments

The slash must be the first character on a line to be recognized as the repeat character.

Examples

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

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

This causes the system to single step and disassemble memory starting at the program counter (PC) location.

```
>STP;DIS PC LEN 10
>/
```

(continued)

Symbols

Symbol definitions provide the capability to refer to addresses and data values using names, rather than numbers. Symbols are 32 bit integer values and sections are 32 bit ranges. Symbols and sections are sometimes collectively referred to as symbols.

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

Symbols are not typed within the Emulator, thus all symbols are global. This implies that a symbol and a section may not be defined using the same name, a symbol name may only be defined once, and 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 your program. These symbols can be easily downloaded if you have a linker and converter that can create Extended Tekhex serial data records. See the SET command (page 5-3) for the serial data format variable. If you are going to download sections that have already been defined (perhaps from a previous download of the same file), purge all symbols or delete the section definitions from memory before downloading. If you do not, an error occurs when you attempt to redefine the value of a section and the download will abort.

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, the symbols are defined.

```
 >SYM
$00000166 LOOP
$00003000 I/O_port_0
 >SEC
$00000166 TO $0000016C DEMON.MODULE
 >
```

The following example shows memory disassembly with symbol definitions.

```
                                               OPERAND       BUS CYCLE
SEQ#         ADDR       OPCODE      MNEMONIC   FIELDS          DATA
---------------------------------------------------------------------
SEC: DEMON.MODULE
0009+LOOP
0009+000000 31C23000                MOVE.W     D2,I/O_PORT_0 003000<8787
0008+000004 D081                    ADD.L      D1,D0
0007+000006 64000004                BCC.L      $000172
0006 000172 D885                    ADD.L      D5,D4
0005 000174 64F0                    BCC.S      $LOOP
SEC: DEMON.MODULE
0004+LOOP
0004+000000 31C23000                MOVE.W     D2,I/O_PORT_0 003000<8787
0003+000004 D081                    ADD.L      D1,D0
0002+000006 64000004                BCC.L      $000172
0001 000172 D885                    ADD.L      D5,D4
0000 000174 64F0                    BCC.S      $Loop
```

DISPLAY SYMBOLS

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

Examples

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

DISPLAY SECTION

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

Examples

```
  >'sec = 1000 LEN 1F
  >'init_mod = 'start TO 'end
  >'ram =$0100 TO $0FFF
  >SEC
$00001000 TO $0000101F sec
$00008000 TO $0000837E init_mod
$00000100 TO $00000FFF RAM
```

SYMBOL DEFINITION

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

Comments

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

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

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

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

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

Examples

"testing" is recognized as the symbol.

```
>'testing =GR0
```

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

```
>'testing=GR0
                ?
>?
ERROR #77
UNDEFINED SYMBOL OR CHARACTER DETECTED
```

```
>'section_X =1000 TO 1FFF
>'main_loop ='prog_start TO 'RAM_START - 1
```

DELETE A SYMBOL OR SECTION

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

Examples

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

DELETE ALL SYMBOLS AND SECTIONS

Command	Result
PUR	Purge all symbols and section references.

Comments

Be sure to purge before downloading symbols that may already be defined. If you don't, an error occurs and the download will be aborted.

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

(continued)

Miscellaneous Commands

DISPLAY THE SOFTWARE REVISION DATES

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

Comments

This command is valid only in pause mode.

If you call AMC Customer Service, they will ask you what software revisions are in your machine. This command gives you this information.

Examples

```
 >REV
WED AUG 6 08:50:26 PDT 1986 - ESL 2.2
WED AUG 6 16:50:26 PDT 1986 - EMU 3.12
 >
```

DISPLAY A BLANK LINE

Command	Result
RET	Executes a RETURN, linefeed.

Comments

This command is used to improve readibility when displaying multiple lines of data.

Examples

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

```
 >DB 0 LEN 20;RET;DB 100 LEN 20
00000000  00 00 9F FE 00 00 01 00 - 00 00 01 AC 00 00 01 B2 ..............
00000010  00 00 01 B8 00 00 01 BE - 00 00 01 C4 00 00 01 CA ..............

00000100  30 3C 01 00 13 FC 00 FF - 00 00 A0 01 13 FC 00 FE 0<............
00000110  00 00 A0 01 13 FC 00 FC - 00 00 A0 01 13 FC 00 F8 ..............
```

SECTION 6

Table of Contents

Target Commands

TARGET COMMANDS

Introduction

The term "run mode" is used to indicate that emulation has begun, that is, the microprocessor in the pod is running a program in the target. Generally, "target" refers to the hardware and software that you are debugging. If there is no target hardware available, the target may be just a program, downloaded into overlay memory. The microprocessor in the Emulator's pod replaces the microprocessor in the target. This provides the Emulator control of the processor, which in turn provides you with the powerful Event Monitor System and the ability to see what is happening within the target system.

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

Emulation

STARTING EMULATION

Entry into emulation or run mode is accomplished by executing any of four run commands. Two of the run commands load the reset vectors before entering run mode, and two of them enable the breakpoints in the Event Monitor. Event system breakpoints may be enabled or disabled during run mode. Even if breakpoints are disabled, all other Event Monitor System functions are active.

The reset vectors cannot be loaded during run mode. The following table provides a quick reference to the run commands.

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

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

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

M	- Memory Mode
@	- Indirection Operator
DB	- Display Block of Memory
ASM	- In-Line Assembler
DIS	- Memory Disassembler
NXT	- Memory Mode
LST	- Memory Mode

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

HALTING EMULATION

Emulation can be halted in one of four ways:

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

3. You can reset the pod or Emulator. This is done by entering a reset command, **RST**, or the system reset character defined in the **SET** menu.

4. Emulation breaks automatically if an access or write violation occurs in the target system. An error message indicates the condition that caused the error.

USING REGISTERS IN RUN MODE

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

Because of this, modifying these registers during run mode simply alters the Emulator's image of the registers. The Emulator does not download the new values of the registers to the microprocessor. When emulation is broken, the current values of the microprocessor registers are uploaded, 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.

Command	Result
RBK	Begins executing the target program at the current PC memory location. If the run with breakpoints is executed while already in run mode (after a **RUN** command is entered), this will enable breakpoints without stopping emulation.
RBV	(68000/08/10) The Emulator loads the SSP and PC registers with the values located in supervisor program space addresses $0 and $4 respectively, initializes the SR register to $2700, and starts emulation with breakpoints enabled. (68020) The Emulator loads the ISP and PC registers with the values located in supervisor program space addresses $0 and $4 respectively, initializes the SR register to $2700, and starts emulation with breakpoints enabled.

(continued)

RUN	Begins executing the target program at the current PC memory location.
RNV	(68000/08/10) The Emulator loads the SSP and PC registers with the values located in supervisor program space addresses $0 and $4 respectively, initializes the SR register to $2700, and starts emulation. (68020) The Emulator loads the ISP and PC registers with the values located in supervisor program space addresses $0 and $4 respectively, initializes the SR register to $2700, and starts emulation.

Comments

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

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

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

Examples

In the following example the Emulator begins running the program at the current PC register address and breaks (halts emulation) when AC1 is encountered.

```
>AC1 = <address>; WHEN AC1 THEN BRK
>RBK
```

In the example below the Emulator begins running the program at the current PC register addresses, waits for the breakpoint, then disassembles one page.

```
>AC1 = <address>;WHEN AC1 THEN BRK
>RBK;WAIT;DTB
```

STOP AND STEP TARGET SYSTEM

Command	Result
R>STP	Stop emulation and return to pause mode. The screen displays the current PC address, the Event Monitor System group number, and prompt for further instruction.
>STP	From pause mode, the **STP** command executes one instruction. To receive visual feedback, combine this command with a display command such as **STP;DT**.

Examples

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

LOAD RESET VECTORS

Command	Result
LDV	The Emulator loads SSP and PC registers with values located in supervisor program space addresses $0 and $4 respectively and initializes the status register (SR) to $2700.
LDV;RUN	The Emulator loads SSP and PC registers with values located in supervisor program space addresses $0 and $4 respectively, initializes the status register (SR) to $2700, and starts emulation with breakpoints enabled. NOTE: See the **RNV** command for auto entry of this function.
LDV;RBK	The Emulator loads SSP and PC registers with values located in supervisor program space addresses $0 and $4 respectively, initializes the status register (SR) to $2700, and starts emulation with breakpoints enabled.

(continued)

Comments

This command valid in pause mode only.

RNV and **RBV** also load the reset vectors, then enter run mode.

To verify that the reset vectors are loaded, execute the **DR** command or individually display the PC and SSP (ISP for 68020) registers.

Refer also to Registers (page 5-68), and the **DR** command (page 5-76).

Examples

Display the registers, then load the reset vectors, clear the address and data registers, and verify the changes by redisplaying the register set.

```
>DR;LDV;CLR;DR
      - 0 -    - 1 -    - 2 -    - 3 -     - 4 -    - 5 -    - 6 -    - 7 -
D = 5D480000 01030000 78787878 00000000  17000000 01000000 00000000 00000000
A = 00000000 00000000 00000000 00000000  00000000 00000000 00000000
     SR  =  .SI7                          SFC = 0            MSP = 00000000
     CCR =  .....     CAAR = 00000000     DFC = 0            ISP = 00009FFE
     PC  =  00000190  CACR = 00000000     VBR = 00000000     USP = 00000000
      - 0 -    - 1 -    - 2 -    - 3 -     - 4 -    - 5 -    - 6 -    - 7 -
D = 00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
A = 00000000 00000000 00000000 00000000  00000000 00000000 00000000
     SR  =  .SI7                          SFC = 0            MSP = 00000000
     CCR =  .....     CAAR = 00000000     DFC = 0            ISP = 00009FFE
     PC  =  00000100  CACR = 00000000     VBR = 00000000     USP = 00000000
>
```

DELAY EXECUTION UNTIL EMULATION BREAK

Command	Result
WAI	Delay executing the specified command until emulation is broken.

Comments

Usually this command is used to delay executing a display command until an Event Monitor System breakpoint is reached.

An event may never occur to bring the Emulator out of run mode. When this happens, use the reset character to reset the system.

After a reset, the delayed command is lost from the input buffer.

Examples

The Emulator disassembles a page of trace after a breakpoint is reached. Entering **RBK;DTB** without the **WAI** command, results in an error. The error message COMMAND INVALID DURING EMULATION is displayed.

```
RBK;WAI;DTB
```

(continued)

The Emulator runs until an access violation or a write violation is encountered, then displays a message pointed at by the A4 register.

```
RUN;WAI;DIA A4
```

Command	Result
CYC	Causes target system to execute a single bus cycle.

Comments

When executing the **CYC** command a bus error is initialized automatically. Refer to page 5-72 for a listing of the bus errors. This forced bus error will make available the stack register information that simulates bus cycles.

CYC does not actually execute the target program the way a **STP** command does. The individual cycles are simulated by performing peeks/pokes to the target system. The Emulator does enter **RUN** while executing the bus cycles. Trace memory does not keep a record of transactions to and from the target systems.

This command is only operable on the 68010 microprocessors.

RESET

Command	Result
RST	The microprocessor reset pin is asserted for 512 clock cycles.

Comments

The **RST** command is not valid during emulation.

The reset pin on the microprocessor is asserted and affects any hardware interfaced to it.

Memory Commands

Memory commands allow you to modify and display memory in a number of different ways. "Memory" refers to memory in your target system or the Emulator's overlay memory. If the overlay memory is mapped (mapped memory will have the RW, RO or ILG attributes assigned to it), read and write accesses are directed to it. Mapped memory is modified by a memory command even if it is mapped as read only. If memory is unmapped, (memory with the TGT attribute assigned to it), memory command accesses are directed to the target system memory. Mapped and unmapped memory may be interleaved in any way you desire. See the Overlay Memory section (page 5-53) for details.

The default data length affects most memory commands. There are three data lengths to choose from: byte mode, (BYM), word mode (WDM), and long word mode (LWM). 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 **FIN** command truncates the data field to `F6` and searchs the range for that byte. Commands that display data use the current data length.

Some memory commands may be executed during run mode. These commands halt emulation for a brief time in order to read from or write to memory. If memory commands are executed while in the run mode, remember that you are not emulating in real-time.

(continued)

The following table shows the commands that can be entered in run mode and the commands that are affected by the default data length.

Command	Legal in Run Mode?	Uses Default Data Length?
DB	YES	YES
FIN	NO	YES
FIL	NO	YES
BMO	NO	NO
VBL	NO	NO
LOV	NO	NO
VFO	NO	NO
ASM	YES	N/A
DIS	YES	N/A
M	YES	YES
@	YES	YES

DISPLAY MEMORY BLOCK

Command	Result
DB *<address range>*	Reads and displays the specified address range.
DB	Reads and displays one page of memory, starting at the last address displayed by any previous **DB** command. On power-up, this command displays a page of memory from address zero.
DB *<address>*	Reads and displays one page of memory, starting at the specified address.

Comments

The page length is defined by a 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 / (page 5-125). This repeats the **DB** command to increment the address and scroll through memory.

(continued)

If the display is longer than one page, the XON/XOFF characters can be used to start and stop scrolling (page 5-3).

This command affects real time operation when entered in run mode (see page 6-15).

Examples

Displays $20 words pointed to by D0.

```
>WDM; DB D0 LEN 40
```

Displays a page of values pointed at by the value on top of the user stack (see page 4-13 for information on @ operator).

```
>DB @USP
```

Display block in byte mode, word mode, and long word mode.

```
>BYM
>DB 0 LEN 20
000000  80 48 45 4C 4C 4F 80 80 - 2F 0F F1 F9 5E 2F F6 F0  .HELLO../...^/..
000010  0F 03 F0 40 0F 0C F0 40 - 07 06 F0 90 0F 0C D8 00  ...@...@........

>WDM
>DB 0 LEN 30
000000  4880  4C45  4F4C  8080 -  0F2F  F9F1  2F5E  F0F6
000010  030F  40F0  0C0F  40F0 -  0607  90F0  0C0F  00D8
000020  0FFF  F9FF  1FFF  7FFF -  3FFF  BDFF  1FFF  FFFF

>LWM
>DB 0 LEN 30
000000   48804C45  454C8080   0F2FF9F1  2F5ER0F6
000010   030F40F0  0C0F40F0   060790F0  0C0F00D8
000020   0FFFF9FF  IFFF7FFF   3FFFBDFF  IFFFFFFF
```

Command	Result
FIN [.B .W .L] *<range>*, *<data>*	Search *<range>* for the data pattern. All occurrences of the pattern are displayed:

```
$<address>=$<data>
 >
```

If the pattern is not found within the range:

```
NOT FOUND
 >
```

Comments

This command is valid in pause mode only.

Data may be either an integer or don't care value. Find uses the default data length, regardless of the length of the *<data>*.

(continued)

Examples

To find a bit pattern using don't cares use either of the following forms:

```
>FIN 1000 TO 2FFF, 60XX

   or

>FIN 2000 LEN 2000,6000 DC OFF
```

Find the initialization data in the start module section.

```
>FIN 'start_module,'init_uart
```

Find any NOPs in the range.

```
>FIN.W 100 TO 1000,4E71
```

FILL OPERATOR

Command	Result
FIL[.B .W .L]<*range*>,<*constant*>	Fills <*range*> with the <*constant*> data pattern.

Comments

This command is valid in pause mode only.

<*constant*> must be an integer.

The fill command uses the default data length, regardless of the length of <*constant*>.

The fill command can be verified using the **VBL** (verify block) command (page 6-22).

Examples

Fill RAM with zero to initialize data space.

```
>FIL 2000 LEN 50,0
```

Fill RAM section with initialization data.

```
>FIL RAM, 'init_data
```

VERIFY BLOCK DATA

Command	Result
VBL *<address range>*, *<data>*	Verifies that *<address range>* contains the specified data. If there are differences: `$<address> = $XX NOT $YY` *<address>* is the address where the misverify occurred. "XX" is the actual data at the location. "YY" is the data specified in the command statement.

Comments

This command is valid only in pause mode.

The **VBL** command uses the default data length, regardless of the length of *<data>*.

Refer also to the **MMS** command (page 5-81), and the **BMO** command (page 6-24).

Examples

Verify that a range contains $3F.

```
 >VBL 0 TO 2000,3F
$00000004 - $00, NOT $3F
$00000126 - $76, NOT $3F
 >
```

BLOCK MOVE

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

Comments

This command valid in pause mode only.

The following rules of thumb may make the numerous forms of this command less confusing. If there is no space specified for the source argument, MMS is always used (page 5-81). If no space is specified for the destination address, MMD is always used.

A non-overlapping block move can be verified using the **VBL** command.

Examples

Move a range to a new location in supervisor data space.

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

       or

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

To move data from range 10 to 20 in supervisor program space to a starting address of 30 in supervisor program space:

```
MMS = SP
MMD = SP
BMO 10  TO  20, 30
```

To move data from the range 100 to 200 in supervisor program space to a starting address 300 in user data space:

```
MMS = SP
MMD = UD
BMO  100  TO  200, 300
```

(continued)

The following example moves data without setting the MMS and MMD registers. To move data from range 400 to 500 in supervisor data space to a starting address of 600 in user data space regardless of what MMS and MMD are set to:

```
BMO  400  TO  500, SD,  600,  UD
```

VERIFY BLOCK MOVE

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

(continued)

Comments

This command is valid only in pause mode.

Verifies that a non-overlapping block move was successful.

Examples

To verify data from range 400 to 500 in supervisor data space with a range starting address of 600 in user data space regardless of what MMS and MMD are set to:

```
  VBM   400  TO  500,  SD,  600, UD
```

LOAD OVERLAY MEMORY

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

Comments

This command is valid only in pause mode.

Refer to the **VFO** command, page 5-66, to verify the load overlay command.

To load overlay memory from the target memory, a target system must be connected to the ES1800 Emulator and overlay memory installed and mapped.

To load a target memory range into the overlay memory at a different address, use the **LOV** command, then do a block move of the range.

Refer also to the Overlay Memory section, page 5-53.

Examples

To load overlay memory from Target memory $80000 to $87FFF:

```
>LOV 80000 LEN 8000
>LOV 'BOOT_RANGE
```

VERIFY OVERLAY MEMORY

Command	Result
VFO *<range>*	Compares *<range>* in target memory to the same range in the overlay memory. If there are any differences, the address and data difference is displayed: `<address> = XX NOT YY` "XX" is the data present in overlay memory. "YY" is the data at that location in the target system memory.

Comments

This command is valid only in pause mode.

Refer also to the Overlay Memory section, page 5-53.

Examples

To verify the two overlay loads in the **LOV** command section:

```
   >VFO 80000 LEN 8000
   >VFO 'BOOT_RANGE
```

(continued)

Line Assembler

The Motorola 68000 series line assembler allows you to enter and assemble Motorola 68000 series mnemonic instructions into target memory. All standard Motorola 68000 mnemonics listed in the *16 Bit Microprocessor Users Manual* or *MC68020 32 Bit Microprocessors Users Manual* are supported. In addition to these instructions many assembler directives are supported.

Parameters:

- Not all arithmetic operators are available for assembly. The following arithmetic operators are allowed in an assembler expression: ($\boxed{+\ -}$). Only 16-bit arithmetic is performed. For proper usage refer to the section on expressions.
- Parentheses are allowed in group algebraic expressions.
- Double quotes ($\boxed{"}$) or a slash sign ($\boxed{/}$) is used to delineate ASCII strings. If you enclose the string in double quotes, you may not use double quotes in the string, but any number of slashes may be used.

  If you enclose the string in slashes, you may not use slashes within the string, but any number of double quotes may be used.
- Capital letters is the default for ASCII text. The use of $\boxed{\text{<backspace>}}$ will allow entry of lower case letters within a text string until you enter a $\boxed{\text{<space>}}$.
- The two number bases are hexadecimal and decimal. Decimal is the default. You do not need to enter a $\boxed{\#}$ sign to denote decimal numbers. Numbers beginning with $ are hexadecimal. All other numbers are decimal.
- When referencing memory, you must use a leading zero with the program counter relative.
- The pound sign $\boxed{\#}$ denotes an immediate addressing mode.

Examples

```
  0  (PC,D3) -PCR with index and displacement.
  0  (A4,D3) -address register indirect with index and displacement.
```

LINE ASSEMBLER

Command	Result
ASM	Assembly begins at the last address displayed during a previous assembly session. At power-up the start address will be 0.

```
 >ASM
**** 680XX LINE ASSEMBLER ****
000000 >X
 >
```

ASM *<arg>*	Assembly begins at the specified address.

```
 >ASM <address>
**** 680XX LINE ASSEMBLER ****
000000 >END
 >
```

END

X

```
000000 >X
**** END OF LINE ASSEMBLY ****
 >
```

Exit line assembly.

Comments

All 680XX instructions can be entered from the line assembly mode. The instructions will be converted to machine code and loaded into memory at the address specified in the prompt.

The following pages describe the supported assembler directives.

ASSEMBLER DIRECTIVES

Command	Result
ORG	Sets program assembly origin:
END or X	Exits from line assembler to the command level:
DC	Defines constant byte, word, long word, or text string.
EQU	Defines/redefines local symbol (L0-L9):

ORG:

```
35F300 > ORG $4000
004000>
```

END or X:

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

DC:

```
002400 > DC.W#$8034,256, 1024
002400 8034 0100 0400
```

```
6590 >L3 EQU $7A44
6590 >
```

or if symbolic debug hardware is installed:

```
6590 > 'Unit EQU OFDEO
6590 >
```

SET

Defines/redefines symbol value (only valid with local label L0X L9X unless symbolic debug is installed).

L0,L1...L9

Prints value of local symbol:

```
756A >L3
756A >L3 EQU 7A44h
756A >
```

'symbol

Prints value of symbol. This is only valid if symbolic debug hardware is installed:

```
756A >'Unit
756A >'Unit EQU FDEOH
756A >
```

(continued)

RETURN

Disassembles one instruction at the current address:

```
000100>
000100 7A75  MOVEQ.L #$75,D5
```

*

Current line assembly address.

```
000130 > BRA.S *
```

MEMORY DISASSEMBLER

Command	Result
DIS *<range>*	Disassembles and displays the data in the specified range.
DIS *<address>*	Disassembles one page of memory beginning at a specified address.
DIS	Disassembles and displayss a page of memory beginning at the last address display during previous **DIS** command. At power-up this value is 0.

Comments

You should be familiar with Motorola 68000 assembly language programming and have the *16 Bit Microprocessor Users Manual* or the *MC68020 32 bit Microprocessor Users Manual* by Motorola.

The information presented here is an overview and should be used in conjunction with Motorola documentation.

(continued)

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. It is exited by typing any other character.

A page of data is defined by a parameter in the **SET** menu.

<space>	Continues disassembling one line at a time.
RETURN	Continues disassembling one page at a time.
any char except <space> or RETURN	Exits disassembly mode.

Memory Mode

Memory mode allows you to view and modify memory using a simple scrolling scheme. Enter memory mode by executing the **M** command. The current address and associated data are displayed. If the first character entered on a memory mode command line is a $\boxed{\text{RETURN}}$, the next address and its data are displayed. If a value is entered before the $\boxed{\text{RETURN}}$, that value is written to the current address before displaying the next address. A list of up to nine values separated by commas may be entered after a memory mode prompt. These data are stored to consecutive addresses.

The scroll direction is determined by two commands, **NXT** and **LST**. **NXT** (next) increments the address and **LST** (last) decrements the address. Entering either of these commands during run or pause mode sets the scroll direction and enters memory mode. The scroll direction can also be changed after you have already entered memory mode by executing the appropriate command. The scroll direction can be manually overridden at any time by using the period $\boxed{.}$ and comma $\boxed{,}$ keys. A period increments the address and 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.

ENTER MEMORY MODE

Command	Result
M [.B .W .L] *<address>*	Enters memory mode at *<address>*. The address and the data at that address are displayed preceding the prompt.
M [.B .W .L]	Enters memory mode at the last address examined in a previous memory mode session. The last address is stored in the MMP register (memory mode pointer). At power-up, this value is zero.
X	Exits memory mode.

Comments

Affects real-time operation when entered in run mode (see page 6-15).

Data displayed in memory mode can be in either byte, word, or long word lengths. Set byte mode (BYM), word mode (WDM), or long word mode (LWM) before entering memory mode. If you are in word mode and enter a byte of data, two bytes of data are written: the byte you entered, and a zero. If you are in byte mode and enter a word of data, the value is truncated, and only a byte is written.

The MMP register is modified if you scroll to a new address while in memory mode. When you exit memory mode, MMP reflects the last address examined.

When a RETURN is entered as the first character on a line, the address is incremented or decremented and the new address and data are displayed. On power-up, the default scroll mode is toward increasing memory addresses. To change the scrolling direction use the **NXT** (forward) and **LST** (backward) commands. These can be entered in memory mode. If they are entered in pause mode, the scroll mode is set and memory mode is entered at MMP.

The scroll mode can be overridden by using the period and comma keys. A . increments the address and a , decrements the address.

To modify data at a memory location, enter the data and press RETURN. The data are written to the current address and the next address and data are displayed.

Data can be entered quickly using a list. A list can contain up to nine values separated by commas. See example below.

Examples

Set the MMP and use the **NXT** command to enter memory mode. Change a word of data and verify.

```
>WDM; MMP=$FF000; NXT
$0FF000 $1234  >1122
$0FF001 $00FF  >,
$0FF000 $1122  >X
>
```

(continued)

Assume that address $1000 is the start of a data table and you want to write a short program to utilize that data.

Initialize the data using a list. Then invoke the line assembler using MMP as the start address (page 6-34).

```
 >M 1000
 $001000 $00  >0,1,2,3,4,5,6,7,8
 $001009 $00  >X
 >ASM MMP
****  680XX LINE ASSEMBLER  ****

1009 >                                  Enter your program here.
                                        Use 'X' or 'END' to exit
                                        the line assembler.
```

EXIT MEMORY MODE

Command	Result
X	Exits memory mode.

SCROLLING IN MEMORY MODE

Command	Result
$\boxed{\text{RETURN}}$	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 **BYT**, **WRD**, **LWM**, page 4-28).
LST	The $\boxed{\text{RETURN}}$ key now decrements addresses in memory mode.
NXT	The $\boxed{\text{RETURN}}$ key now increments (default mode) addresses in memory mode.
.	Increments the address in memory mode.
,	Decrements the address in memory mode.

Comments

The **NXT** and **LST** commands may be entered from either pause, run or memory mode. If entered from the run or pause mode the $\boxed{\text{RETURN}}$ 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.

MEMORY MODE POINTER

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

Comments

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

The default power-up value of the MMP register is zero. This register may be stored in EEPROM.

The memory mode pointer is modified if you change to a new address after entering memory mode. When you exit memory mode, the MMP reflects the last address examined.

As with any register, the MMP can be used as a parameter to another command.

(continued)

Examples

Set the MMP and verify.

```
 >MMP=$12330;MMP
$00012330
 >
```

Diagnostic Functions

The diagnostic functions (also called special functions, or SFs) are a group of utility routines and special tests. They are valuable for locating address, data, status or control line problems. There are two categories: RAM tests and scope loops. For a complete list see the **SF** command (page 6-50.)

RAM TESTS

These prewritten tests check that RAM is operating properly. They can be run on the target or Emulator overlay memory and may be executed in either byte or word mode. Byte or word mode must be specified prior to initiating the SF test.

SF 1 and 3 are modeled after a study by Abraham, Thatte, and Narir entitled *Efficient Algorithms for Testing Semiconductor Random-Access Memories* [IEEE Transaction on Computers, vol. c-27, no. 6 June 1978]. Refer to this publication for background information on these two diagnostics.

SCOPE LOOPS

Scope loops are diagnostic routines built into the Emulator firmware for use when troubleshooting with an oscilloscope. The uses for these special functions range from locating stuck address data, status or control lines, to generating signatures using signature analysis equipment.

The firmware is optimized so that the loops execute at maximum speed. This short cycle time allows the hardware engineer to review the timing of pertinent signals in the target system without using a storage oscilloscope. All of these routines must be terminated by resetting the Emulator. The scope loops can be executed in either byte or word mode.

CUSTOM DIAGNOSTICS

Special functions 40-49 allow you to execute up to ten user written diagnostics. In the event that the above mentioned RAM tests and scope loops do not provide the desired test for your equipment, these custom diagnostics will allow you to download, debug, and execute diagnostics of your own design. You may also read or write to any address or function code space in the target system and/or overlay memory with these diagnostics.

SPECIAL FUNCTIONS LIST

Command	Result
SF	Displays list of all available RAM tests, scope loops and miscellaneous tests.

Examples

```
>SF
SF 0,<RANGE><CR>                          SIMPLE RAM TEST, SINGLE PASS
SF 1,<RANGE><CR>                          COMPLETE RAM TEST, SINGLE PASS
SF 2,<RANGE><CR>                          SIMPLE RAM TEST, LOOPING
SF 3,<RANGE><CR>                          COMPLETE RAM TEST, LOOPING
SCOPE LOOPS:
SF 10, <ADDRESS>,<PATTERN><CR>            READ CONTINUOUSLY FROM ADDRESS
SF 11, <ADDRESS>,<DATA><CR>               WRITE CONTINUOUSLY TO ADDRESS
SF 12, <ADDRESS>,<PAT 1>,<PAT 2><CR>      WRITE ALTERNATE PATTERNS
SF 13, <ADDRESS>,<PATTERN><CR>            WRITE PATTERN, THEN ROTATE
SF 14, <ADDRESS>,<DATA><CR>               WRITE DATA, THEN READ
SF 15, <RANGE><CR>                        READ DATA OVER ENTIRE RANGE
SF 16, <ADDRESS><CR>                      WRITE INCREMENTING COUNT
SF 17, <CR>                               GENERATE RESET PULSES
MISCELLANEOUS:
SF 20 <CR>                                CRC CHECK OF EMULATOR FIRMWARE
SF 40-49 <CR>                             RUN USER PROGRAMS INTERNAL SPACE
CLK <CR>                                  DISPLAY TARGET CLOCK FREQUENCY
CRC <RANGE> <CR>                          CALCULATE CRC OF SPECIFIED RANGE
```

SIMPLE RAM TEST, SINGLE PASS

Command	Result
SF 0, *<range>*	Writes a test pattern to all locations within the specified range, then reads each location to verify the data. The following pattern sequence is used:

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

(continued)

Comments

All RAM tests default to WDM.

This command is valid in pause mode only.

If a location is read that doesn't match the test pattern, a failure is reported.

The address, correct data, and faulty data are displayed.

If no failure is detected, the following prompt is displayed:

```
            COMPLETE
```

This is a single pass test.

COMPLETE RAM TEST, SINGLE PASS

Command	Result
SF 1, *<range>*	Writes, then reads a test pattern to all locations in the specified range. Refer to *Efficient Algorithms for Test Semiconductor Random-Access Memories* mentioned in the introduction to Diagnostic Functions for the test pattern.

Comments

All RAM tests default to WDM.

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

```
MEMORY FAILURE: $00000800=$FFFF, NOT $0000; CODE=$06
```

An error is detected.

SIMPLE RAM TEST, LOOPING

Command	Result
SF 2, *<range>*	Writes a test pattern to all locations in *<range>*, then reads each location to verify the data. See SF 0 for test pattern. Each time the test is executed, the pass count is incremented and displayed on the screen.

Comments

All RAM tests default to WDM.

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
MEMORY FAILURE: $00000002=$00FE, NOT $00FF
PASS # 1
  .
  .
  .
  .
UNTIL RESET
```

You must reset the Emulator to terminate this test.

COMPLETE RAM TEST, LOOPING

Command	Result
SF 3, *<range>*	Writes a test pattern to all locations within *<range>*, then reads each location to verify the data. See SF 1 for test reference information.

Comments

All RAM tests default to WDM.

This commands 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, correct data, faulty data, and test sequence number are displayed.

```
 >SF 3, 0 TO 2
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
MEMORY FAILURE: $00000002=$0400, NOT $0000; CODE=$01
MEMORY FAILURE: $00000000=$0400, NOT $0000; CODE=$03
PASS #1
MEMORY FAILURE: $00000002=$0400, NOT $0000; CODE=$01
MEMORY FAILURE: $00000000=$0400, NOT $0000; CODE=$03
PASS #2
   .
   .
   .
UNTIL RESET
```

You must reset the Emulator to terminate this test.

Command	Result
SF [.B .W .L] 10<*address*>	Consecutively reads from the specified memory address using MMS as status space register.

Comments

This command is valid in pause mode only.

You must reset the Emulator to terminate this test.

Examples

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

POKES INTO THE TARGET SYSTEM

Command	Result
SF[.B.W.L]11,*<address>,<data>*	Consecutively writes the user defined data pattern to the specified memory address using MMS as status space register.

Comments

This command is valid in pause mode only.

You must reset the Emulator to terminate this test.

Examples

```
>SF.B 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
```

WRITE ALTERNATE PATTERNS

Command	Result
SF [.B .W .L] 12 *<address>*, *<pattern 1>*, *<pattern 2>*	Consecutively writes the user defined data pattern to the specified memory address using MMS as status space register followed by the complement of that data pattern to the same address.

Comments

This command is valid in pause mode only.

You must reset the Emulator to terminate this test.

Examples

```
  >SF 12 $FF0000,$AA,55
 YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```

The following data pattern is written to address $FF0000.

```
AA
55
AA
55
 .
 .
 .
 .
until reset
```

WRITE PATTERN THEN ROTATE

Command	Result
SF [.B .W. L.] 13, *<address>*, *<pattern>*	Consecutively writes the data pattern to the specified memory address using MMS as status space register, rotates the pattern 1 bit to the left and writes to the same address.

Comments

This command is valid in pause mode only.

You must reset the Emulator to terminate this test.

Examples

```
  >SF 13,1000,05
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```

The following data pattern is written to address 10:

```
05
0A
14
28
50
A0
41
82
```

WRITE DATA THEN READ

Command	Result
SF[.B.W.L]14,*<address>*,*<data>*	Consecutively writes the specified data pattern to the specified memory address using MMS as status space register, then reads from that same address.

Comments

This command is valid in pause mode only.

You must reset the Emulator to terminate this test.

Examples

```
>SF 14, 100,$FFFF
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```

READ DATA OVER AN ENTIRE RANGE

Command	Result
SF [.B .W .L] 15, *<range>*	Consecutively reads from the specified memory address range using MMS as status space register.

Comments

This command is valid in pause mode only.

You must reset the Emulator to terminate this test.

Examples

```
>SF 15, 10 TO 20
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```

WRITE INCREMENTING COUNT

Command	Result
SF [.B .W .L] 16, *<address>*	Consecutively writes a constantly incrementing value to the specified memory address using MMS as status space register.

Comments

This command is valid in pause mode only.

You must reset the Emulator to terminate this test.

Examples

```
>SF 16, 10 $FFFF
YOU MUST RESET ME TO TERMINATE THIS FUNCTION
```

GENERATE RESET PULSES

Command	Result
SF 17	The Emulator sends continuous reset pulses to the target system.

Comments

You must reset the Emulator to terminate this test. CTRL Z is the factory default reset.

CYCLIC REDUNDANCY CHECK

Command	Result
SF 20	A CRC is calculated on the ES1800 internal PROM that contains the Emulator firmware.

Comments

This commands is valid in pause mode only.

This is an Emulator self-test.

If a failure is detected, a CRC error is displayed.

This is a single pass routine.

Command	Result
SF *<40-49>*	Executes custom diagnostic.

SF 40	Execute diagnostic at $7000
SF 41	Execute diagnostic at $7004
SF 42	Execute diagnostic at $7008
SF 43	Execute diagnostic at $700C
SF 44	Execute diagnostic at $7010
SF 45	Execute diagnostic at $7014
SF 46	Execute diagnostic at $7018
SF 47	Execute diagnostic at $701C
SF 48	Execute diagnostic at $7020
SF 49	Execute diagnostic at $7024

Comments

You may determine that the built-in diagnostics and scope loops do not provide the proper test for your equipment; for that reason custom diagnostics are provided to allow you to download, debug and execute diagnostics of your own design.

Custom diagnostics can access or modify parameters stored in GD0-7 and may also read or write to any function code space in the target system and/or overlay memory. Special functions 40-49 provide the means of executing up to ten custom diagnostics.

To make a group of diagnostics, you must first create a table containing up to ten long-branches that starts at address 7000 Hex. The **BRA.L** must be present or the diagnostic will not be executed. The **BRA.L** will vector to

(continued)

the start of the user-written routine. (This routine must be located in internal RAM, in the range of 7000-78FF (3K) unless a **JMP** command vectors control to overlay memory that is mapped at an address above 80000 Hex.) To return control to the emulator, the routine will terminate with an **RTS** instruction.

PEEKING AND POKING INTO THE TARGET SYSTEM (68010/68020)

There are four special function codes reserved for the purpose of accessing the target and/or overlay from a custom diagnostic. These special function codes can be generated by the use of the **MOVES** instruction. (Refer to the Motorola *MC68020 32 Bit Microprocessor Users Manual.*) The function code seen by the target system is not the function code that the **MOVES** instruction generates. Instead, the target function code is picked up from a register in the internal memory space.

The register containing the substitution function code is six bits wide and contains two function code values:

- The lowest three bits of this register (0-2) are referred to as X
- The upper three bits (3-5) are referred to as Y

By storing the proper codes in the SFC and DFC registers, it is possible to read from one function code space, controlled by X, and write to another function code space, controlled by Y.

Two of the four special function codes are used to access either overlay memory exclusively (OVO) or the target exclusively (TGO). The other two codes access the target and also overlay memory, when it is mapped.

The following table lists three columns that show how the three special function codes are used. The left column shows the function code that originates at the SFC or DFC register inside the CPU; the middle column shows which substitution register is used to generate the space code for the target system, and to enable overlay memory; the last column indicates the two special function codes that are used to enable the target only (TGO) and the overlay only (OVO).

(continued)

Use of special function codes:

Origin		
0	X	TGO
1	X	
3	Y	
4	Y	OVO

Note that before a custom diagnostic can peek or poke to the target, the X and Y registers must be initialized. In most cases only the X bits need to be initialized since typically one function code space is all that's needed. This register is located at address $3F63 in the internal memory space:

7	6	5	4	3	2	1	0
n/c	n/c	Y2	Y1	Y0	X2	X1	X0

Bits 6 and 7 are not used. Data stored to this register can also be read, so read/modify/write instructions like AND and OR will work.

After the X and Y registers are initialized the SFC and DFC registers can be loaded. These registers would typically both be loaded with a $1 to cause the X register to be substituted, and to enable both the target system and the overlay memory when it is mapped.

PEEKING AND POKING INTO THE TARGET SYSTEM (68000/68008)

Unlike the 68010, the 68000 and 68008 processors do not support the **MOVES** instruction. Consequently, the 68000/08 Emulators access the target differently.

When the 68000/08 Emulator is not in emulation, the function code seen by the target is in a register in the internal memory space.

The register containing the substitution function code is six bits wide and contains the following:

- The lowest three bits (0-2) are referred to as X.
- Bit 3 is referred to as OVO (overlay only).
- Bit 4 is referred to as TGO (target only).
- Bit 5 is not used.

In order to read and write to different function code spaces, X must contain the function code space before accessing the target.

Normally during a peek or poke, if the space to be accessed is overlaid, the target and overlay memory are accessed simultaneously. However, if accesses to that space in the target cause a bus error, you may set the OVO bit and only the overlay will be accessed. Likewise, to restrict accesses to the target only, set the TGO bit.

To access the target (or overlay memory, if it mapped), the diagnostic program must generate a function code 1 (user data) during a target read or write. Then the function code in the substitution register (located at $3F63 in the internal memory space) will be seen by the target.

(continued)

7	6	5	4	3	2	1	0
n/c	n/c	n/c	TGO	OVO	X2	X1	X0

Bits 5, 6 and 7 are not used. Data stored to this register can also be read, so read/modify/write instructions like AND and OR will work.

DOWNLOAD CUSTOM DIAGNOSTICS

A user written diagnostic assembled on a host system may be downloaded to the Emulator internal RAM.

The diagnostics to be downloaded must be assembled first.

The introspective mode software switch setting must be enabled (**ON IM**) before executing a download sequence.

After enabling the **IM** switch the Emulator is considered a target system and the **DNL** command (page 5-38) may be used in its typical manner for downloading to a target system.

Custom diagnostics will not access your target system when in the introspective mode.

To terminate the download process, set the introspective mode software switch to off (**OFF IM**). The Emulator is taken out of introspective mode and no longer is considered a target system.

The system reset character (CTRL Z) will also take the Emulator out of the introspective mode. (CTRL Z) is the system default reset character, and **SET** parameter #2 is the user defined reset character (see page 5-3).

The emulator has 3K of memory space allocated for custom diagnostics. If this is not sufficient then overlay memory may be used for additional space. To set up overlay memory for custom diagnostics:

- Map overlay memory only above $8000.
- Map into a space that is not already used by the target system. This will assure that mapping is not done on top of already mapped space.
- Map as read-only. This will prevent overlay memory from being modified by a custom diagnostic.
- The **OVS** (page 5-67) must be set to a non-zero value. This will cause overlay memory to provide a DTACK (DSACK for the 68020) to the microprocessor.

(continued)

PASSING PARAMETERS TO CUSTOM DIAGNOSTICS

Many times when writing diagnostics it may be necessary to pass parameters to those routines. The SF 40-49 commands do not take parameters, so you should store the parameters into one of the eight general range registers (GR0-7) or into one of the eight general data registers (GD0-7). Your custom diagnostic may then pick up the data or store results at the following locations:

68000/08/10

	BEGINNING RANGE	ENDING RANGE
GR0	$3100 - $3103	$3104 - $3107
GR1	$3108 - $310B	$310C - $310F
GR2	$3110 - $3113	$3114 - $3117
GR3	$3118 - $311B	$311C - $311F
GR4	$3120 - $3123	$3124 - $3127
GR5	$3128 - $312B	$312C - $312F
GR6	$3130 - $3133	$3134 - $3137
GR7	$3138 - $313B	$313C - $313F

68020 ONLY

GR0	$3200 - $3203	$3204 - $3207
GR1	$3208 - $320B	$320C - $320F
GR2	$3210 - $3213	$3214 - $3217
GR3	$3218 - $321B	$321C - $321F
GR4	$3220 - $3223	$3224 - $3227
GR5	$3228 - $322B	$322C - $322F
GR6	$3230 - $3233	$3234 - $3237
GR7	$3238 - $323B	$323C - $323F

68000/08/10

	DATA	DON'T CARE DATA
GD0	$3140 - $3143	$3144 - $3147
GD1	$3148 - $314B	$314C - $314F
GD2	$3150 - $3153	$3154 - $3157
GD3	$3158 - $315B	$315C - $315F
GD4	$3160 - $3163	$3164 - $3167
GD5	$3168 - $316B	$316C - $316F
GD6	$3170 - $3173	$3174 - $3177
GD7	$3178 - $317B	$317C - $317F

68020 ONLY

	DATA	DON'T CARE DATA
GD0	$3240 - $3243	$3244 - $3247
GD1	$3248 - $324B	$324C - $324F
GD2	$3250 - $3253	$3254 - $3257
GD3	$3258 - $325B	$325C - $325F
GD4	$3260 - $3263	$3264 - $3267
GD5	$3268 - $326B	$326C - $326F
GD6	$3270 - $3273	$3274 - $3277
GD7	$3278 - $327B	$327C - $327F

GR registers are 32 bits wide while ranges are only valid for the 24 bits. A custom diagnostic that required a range parameter and a don't care and returned a 32-bit data parameter might look like:

```
>GR4=$1000 LEN $40;GD4=$CXFF;SF  $44;GD0
```

The range parameter goes into GR4; the Don't Care bit (X) goes into GD4; then the user diagnostic is called; and finally, the result is displayed as the content of the register GD0.

(continued)

DEBUGGING CUSTOM DIAGNOSTICS

A custom diagnostic may be set up to be debugged similar to debugging a program in target memory space.

After a custom diagnostic is downloaded to the Emulator internal RAM it may be debugged by the following procedures:

- Enable introspective mode with the **IM** software switch, page 5-21.
- Set the SSP register to point to an area of memory that will provide a stack. This stack should be the top of the user RAM area (7C00 hex) in overlay memory that is mapped as read/write.

If a **RTS** is encountered while single stepping through a routine the PC register will be undefined.

READ TARGET SYSTEM CLOCK

Command	Result
CLK	Reads the target system clock and displays the value in KHz, accurate to 1-2 KHz.

Examples

```
  >CLK
CLOCK FREQUENCY = #2001 KHZ
  >
```

TARGET CYCLIC REDUNDANCY CHECK

Command	Result
CRC *<range>*	The system calculates a cyclic redundancy check on all addresses in *<range>*.

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.

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 your program and do the **CRC** over the range again. If the values do not match, data are being corrupted. The Event Monitor System can be setup to catch writes to the data base.

The **CRC** algorithm is based on the polynomial $X^{16}+X^{15}+X^2+1$.

DISPLAY STATUS OF SEVERAL STATUS LINES

Command	Result
BUS	Displays the bus status.

Comments

Your console will display bus status:

HLT	Halt
IPL	Interrupt Priority Level
RST	Reset
VCC	Power
BER	Bus Error

Examples

The system default status is:

```
HLT   IPL   RST   VCC   BER
 0     0     0     0     0
```

0 indicates an inactive condition. 1 indicates an active condition. 0-7 in the IPL column represents the interrupt priority level encoded on the IPL lines.

SECTION 7

Table of Contents

68000/08/10 Event Monitor System

68000/08/10 EVENT MONITOR SYSTEM

Overview

The ES1800 Event Monitor System (EMS) provides extremely flexible system and breakpoint control, enabling the user to isolate or break on any predefined series of events and then perform various actions. The user controls and monitors the target by entering commands that define events as logical combinations of address, data, status, count limit, and optional logic state probe inputs. When an event is detected, the ES1800 can break emulation, trace specific sequences, count events, execute user supplied target routines, and trigger a TTL output.

The ES1800 Event Monitor System is based on a 16 bit environment. The 68020 microprocessor consists of a 32 bit architecture. To incorporate the 68020 into the ES1800 chassis, some enhancements were introduced into the emulator's operating system to facilitate selective tracing. 68020 tracing requirements are described in section 8.

NOTE: Use of the Dynamic Trace feature is not possible with the Event Monitor System. Refer to the Dynamic Trace Capture Enable command in section 5 for more information.

WHEN/THEN control statements define events and their corresponding actions. There can be several actions for any event. The system only recognizes the first three letters of any word in a control statement (e.g., WHEN=WHE; THEN=THE). There can be many control statements in effect at any time. The Event Monitor System can also switch groups to allow for sequencing events. There are four event groups available and the control statements and comparator values for any group are independent of those in other groups.

Event Monitor System control statements can be entered while in run mode. The event comparator values can also be modified during run mode. However, these new statements and values will not go into effect until you stop and restart run mode.

The ES1800 Event Monitor System monitors target information at the bus cycle level; i.e., every read or write cycle that the microprocessor executes. The EMS system detects every signal that can affect the target system. It can also monitor inputs from the logic state analyzer probe.

The Event Monitor System essentially takes a picture of the Motorola 680XX microprocessor's signals at the end of every bus cycle (refer to Motorola manual, *16 Bit Microprocessor Users Manual*). The information that is recorded into trace memory is the same information that the EMS is monitoring.

The Event Monitor System will only break on the address information that was traced. The breakpoint comparators only look at the addresses output by the microprocessor.

The basic Event Monitoring System control statement is of the form:

[*Group*] **WHE**[*N*] <*event*> **THE**[*N*] <*action*>

Notice that the ESL command processor needs only the first three letters of the symbol.

COMPARATOR REGISTERS

An event is based on the value of a comparator register or a combination of comparator registers. Comparators can be combined with AND, OR, or NOT operators. The action that follows an event may be a single command or multiple commands separated by commas.

There are eight comparator registers for each of the four event groups. These event registers are listed in the table on the following page. The address comparators are used to detect discrete addresses or addresses inside or outside a specified range. The data comparators can detect specific data patterns and can ignore specified bit positions. The status comparators monitor all of the status signals from the microprocessor as well as some generated by the ES1800. The status comparators can also ignore bit positions. The count limit register can be used to detect when an event has occurred more than a specified number of times. The logic state analyzer register can detect bit patterns in the inputs from the logic state probe.

The following table describes the available event comparator registers.

Register		Size	Name by Group			
Description	Type	(bits)	1	2	3	4
Address 1	Range,Integer	24	AC1 or AC1.1	AC1.2	AC1.3	AC1.4
Address 2	Range,Integer	24	AC2 or AC2.1	AC2.2	AC2.3	AC2.4
Data 1	Don't Care,Integer	16	DC1 or DC1.1	DC1.2	DC1.3	DC1.4
Data 2	Don't Care,Integer	16	DC2 or DC2.1	DC2.2	DC2.3	DC2.4
Status 1	Don't Care,Integer	16	S1 or S1.1	S1.2	S1.3	S1.4
Status 2	Don't Care,Integer	16	S2 or S2.1	S2.2	S2.3	S2.4
LSA	Don't Care,Integer	16	LSA or LSA.1	LSA.2	LSA.3	LSA.4
Count	Integer	16	CL or CL.1	CL.2	CL.3	CL.4

ADDRESS COMPARATORS

Address comparators may be assigned integer values or range values. Ranges may be either internal (IRA) or external (XRA). If a range is specified without IRA or XRA operators, the default range type will be IRA. The following are examples of valid address comparator assignments.

```
>AC1=2000
>AC2=1000 LEN 20
>AC2.2=XRA 1100 TO 1250
>AC1.4 = IRA $FF006 LEN $FF
>AC1.1 = A1
>AC2='Symbol
>AC1 =PC + 200
>AC1.2 = !AC1.4
```

DATA AND LSA COMPARATORS

The data comparators monitor the data bus for specified patterns. The LSA comparators monitor the input pulses from the logic state probe.

Data and LSA comparators may be assigned integer values or don't care values. Don't care values may be assigned in two ways. The first is to specify the value followed by the don't care mask; the second is to specify the value using $\boxed{\text{X}}$ in the don't care positions. The following are examples of valid data and LSA comparator assignments.

```
>DC1=237F
>LSA=5300 DC $FF
>LSA.3 = 53XX
>LSA = %110101 DC $FF00
>DC2.2 = 42 DC %101
>DC2 = GD0 + $F
>DC1.4 = @'data table + 56
```

STATUS COMPARATORS

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

STATUS MNEMONICS

BER	bus error
VM	valid memory address
VP	valid peripheral address
IP(0-7)	interrupt levels 0-7
SP	supervisor program
SD	supervisor data
UP	user program
CPU	CPU space
SC(0-7)*	numeric names for all 8 space codes*
TAR	target system access
OVL	overlay memory access
RD	read access
WR	write access
BYT	byte access
WRD	word access

*SCO, SC3, SC4, used only by 68010 and 68020.

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 2-9. Mnemonics are combined using an addition operator ($\boxed{+}$) as a Boolean AND.

```
                         STATUS MNEMONIC TABLE

1    2      3     4      5       6      7      8     9
S1 = BYT  + RD  + TAR  + SCO   + BER  + IPO  + VM  + VP
S2   WRD    WR    OVL    SC1/UD         IP1
                         SC2/UP         IP2
                         SC3            IP3
                         SC4            IP4
                         SC5/SD         IP5
                         SC6/SP         IP6
                         SC7/CPU        IP7
```

Some examples of status comparator assignments:

```
>S1=BYT
>S2=IP7+SD
>S1.2=OVL+BER
```

In the following example, all bits except bit 1 (one) have been masked.

```
S1 = RD
S1
$00000002 DC $0000FFFD
```

In the next example, all bits except bits 1, 2, 9, 10 and 11 are masked. These bits are enabled (1s or 0s) and should be considered.

```
S1 = IP7 + RD + TAR
$00000E06 DC 0000FIF9
```

```
                        STATUS COMPARATOR BREAKDOWN

14    13    12    11    10    9   8   7   6   5   4    3   2   1   0
 |     |     |     \     |    /   |       \   |   /        |   |   |
1=VM  1=VP              0=IP0     1=BER     0=SC0        0=OVL  |   |
                        1=IP1               1=SC1/UD     1=TAR  |   |
                        2=IP2               2=SC2/UP        0=WR    |
                        3=IP3               3=SC3           1=RD    |
                        4=IP4               4=SC4                0=WRD
                        5=IP5               5=SC5/SD             1=BYT
                        6=IP6               6=SC6/SP
                        7=IP7               7=SC7/CPU

                     STATUS COMPARATOR BIT REPRESENTATION

VM=00004000 DC BFFF
VP=00002000 DC DFFF
IP0=00000000 DC FF1F
IP1=00000200 DC F1FF
IP2=00000400 DC F1FF
IP3=00000600 DC F1FF
IP4=00000800 DC F1FF
IP5=00000A00 DC F1FF
IP6=00000C00 DC F1FF
IP7=00000E00 DC F1FF
BER=00000100 DC FEFF
SC0=00000000 DC FF8F
SC1/UD=00000010 DC FF8F
SC2/UP=00000020 DC FF8F
SC3=00000030 DC FF8F
SC4=00000040 DC FF8F
SC5/SD=00000050 DC FF8F
SC6/SP=00000060 DC FF8F
SC7/CPU=00000070 DC FF8F
TAR=00000004 DC FFFB
OVL=00000000 DC FFFB
RD=00000002 DC FFFD
WR=00000000 DC FFFD
BYT=00000001 DC FFFE
WRD=00000000 DC FFFE
```

When you display the value of the status comparators, you will see don't care values rather than the mnemonics you originally assigned. The status comparator breakdown table is provided to aid you in decoding the numbers back into mnemonics.

The don't care mask is the value to the right of DC. A zero in a mask bit position enables the status bit in the same position on the left side of the DC, and "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 0s 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 status comparator breakdown 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
$00000E06 DC 0000F1F9
```

All bits except bits 1, 2, 9, 10 and 11 are masked. Bits 9, 10, and 11 are enabled and a 7 in these three bits of the status value indicates IP7 was entered.

Bit 1 is enabled and there is a 1 in bit 1. So RD was entered.

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

Therefore, the original input was:

```
>S1=IP7+RD+TAR
```

COUNT LIMIT COMPARATOR

The count limit comparator CL, is used to detect when events have occurred a certain number of times. The CL value for group 1 is loaded into a hardware counter which is decremented whenever the action **CNT** is executed (see the sections on Defining Events and 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.

Defining Events

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

EVENT

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

The NOT operator is used to reverse the sense of the comparator output. NOT has higher precedence than either of the conjunctives. `WHEN AC1 AND NOT DC1 THEN BRK` means break whenever any data pattern other than that in DC1 occurs on the bus along with AC1.

AND and OR can be used where needed to form more restrictive event definitions. AND terms have higher precedence than OR terms. `AC1 AND DC1 OR DC2` is the same as `AC1 AND DC1` in one statement and `DC2` in another. If you are looking for two different data values at an address, use `AC1 AND DC1 OR AC1 AND DC2`.

The OR operator is evaluated left to right and is useful for simple comparator combinations. For complex event specifications, OR combinations can be replaced with separate EMS control statements for clarity. `AC1 AND S1 OR AC2 AND S2` is the same as `AC1 AND S1` and `AC2 AND S2`.

Defining Action Lists

The action list in a WHEN/THEN statement defines what the Emulator does when an event is detected. Actions are specified in an action list separated by commas. The action list may have one or more actions defined.

Examples

<group> WHEN *<event>* THEN *<action>,<action>, ... ,<action>*

The following table lists all possible actions.

Event Monitor System Actions

Action	Description
BRK	Break emulation
CNT	Count Bus Cycle
FSI	Force Special Interrupt
GRO n	Change Event Group
RCT	Load Count Value
TGR	Output Trigger Signal
TOC	Toggle Count State
TOT	Toggle Trace State
TRC	Trace Bus Cycle

The **TRC** and **TOT** actions are described in the Tracing Events section. The **CNT**, **RCT**, and **TOC** actions are described in the Counting Events section. The **FSI** action is described in the Special Interrupt section. The **GRO** action is described in the Changing Event Groups section. The **TGR** action is described in the Trigger Signal section. The **BRK** is described in the Breaking Emulation section.

The EMS will always resolve conflicting EMS control statements. The **TOC** action in the first statement will be changed to **CNT**.

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

DISPLAY EVENT SPECIFICATIONS

Command	Result
DES	Displays all of the WHEN/THEN statements currently active from all groups.
DES *<group number>*	Displays all of the WHEN/THEN statements and the comparator values for the specified group.

Examples

Display the statements and comparators for groups 1 and 2.

```
 >DES 1;DES 2
1 WHEN AC1 THEN BRK
AC1.1 = $007632
AC2.1 = $000000
DC1.1 = $0000
DC2.1 = $0000
S1 .1 = $0000
S2 .1 = $0000
LSA.1 = $0000
CL .1  = $0000

2 WHEN S1 AND DC1 THEN CNT,TRC
2 WHEN CL 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
CL .2 = $0010
```

CLEAR WHEN/THEN STATEMENTS

Command	Result
CES	Clears all of the WHEN/THEN statements currently active.
CES *<group number>*	Clears all of the WHEN/THEN statements for the specified group.

Comments

The comparator values are not affected by the **CES** command.

Breaking Emulation

The **BRK** action stops emulation, returning the system to pause mode. When a break event is detected and emulation has been broken, the current PC and event group is displayed on the terminal. Emulation begins at the values displayed if the registers are not altered and you run or step following a break. When entering emulation, the Event Monitor System always begins looking for events specified in group 1.

Breakpoints stop program execution at specific times. After a break you can disassemble the trace memory, look at the LSA bits in the raw trace, check the CPU register values, or begin stepping through your code.

Breakpoint actions may be enabled or disabled by selecting the appropriate run commands. If you enter emulation with the **RBK** or **RBV** run commands, breakpoints are enabled. If you enter emulation with the **RUN** or **RNV** commands, breakpoints are disabled, even if there are event statements specifying the **BRK** action. If emulation is entered with breakpoints disabled, you can enable them while running by entering the **RBK** command. If you enter emulation with breakpoints enabled, you can disable them while running by entering the **RUN** command. The **RNV** and **RBV** commands are not allowed during emulation. These commands load the reset vectors, which cannot be done during emulation.

Emulation may be halted using the **STP** command. The reset character (Default is CTRL Z. May be changed through the **Set** command, option 2) also breaks emulation.

Examples

Break when address $3000 is accessed.

```
  >AC1=3000
  >WHEN AC1 THEN BRK
  >RBK
R>
```

(continued)

Trace only accesses between 1000 and 113C. Break after ten accesses to this address range.

```
 >AC1=1000 to 113C
 >CL =#10
 >WHEN AC1 THEN CNT,TRC
 >WHEN CL THEN BRK
 >RBV
R>
```

Break when 55AA is written to port A.

```
 >AC1='PORT_A
 >DC1=55AA
 >S1=WR
 >WHEN AC1 AND DC1 AND S1 THEN BRK
 >RBK
R>
```

The 68008 uses an 8 bit data field. The emulator traces this information in a 16 bit field where the high and low positions are dependent on it being an even or odd address.

To break on an 8 bit data field use the following example:

```
DC1 = 55XX
DC2 = 0XX55
WHEN DC1 OR DC2 THEN BRK
```

Tracing Events

```
        Events:

                TRC
                TOT
```

The Event Monitor System can be set up to trace bus cycles selectively. If all of the conditions specified in the event portion of the WHEN/THEN clause are satisfied, the trace action, **TRC**, causes the specified bus cycle to be recorded into the trace memory.

The toggle trace, **TOT**, provides a way to turn tracing on and off. When a **TOT** event is detected, the trace is toggled to the opposite state, either on or off. You can specify a single event that starts and stops trace each time it is detected or specify any number of events that toggle trace on and off.

If there are no event actions that specify **TRC** or **TOT**, all bus cycles are traced. If there is a **TRC** event, only qualified bus cycles are traced. If there is a **TOT** event, trace will be off until the **TOT** is detected, then all bus cycles are traced until encountering another **TOT** event.

(continued)

This table describes the trace conditions immediately before and immediately after a group change.

	New Group		
Previous Group	Nothing Specified	TRC	TOT
Nothing Specified	Trace all cycles	Trace only qual'd cycles	Trace nothing until first TOT
TRC	Trace all cycles	Trace only qual'd cycles	Trace nothing until first TOT
TOT (Not tracing)	Trace all cycles	Trace only qual'd cycles	Trace nothing until first TOT
TOT (Tracing)	Trace all cycles	Trace only qual'd cycles	Trace nothing until first TOT

This table describes initial trace conditions.

Action Specified	Trace Condition
Nothing	Trace all cycles
TRC	Trace only Qual'd TRC events
TOT	Trace nothing until TOT event

Examples

Trace only a specific subroutine. Break at the end of the routine.

```
  >AC1='Sub_start to 'Sub_end
  >AC2='Sub_end
  >WHEN AC1 THEN TRC
  >WHEN AC2 THEN BRK
  >RBK
 R>
```

The **WHEN LSA THEN TOT** command statement is used to turn on a trace when a specific hardware event occurs and turn off when the activity is terminated.

Two event groups are required to specify special on and off points.

1. Identify the logical state analyzer addresses:

   ```
   LSA.1 = $0000 DC $FFFE
   LSA.2 = $0001 DC $FFFE
   ```

   This sets up the LSA comparator to a specific address.

2. To set up a WHEN/THEN statement to monitor a specific activity outside the microprocessor:

   ```
   WHEN LSA THEN TOT, GRO 2
   WHEN LSA.2 THEN BRK
   ```

   The system 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 group 2 to switch groups. In group 2 all bus cycles are traced until LSA pod bit 0 goes high. Then, emulation is broken.

(continued)

Counting Events

```
        Registers:
                                            Value Type - 16 bit integer
                CL
                CL.1
                CL.2
                CL.3
                CL.4

                CL=<EXP>
                CL<.group>=<EXP>

        Events:

                CNT
                RCT
                TOC
```

Events can be defined to count bus cycles selectively. There is one hardware counter and there are four count registers, one register for each group. Each group has its own count limit register (CL). The Event Monitor System has only one counter. The hardware counter is automatically loaded with the count limit register for group 1 when entering run mode.

The count, **CNT**, action decrements the hardware counter. When the count reaches zero, the CL event becomes true. If all other conditions specified in the WHEN/THEN clause are satisfied, the appropriate action is taken.

Whenever the reset count, **RCT**, action is specified, the count comparator value for the specified group is loaded into the hardware counter. When switching groups, the current value of the hardware counter is passed along as a global count value unless a **RCT** action is specified in the same list of events that causes the group switch.

The toggle count, **TOC**, command provides a way to turn counting on and off. When a **TOC** event is detected, the count is toggled to the opposite state, either on or off. You can specify an event that starts and stops the counter each time it is detected or specify any number of events that toggle the counter on and off.

The current value of the counter cannot be read. You can only detect when you have reached a limit.

This table describes the count conditions immediately before an after a group change.

	New Group		
Previous Group	Nothing Specified	CNT	TOC
Nothing Specified	No cycles counted	Count only qual'd cycles	Count nothing until first TOC
CNT	No cycles counted	Count only qual'd cycles	Count nothing until first TOC
TOC (Not counting)	No cycles counted	Count only qual'd cycles	Count nothing until first TOC
TOC (Tracing)	No cycles counted	Count only qual'd cycles	Count all until first TOC

(continued)

This table describes initial count conditions (always group 1).

Action Specified	Count Condition
Nothing	Count nothing
CNT	Count only Qual'd CNT events
TOC	Count nothing until TOC event

Examples

Count the times that the specified data are written to a specific address. Break if the data are written 20 times.

```
  >CL=#20
  >S1=WR
  >AC1=4020; DC1=$XXF3
  >WHEN AC1 AND DC1 AND S1 THEN CNT
  >WHEN CL THEN BRK
  >RBK
R>
```

To cause a break to occur after $400 writes have lapsed:

```
S1 = WR
CL = 400
WHEN S1 THEN CNT
WHEN CL THEN BRK
```

Look for a read from a specific I/O port. After it is found go to group 2, load the group 2 counter register value into the hardware counter and set a group 2 address comparator to count every bus cycle (all addresses). Break after 100 bus cycles.

```
 >AC1='IOport
 >S1=RD
 >WHEN AC1 AND S1 THEN GRO 2, RCT
 >CL.2=#100
 >AC1.2=0 TO -1
 >2 WHEN AC1 THEN CNT
 >2 WHEN CL THEN BRK
 >RBK
R>
```

(continued)

Trigger Signal

The trigger signal is an output that is available from the BNC connector labelled TRIG on the back panel of the ES1800 chassis and from pin 19 of the optional LSA pod. When a **TGR** event is detected, the trigger signal is asserted, and remains so for the duration of the specified bus cycle. If a trigger event is specified for more than one consecutive bus cycle, the signal stays high for the duration of the consecutive bus cycles.

The trigger signal can be used as a pulse output for triggering other diagnostic equipment. It can also be used in conjunction with a counter/timer for timing subroutines.

Examples

Trigger a scope when reading data from a UART.

```
>AC1='DATA_PORT
>S1=RD
>WHEN AC1 AND S1 THEN TGR
```

Determine the duration of a subroutine using the trigger pulse. The trigger pulse can be the input to a counter/timer or a scope. The duration of the subroutine can be determined from the pulse width displayed on the scope or the counter/timer readout.

```
 >AC1=2500                 Start of subroutine
 >AC1.2=AC1+38E            End of subroutine
 >DC1.2=XXXX               Detect any data pattern

 >WHEN AC1 THEN TGR, GRO 2  Go to group 2 when subroutine is entered
 >2 WHEN DC1 THEN TGR       Trigger during all cycles while in group 2
 >2 WHEN AC1 THEN GRO 1     Go back to group 1 when last instruction
 >RUN                       in subroutine is executed.
R>
```

Special Interrupts

```
Registers:

        SIA                     Value Type - 32 Bit Integer

Events:

        FSI
```

The force special interrupt action, **FSI**, provides a way to jump to a specified address when a specific event is detected.

The special interrupt address register, SIA, should be set prior to entering run mode if you are using the **FSI** event. It defines the address your program vectors to when the **FSI** is executed.

When an **FSI** event is detected, an `FSI ACTIVE` message is displayed on the screen. You may also see some unusual cycles in the trace memory at the address where the **FSI** occurred. These are internal cycles that get traced as the execution address is changed. They are not purged from the trace memory because of the need for speed when executing an **FSI**.

The **FSI** event can be used as a fast way to make a patch to your code. It is also used to write soft shutdown routines for machinery that cannot be halted using a simple breakpoint.

The **FSI** routine residing at the SIA address should terminate with a return from exception **(RTE)** instruction. Execution will resume 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 **RTE** instruction.

(continued)

Examples

Make a patch using overlay memory.

```
 >MAP 1000
 >AC1=8F36
 >WHEN AC1 THEN FSI
 >SIA=1000
 >ASM SIA                                 Single line assembler - patch code
     .                                    can be assembled here.
     .
     .
 >RUN
R>
```

Assume the program needs to break at a certain address, but the machine cannot be turned off until a soft shutdown routine is executed. Set SIA to the address of the soft shutdown routine. Use an **FSI** action at the break address, then set a breakpoint at the end of the soft shutdown routine.

```
 >SIA='SHUT_down
 >AC1=$7F4E2
 >AC2='SHUT_down + 4E
 >WHEN AC1 THEN FSI
 >WHEN AC2 THEN BRK
 >RBK
R>
```

Changing Event Groups

The four event groups provide a way to detect sequential events. When emulation is entered, event monitoring always begins in group 1. The example below describes a common use of the EMS group structure.

You may wish 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, you would 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
>'Module_B =8750 LEN 408
>'Sub_X =8934 LEN 56

>AC1='Module_A
>AC2='Module_B

>WHE AC1 OR AC2 THE GRO 2

>AC1.2='Sub_X

>2 WHEN AC1 THE TRC
>2 WHE NOT AC1 THE GRO 1
```

(continued)

The **TRC/TOT** and **CNT/TOC** actions interact in a specific way when event groups are switched. The following state transition tables describe the actions taken when each of the different event combinations are specified.

Previous Group	**New Group**		
	Nothing Specified	*TRC*	*TOT*
Nothing specified	Trace all cycles	Trace only qualified cycles	No trace until first TOT
TRC	Trace all cycles	Trace only qualified cycles	No trace until first TOT
TOT OFF (not tracing)	Trace all cycles	Trace only qualified cycles	No trace until first TOT
TOT ON (tracing)	Trace all cycles	Trace only qualified cycles	No trace until first TOT

Previous Group	**New Group**		
	Nothing Specified	*CNT*	*TOC*
Nothing specified	No cycles counted	Count only qualified cycles	No count until first TOC
CNT	No cycles counted	Count only qualified cycles	No count until first TOC
TOC OFF (not counting)	No cycles counted	Count only qualified cycles	No count until first TOC
TOC ON (counting)	No cycles counted	Count only qualified cycles	No count until first TOC

SECTION 8

Table of Contents

68020 Event Monitor System

68020 EVENT MONITOR SYSTEM

Overview

The ES1800 Event Monitor System is based on a 16 bit environment. The 68020 microprocessor consists of a 32 bit architecture. To incorporate the 68020 into the ES1800 chassis, some enhancements were introduced into the emulator's operating system to facilitate selective tracing. The 68020 microprocessor Event Monitor System including special tracing considerations is described in this chapter. Please review the entire chapter, including the last section on shortcuts, before using the Event Monitor System for the 68020 microprocessor.

The ES1800 Event Monitor System (EMS) provides extremely flexible system and breakpoint control, enabling the user to isolate or break on any predefined series of events and then perform various actions. The user controls and monitors the target by entering commands that define events as logical combinations of address, data, status, count limit, and optional logic state probe inputs. When an event is detected, the ES1800 can break emulation, trace specific sequences, count events, execute user supplied target routines, and trigger a TTL output.

NOTE: Simultaneous use of the Dynamic Trace feature is not possible with the Event Monitor System. Refer to the Dynamic Trace Capture Enable command in section 5 for more information.

WHEN/THEN control statements define events and their corresponding actions. There can be several actions for any event. The system only recognizes the first three letters of any word in a control statement (e.g., WHEN=WHE; THEN=THE). There can be many control statements in effect at any time. The Event Monitor System can also switch groups to allow for sequencing events. There are four event groups available and the control statements and comparator values for any group are independent of those in other groups.

Event Monitor System control statements can be entered while in run mode. The event comparator values can also be modified during run mode. However, these new statements and values will not go into effect until you stop and restart run mode.

The ES1800 Event Monitor System monitors target information at the bus cycle level; i.e., every read or write cycle that the microprocessor executes. The EMS system detects every signal that can affect the target system. It can also monitor inputs from the logic state analyzer probe.

The Event Monitor System essentially takes a picture of the Motorola 68020 microprocessor's signals at the end of every bus cycle (refer to Motorola manual, *MC68020 32 Bit Microprocessor Users Manual*). The information that is recorded into trace memory is the same information that the EMS is monitoring.

The Event Monitor System will only break on the address information that was traced. The breakpoint comparators only look at the addresses output by the microprocessor.

The basic Event Monitoring System control statement is of the form:

[*Group*] WHE[*N*] <*event*> THE[*N*] <*action*>

Notice that the ESL command processor needs only the first three letters of the symbol.

COMPARATOR REGISTERS

An event is based on the value of a comparator register or a combination of comparator registers. Comparators can be combined with AND, OR, or NOT operators. The action that follows an event may be a single command or multiple commands separated by commas.

There are eight comparator registers for each of the four event groups. These event registers are listed in the table on the following page. The address comparators are used to detect discrete addresses or addresses inside or outside a specified range. The data comparators can detect specific data patterns and can ignore specified bit positions. The status comparators monitor all of the status signals from the microprocessor as well as some generated by the ES1800. The status comparators can also ignore bit positions. The count limit register can be used to detect when an event has occurred more than a specified number of times. The logic state analyzer register can detect bit patterns in the inputs from the logic state probe.

The following table describes the available event comparator registers.

Register		Size	Name by Group			
Description	Type	(bits)	1	2	3	4
Address 1	Range,Integer	24	AC1 or AC1.1	AC1.2	AC1.3	AC1.4
Address 2	Range,Integer	24	AC2 or AC2.1	AC2.2	AC2.3	AC2.4
Data 1	Don't Care,Integer	16	DC1 or DC1.1	DC1.2	DC1.3	DC1.4
Data 2	Don't Care,Integer	16	DC2 or DC2.1	DC2.2	DC2.3	DC2.4
Status 1	Don't Care,Integer	16	S1 or S1.1	S1.2	S1.3	S1.4
Status 2	Don't Care,Integer	16	S2 or S2.1	S2.2	S2.3	S2.4
LSA	Don't Care,Integer	16	LSA or LSA.1	LSA.2	LSA.3	LSA.4
Count	Integer	16	CL or CL.1	CL.2	CL.3	CL.4

ADDRESS COMPARATORS

Address comparators may be assigned integer values or range values. Ranges may be either internal (IRA) or external (XRA). If a range is specified without IRA or XRA operators, the default range type will be IRA.

DATA AND LSA COMPARATORS

The data comparators monitor the data bus for specified patterns. The LSA comparators monitor the input pulses from the logic state probe.

Data and LSA comparators may be assigned integer values or don't care values. Don't care values may be assigned in two ways. The first is to specify the value followed by the don't care mask; the second is to specify the value using $\boxed{X}$ in the don't care positions.

STATUS COMPARATORS

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

STATUS MNEMONICS

BER	bus error
IP	interrupt pending
AV	autovector
IP(0-7)	interrupt levels 0-7
SP	supervisor program
SD	supervisor data
UP	user program
CPU	CPU space
SC(0-7)*	numeric names for all 8 space codes*
TAR	target system access
OVL	overlay memory access
RD	read access
WR	write access
BYT	byte access
WRD	word access

COUNT LIMIT COMPARATOR

The count limit comparator CL, is used to detect when events have occurred a certain number of times. The CL value for group 1 is loaded into a hardware counter which is decremented whenever the action CNT is executed (see the following sections on Defining Events and 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.

Defining Events

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

EVENT

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

The NOT operator is used to reverse the sense of the comparator output. NOT has higher precedence than either of the conjunctives. `WHEN AC1 AND NOT DC1 THEN BRK` means break whenever any data pattern other than that in DC1 occurs on the bus along with AC1.

AND and OR can be used where needed to form more restrictive event definitions. AND terms have higher precedence than OR terms. `AC1 AND DC1 OR DC2` is the same as `AC1 AND DC1` in one statement and `DC2` in another. If you are looking for two different data values at an address, use `AC1 AND DC1 OR AC1 AND DC2`.

The OR operator is evaluated left to right and is useful for simple comparator combinations. For complex event specifications, OR combinations can be replaced with separate EMS control statements for clarity. `AC1 AND S1 OR AC2 AND S2` is the same as `AC1 AND S1` and `AC2 AND S2`.

Defining Action Lists

The action list in a WHEN/THEN statement defines what the Emulator does when an event is detected. Actions are specified in an action list separated by commas. The action list may have one or more actions defined.

Examples

<group> WHEN *<event>* THEN *<action>,<action>, ... ,<action>*

The following table lists all possible actions.

Event Monitor System Actions

Action	Description
BRK	Break emulation
CNT	Count Bus Cycle
FSI	Force Special Interrupt
GRO	Change Event Group
RCT	Load Count Value
TGR	Output Trigger Signal
TOC	Toggle Count State
TOT	Toggle Trace State
TRC	Trace Bus Cycle

The **TRC** and **TOT** actions are described in the Tracing Events section. The **CNT**, **RCT**, and **TOC** actions are described in the Counting Events section. The **FSI** action is described in the Special Interrupt section. The **GRO** action is described in the Changing Event Groups section. The **TGR** action is described in the Trigger Signal section. The **BRK** is described in the Breaking Emulation section.

The EMS will always resolve conflicting EMS control statements. The **TOC** action in the first statement will be changed to **CNT**.

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

DISPLAY EVENT SPECIFICATIONS

Command	Result
DES	Displays all of the WHEN/THEN statements currently active from all groups.
DES *<group number>*	Displays all of the WHEN/THEN statements and the comparator values for the specified group.

Examples

Display the statements and comparators for groups 1 and 2.

```
 >DES 1;DES 2
1 WHEN AC1 THEN BRK
AC1.1 = $007632
AC2.1 = $000000
DC1.1 = $0000
DC2.1 = $0000
S1 .1 = $0000
S2 .1 = $0000
LSA.1 = $0000
CL .1 = $0000

2 WHEN S1 AND DC1 THEN CNT,TRC
2 WHEN CL 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
CL .2 = $0010
```

CLEAR WHEN/THEN STATEMENTS

Command	Result
CES	Clears all of the WHEN/THEN statements currently active.
CES *<group number>*	Clears all of the WHEN/THEN statements for the specified group.

Comments

The comparator values are not affected by the CES command.

Breaking Emulation

The **BRK** action stops emulation, returning the system to pause mode. When a break event is detected and emulation has been broken, the current PC and event group is displayed on the terminal. Emulation begins at the values displayed if the registers are not altered and you run or step following a break. When entering emulation, the Event Monitor System always begins looking for events specified in group 1.

Breakpoints stop program execution at specific times. After a break you can disassemble the trace memory, look at the LSA bits in the raw trace, check the CPU register values, or begin stepping through your code.

Breakpoint actions may be enabled or disabled by selecting the appropriate run commands. If you enter emulation with the **RBK** or **RBV** run commands, breakpoints are enabled. If you enter emulation with the **RUN** or **RNV** commands, breakpoints are disabled, even if there are event statements specifying the **BRK** action. If emulation is entered with breakpoints disabled, you can enable them while running by entering the **RBK** command. If you enter emulation with breakpoints enabled, you can disable them while running by entering the **RUN** command. The **RNV** and **RBV** commands are not allowed during emulation. These commands load the reset vectors, which cannot be done during emulation.

Emulation may be halted using the **STP** command. The reset character (Default is CTRL Z. May be changed through the Set command, option 2) also breaks emulation.

(continued)

Tracing Events

```
        Events:

                 TRC
                 TOT
```

The Event Monitor System can be set up to trace bus cycles selectively. If all of the conditions specified in the event portion of the WHEN/THEN clause are satisfied, the trace action, **TRC**, causes the specified bus cycle to be recorded into the trace memory.

The toggle trace, **TOT**, provides a way to turn tracing on and off. When a **TOT** event is detected, the trace is toggled to the opposite state, either on or off. You can specify a single event that starts and stops trace each time it is detected or specify any number of events that toggle trace on and off.

If there are no event actions that specify **TRC** or **TOT**, all bus cycles are traced. If there is a **TRC** event, only qualified bus cycles are traced. If there is a **TOT** event, trace will be 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.

	New Group		
Previous Group	Nothing Specified	TRC	TOT
Nothing Specified	Trace all cycles	Trace only qual'd cycles	Trace nothing until first TOT
TRC	Trace all cycles	Trace only qual'd cycles	Trace nothing until first TOT
TOT (Not tracing)	Trace all cycles	Trace only qual'd cycles	Trace nothing until first TOT
TOT (Tracing)	Trace all cycles	Trace only qual'd cycles	Trace nothing until first TOT

This table describes initial trace conditions.

Action Specified	Trace Condition
Nothing	Trace all cycles
TRC	Trace only Qual'd TRC events
TOT	Trace nothing until TOT event

(continued)

Tracing and Breaking with the 68020

The ES1800 Event Monitor System is based on a 16 bit environment. The 68020 microprocessor consists of a 32 bit architecture. To incorporate the 68020 into the ES1800 chassis, some enhancements were introduced to the Emulator's operating system to facilitate selective tracing.

The 68020 has a wider address and data path than the 68000/08/10 and other 16 bit processors. The ES1800 chassis was designed for the 16 bit processors and therefore has a 70 bit trace memory. Because of the 70 bit wide trace, it is not possible to trace all of the 68020 signals that might be of interest at the same time. Instead, a selective trace mode is available to allow the user to pick which bits are to be traced depending on the current stage of development. There are four modes the user may choose from and the bits utilized are listed below:

MODE	ADDR	DATA	STATUS	TIMER	LSA
0	24	16	19	24	16
1	24	32	19	24	0
2	32	32	11	24	0
3	32	16	21	24	6

Regardless of which trace mode is selected, the following information is always traced and is displayed by the Event Monitor System:

```
AO  - A23
D16 - D31
R/W~
TGT/OVL
FCO - FC2
```

To program the Event Monitor System, a detailed description of the signals traced and their location in the event comparators is needed. The chart on page 8-17 shows the signals and their locations for the four trace modes.

MODE SELECTION

The Event Monitor System in the ES1800 is designed to use the trace information to cause the event actions. In the 68020, the trace information is different depending on the selected mode (0, 1, 2, or 3). Refer to **SET** menu, line #4.

Trace mode 0...page 8-19

When using trace mode 0, the event system operates basically the same as the present 68010 EMS system. The AVEC˜ signal replaces VPA˜ and IPEND˜ replaces VMA˜. The breakpoint assembler recognizes AV for AVEC and IP for IPEND. Five additional signals are traced: OCS, DSACK1, DSACK0, SIZ1, and SIZ0.

Be aware, however, that only 24 address and 16 data bits are being traced which means that the upper 8 address bits and the lower 16 data bits are ignored. Also, RMC˜ and CDIS˜ are not traced in Mode 0.

Trace mode 1...page 8-27

Trace mode 1 would be selected by the user if all 32 bits of data needed to be traced. As in trace mode 0, the high 8 address bits are ignored. All the signals traced in trace mode 0 are also used in mode 1 except the 16 LSA bits, which are replaced by data bits 0-15.

Trace Mode 2...page 8-36

Trace mode 2 would be selected if all 32 address and data bits need to be traced. The LSA bits are replaced with data bits 0-15, and certain status bits are replaced with A24-A31. IPL0-2˜, IPEND˜, BERR˜, and AVEC˜ are not traced in this mode.

Trace Mode 3...page 8-45

Trace mode 3 would be selected to trace 32 address, 16 data, and all status lines. There are 6 LSA lines available for use in this mode.

(continued)

Figure 9. Address data, status lines and LSA bits for each trace mode

	101 BITS																				
	ACX BITS	DCX BITS	TIMER BITS	STATUS BITS	SX BITS																LSA BITS
BIT NO.	A23 ← A0	D15 ← D0	T23 ← T0	OCS, DSACK1,DSACK0, SIZ1, SIZ0	S15	S14	S13	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0	
MODE 0	A23 ← A0	D31 ← D16	T23 ← T0	OCS, DSACK1,DSACK0, SIZ1, SIZ0	0	IP	AV	BRK	IPL2	IPL1	IPL0	BER	MAV/ MWV	FC2	FC1	FC0	0	TGT/ OVL	R/W	B/W	See
MODE 1	A23 ← A0	D31 ← D16	T23 ← T0	OCS, DSACK1,DSACK0, SIZ1, SIZ0	0	IP	AV	BRK	IPL2	IPL1	IPL0	BER	MAV/ MWV	FC2	FC1	FC0	0	TGT/ OVL	R/W	B/W	Below
MODE 2	A23 ← A0	D31 ← D16	T23 ← T0	OCS, DSACK1,DSACK0, SIZ1, SIZ0	0	A31	A30	A29	A28	A27	A26	A25	A24	FC2	FC1	FC0	0	TGT/ OVL	R/W	B/W	
MODE 3	A23 ← A0	D31 ← D16	T23 ← T0	OCS, DSACK1,DSACK0, SIZ1, SIZ0	0	IP	AV	BRK	IPL2	IPL1	IPL0	BER	MAV/ MWV	FC2	FC1	FC0	0	TGT/ OVL	R/W	B/W	

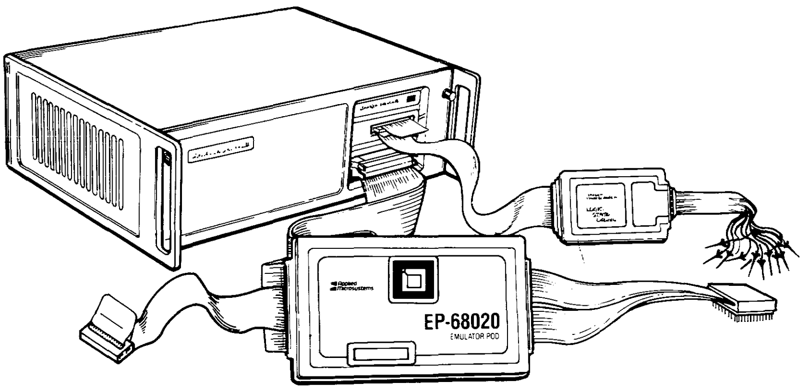
LSA BITS															
LS15	LS14	LS13	LS12	LS11	LS10	LS9	LS8	LS7	LS6	LS5	LS4	LS3	LS2	LS1	LS0
LS15	LS14	LS13	LS12	LS11	LS10	LS9	LS8	LS7	LS6	LS5	LS4	LS3	LS2	LS1	LS0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
A31	A30	A29	A28	A27	A26	A25	A24	RMC	CDS	L5	L4	L3	L2	L1	L0

(continued)

Trace Mode 0

When using trace mode 0, the 40 conductor cable from the pod is not used. It may be removed from the Logic State Probe connector on the front panel of the ES and the optional Logic State Probe installed.

Figure 10. Trace Mode 0



When using trace mode 0, the event system operates basically the same as the 68000/08/10 EMS system. The AVEC~ signal replaces VPA~ and IPEND~ replaces VMA~. The breakpoint assembler recognizes AV for AVEC and IP for IPEND. Five additional signals are traced: OCS, DSACK1, DSACK0, SIZ1, and SIZ0.

Be aware, however, that only 24 address and 16 data bits are being traced which means that the upper 8 address bits and the lower 16 data bits are ignored. Also, RMC~ and CDIS~ are not traced in mode 0.

(continued)

SETUP

To select mode 0 operation, type in:

```
> SET 4,0
```

To view the selected mode of operation, type in:

```
> SET (refer to line #4)
```

OPERATION

The ES1800 Event Monitor System is designed to use trace information to cause event actions. In the 68020, trace information is different depending on the selected mode (0, 1, 2, 3).

The trace buffer contains 24 address bits, 16 data bits, 20 status bits, 16 logic state bits, and 24 timer bits. For mode 0, these bits trace the following:

Trace address bits:

```
        A0 through A23 on the trace card will trace
        A0 through A23 of the 68020 address bus.
```

Trace data bits:

```
        D0 through D15 on the trace card will trace
        D16 through D31 of the 68020 data bus.
```

Trace status bits:

```
S0 through S14 on the trace card will trace:

Interrupt Pending               (S14)
Autovector                      (S13)
Break                           (S12)
Interrupt Line   2              (S11)
Interrupt Line   1              (S10)
Interrupt Line   0              (S9)
Bus Error                       (S8)
Memory Access Violation         (S7)
Memory Write Violation          (S7)
Function code line 2            (S6)
Function code line 1            (S5)
Function code line 0            (S4)
(not used)                      (S3)
TGT/OVL                         (S2)
Read/Write                      (S1)
Byte/Word                       (S0)
Additional bits displayed in
raw trace:
OCS
DSACK0
DSACK1
SIZ0
SIZ1
```

Trace logic state bits:

```
LS0 through LS15 on the trace card will trace
LS0 through LS15 of the logic state pod. Please refer
to installation instructions (page 8-64) for set up.
```

Examples

When setting up the Event Monitor System, comparators are used to qualify the event actions. Please refer to page 8-2, for a complete description. The comparators used are:

(continued)

- Two address comparators for use with the address area of the trace: AC1 and AC2
- Two data comparators for use with the data area of the trace: DC1 and DC2
- Two status comparators for use with the status area of the trace: S1 and S2
- One logic state comparator for use with the logic state area of the trace: LSA
- One count limit comparator for use with the internal count register: CL

When assigning values to AC1 and AC2, single values and ranges are acceptable. DC1, DC2 and LSA may have masked values. The CL register can be between 1 and $FFFF.

The status comparators (S1 and S2) may be set up with a variety of different conditions to further qualify your particular event action. The acceptable statuses are as follows:

```
BYT or WRD
RD  or WR
TAR or OVL
Any one of the function codes  SC0
                               SC1/UD
                               SC2/UP
                               SC3
                               SC4
                               SC5/SD
                               SC6/SP
                               SC7/CPU

BER
Any one of the interrupt priorities, IP0 through IP7
AV (autovector)
IP (interrupt pending)
```

When setting up a status comparator, the following examples are acceptable:

```
S1=BYT
S1=RD+OVL
S2=BER+TAR
S2=SC4+RD+BYT
S1=IP4
```

As you can see, multiple choices for the status comparators are allowed, however, only *one* from each category should be used. For example, you should not set the status comparator to BYT+WRD, RD+WR, or SC3+SC4 because these types of statuses would not happen simultaneously.

Therefore, S1 and S2 can equal the following:

```
                          STATUS MNEMONIC TABLE

S1 = BYT  +  RD +  TAR  +  SC0   +  BER  +  IP0  +  AV  +  IP0  +  AV  +  IP
     WRD     WR    OVL     SC1/UD           IP1
                           SC2/UP           IP2
                           SC3              IP3
                           SC4              IP4
                           SC5/SD           IP5
                           SC6/SP           IP6
                           SC7/CPU          IP7
(Only one out of each category)
```

(continued)

```
                      STATUS COMPARATOR BREAKDOWN

14    13     12    11    10    9   8   7   6   5   4    3   2   1   0
 |     |      |     \     |    /   |       \   |   /        |   |   |
1=IP 1=AV             0=IP0      1=BER       0=SC0        1=OVL |   |
                      1=IP1                  1=SC1/UD     1=TAR |   |
                      2=IP2                  2=SC2/UP         0=WR  |
                      3=IP3                  3=SC3            1=RD  |
                      4=IP4                  4=SC4               0=WRD
                      5=IP5                  5=SC5/SD            1=BYT
                      6=IP6                  6=SC6/SP
                      7=IP7                  7=SC7/CPU
```

```
                STATUS COMPARATOR BIT REPRESENTATION

IP=00004000 DC BFFF
AV=00002000 DC DFFF
IP0=00000000 DC FF1F
IP1=00000200 DC F1FF
IP2=00000400 DC F1FF
IP3=00000600 DC F1FF
IP4=00000800 DC F1FF
IP5=00000A00 DC F1FF
IP6=00000C00 DC F1FF
IP7=00000E00 DC F1FF
BER=00000100 DC FEFF
SC0=00000000 DC FF8F
SC1/UD=00000010 DC FF8F
SC2/UP=00000020 DC FF8F
SC3=00000030 DC FF8F
SC4=00000040 DC FF8F
SC5/SD=00000050 DC FF8F
SC6/SP=00000060 DC FF8F
SC7/CPU=00000070 DC FF8F
TAR=00000004 DC FFFB
OVL=00000000 DC FFFB
RD=00000002 DC FFFD
WR=00000000 DC FFFD
BYT=00000001 DC FFFE
WRD=00000000 DC FFFE
```

Example of mode 0 display:

```
>SET
ES SETUP: SEE MANUAL FOR DETAILS...
SET #X, #Y - SET ITEM X TO VALUE CORRESPONDING TO Y
LD 0;SAV 0   LOAD/SAVE SETUP FOR CURRENTLY SELECTED USER

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

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

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

(continued)

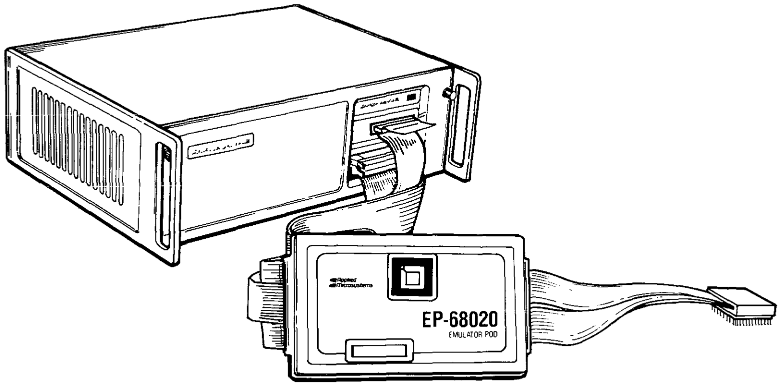
Example of mode 0 display:

```
>DRT
LINE  ADDRESS   DATA  R/W        FC  IPL    LSA  -  8  7  -  0      TIME
 #20  F0053C  > 51CB   R   TAR   SP   0     %11111111 %11111111  307.418 MS
 #19  F0053E  > FFF8   R   TAR   SP   0     %11111111 %11111111  307.418 MS
 #18  F00540  > 6024   R   TAR   SP   0     %11111111 %11111111  307.419 MS
 #17  F00542  > 4A03   R   TAR   SP   0     %11111111 %11111111  307.420 MS
 #16  F00534  > 007E   R   TAR   SP   0     %11111111 %11111111  307.421 MS
 #15  F00536  > 301A   R   TAR   SP   0     %11111111 %11111111  307.421 MS
 #14  F00538  > E309   R   TAR   SP   0     %11111111 %11111111  307.422 MS
 #13  F0053A  > 6506   R   TAR   SP   0     %11111111 %11111111  307.423 MS
 #12  F005BA  > 0000   R   TAR   SD   0     %11111111 %11111111  307.424 MS
 #11  F0053C  > 51CB   R   TAR   SP   0     %11111111 %11111111  307.424 MS
 #10  F0053E  > FFF8   R   TAR   SP   0     %11111111 %11111111  307.425 MS
  #9  F00540  > 6024   R   TAR   SP   0     %11111111 %11111111  307.426 MS
  #8  F00542  > 4A03   R   TAR   SP   0  IP %11111111 %11111111  307.427 MS
  #7  F00534  > 007E   R   TAR   SP   0  IP %11111111 %11111111  307.427 MS
  #6  F00536  > 301A   R   TAR   SP   0  IP %11111111 %11111111  307.428 MS
  #5  F00538  > E309   R   TAR   SP   0  IP %11111111 %11111111  307.429 MS
  #4  F0053A  > 6506   R   TAR   SP   0  IP %11111111 %11111111  307.429 MS
  #3  F005BC  > 0000   R   TAR   SD   0  IP %11111111 %11111111  307.430 MS
  #2  F0053C  > 51CB   R   TAR   SP   0  IP %11111111 %11111111  307.431 MS
  #1  F0053E  > FFF8   R   TAR   SP   0  IP %11111111 %11111111  305.431 MS
  #0  BREAK
```

Trace Mode 1

Trace mode 1 requires that the 40 conductor cable from the pod be connected to the Logic State Probe connector on the front panel of the ES.

Figure 11. Trace Mode 1 and 2



Trace mode 1 would be selected by the user if all 32 bits of data needed to be traced. As in trace mode 0, the high 8 address bits are ignored. All the signals traced in trace mode 0 are also used in mode 1 except the 16 LSA bits, which are replaced by data bits 0-15. The LSA field in the raw trace display is not shown. The Event Monitor System, however, still uses the LSA comparator for this field, so in order to detect a 32 bit data event, a data comparator *and* the LSA comparator must be set up.

SETUP

To select mode 1 operation, type in:

```
>SET 4,1
```

(continued)

To view the selected mode of operation, type in:

```
>SET  (refer to line #4)
```

Disconnect the Logic State Probe option and replace with the smaller cable that comes out of the 68020 pod.

OPERATION

The ES1800 Event Monitor System is designed to use trace information to cause event actions. In the 68020, trace information is different depending on the selected mode (0, 1, 2, or 3).

The trace buffer contains 24 address bits, 16 data bits, 20 status bits, 16 logic state bits, and 24 timer bits. For mode 1, these bits trace the following:

Trace address bits:

```
A0 through A23 on the trace card will trace
A0 through A23 of the 68020 address bus.
```

Trace data bits:

```
D0 through D15 on the trace card will trace
D16 through D31 of the 68020 data bus.
```

Trace status bits:

S0 through S14 on the trace card will trace:

Interrupt Pending	(S14)
Autovector	(S13)
Break	(S12)
Interrupt Line 2	(S11)
Interrupt Line 1	(S10)
Interrupt Line 0	(S9)
Bus Error	(S8)
Memory Access Violation	(S7)
Memory Write Violation	(S7)
Function code line 2	(S6)
Function code line 1	(S5)
Function code line 0	(S4)
(not used)	(S3)
TGT/OVL	(S2)
Read/Write	(S1)
Byte/Word	(S0)

Additional bits displayed in raw trace:
OCS
DSACK0
DSACK1
SIZ0
SIZ1

Trace logic state bits:

LS0 through LS15 on the trace card will trace
D0 through D15 of the 68020 data bus.

When viewing the raw trace, all information will be in logical areas for easy viewing. For example, even though the logic state bits are tracing the lower data bits, the lower data bits will show in the data field of the trace. The only item that is affected by this arrangement is the way the comparators are used for setting up the Event Monitor System.

(continued)

Examples

When setting up the Event Monitor System, comparators are used to qualify the event actions. Please refer to page 8-2 for a complete description. The comparators used are:

- Two address comparators for use with the address area of the trace: AC1 and AC2
- Two data comparators for use with the data area of the trace: DC1 and DC2
- Two status comparators for use with the status area of the trace: S1 and S2
- One logic state comparator for use with the lower data bits of the trace: LSA
- One count limit comparator for use with the internal count register: CL

When assigning values to AC1 and AC2, single values and ranges are acceptable. DC1, DC2 and LSA may have masked values. The CL register can be between 1 and $FFFF.

To set up a comparator using the data bus, please remember to use the data comparators (DC1 and DC2) *and* the logic state comparator (LSA). For example, to break on data pattern $12345678, use the following procedure:

```
>DC1=$1234
>LSA=$5678
>WHEN DC1 AND LSA THEN BRK
>RBK
```

Notice the LSA comparator is used for the lower data bits (D0-D15) and DC1 is used for the upper data bits (D16-D32).

The status comparators (S1 and S2) may be set up with a variety of different conditions to further qualify your particular event action. The acceptable statuses are as follows:

```
BYT or WRD
RD  or WR
TAR or OVL
Any one of the function codes  SC0
                               SC1/UD
                               SC2/UP
                               SC3
                               SC4
                               SC5/SD
                               SC6/SP
                               SC7/CPU

BER
Any one of the interrupt priorities, IP0 through IP7
AV (autovector)
IP (interrupt pending)
```

When setting up a status comparator, the following examples are acceptable:

```
S1=BYT
S1=RD+OVL
S2=BER+TAR
S2=SC4+RD+BYT
S1=IP4
```

As you can see, multiple choices for the status cmparators are allowed, however, only *one* from each category should be used. For example, you should not set the status comparator to BYT+WRD, RD+WR, or SC3+SC4 because these types of statuses would not happen simultaneously.

(continued)

Therefore, S1 and S2 can equal the following:

```
                          STATUS MNEMONIC TABLE

S1 = BYT  +  RD  +  TAR  +  SC0  +  BER  +  IP0  +  AV  +  IP0  +  AV  +  IP
     WRD     WR     OVL     SC1/UD          IP1
                            SC2/UP          IP2
                            SC3             IP3
                            SC4             IP4
                            SC5/SD          IP5
                            SC6/SP          IP6
                            SC7/CPU         IP7
(Only one out of each category)
```

```
                        STATUS COMPARATOR BREAKDOWN

14    13     12    11    10    9   8   7   6   5   4    3   2   1   0
 |     |      |     \     |   /    |        \  |  /         |   |   |
1=IP 1=AV              0=IP0     1=BER      0=SC0        0=OVL |   |
                       1=IP1                1=SC1/UD     1=TAR |   |
                       2=IP2                2=SC2/UP         0=WR  |
                       3=IP3                3=SC3            1=RD  |
                       4=IP4                4=SC4                0=WRD
                       5=IP5                5=SC5/SD             1=BYT
                       6=IP6                6=SC6/SP
                       7=IP7                7=SC7/CPU
```

```
                        STATUS COMPARATOR BIT REPRESENTATION

IP=00004000 DC BFFF
AV=00002000 DC DFFF
IP0=00000000 DC FF1F
IP1=00000200 DC F1FF
IP2=00000400 DC F1FF
IP3=00000600 DC F1FF
IP4=00000800 DC F1FF
IP5=00000A00 DC F1FF
IP6=00000C00 DC F1FF
IP7=00000E00 DC F1FF
BER=00000100 DC FEFF
SC0=00000000 DC FF8F
SC1/UD=00000010 DC FF8F
SC2/UP=00000020 DC FF8F
SC3=00000030 DC FF8F
SC4=00000040 DC FF8F
SC5/SD=00000050 DC FF8F
SC6/SP=00000060 DC FF8F
SC7/CPU=00000070 DC FF8F
TAR=00000004 DC FFFB
OVL=00000000 DC FFFB
RD=00000002 DC FFFD
WR=00000000 DC FFFD
BYT=00000001 DC FFFE
WRD=00000000 DC FFFE
```

(continued)

Example of mode 1 display:

```
>SET
ES SETUP: SEE MANUAL FOR DETAILS...
SET #X, #Y - SET ITEM X TO VALUE CORRESPONDING TO Y
LD 0;SAV 0   LOAD/SAVE SETUP FOR CURRENTLY SELECTED USER

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

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

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

Example of mode 1 display:

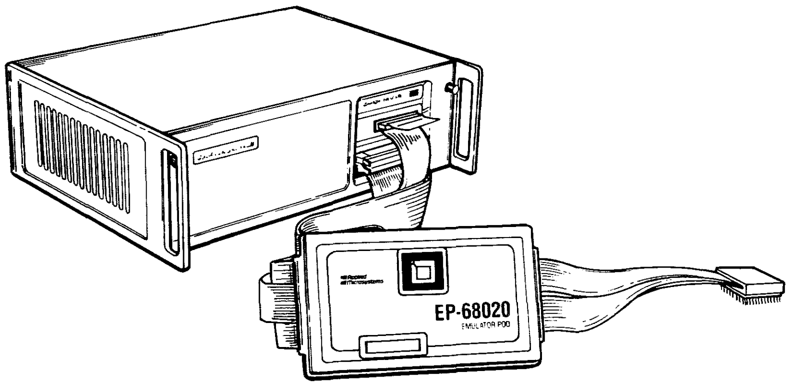
```
>DRT
LINE  ADDRESS   DATA      R/W       FC    IPL       DS   SZ   OCS    TIME
 #20  F0053C  > 51CBFFFF  R    TAR  SP    0         01   00   0      1.396 S
 #19  F0053E  > FFF8FFFF  R    TAR  SP    0         01   10   1      1.396 S
 #18  F00540  > 6024FFFF  R    TAR  SP    0         01   00   0      1.396 S
 #17  F00542  > 4A03FFFF  R    TAR  SP    0         01   10   1      1.396 S
 #16  F00534  > 007EFFFF  R    TAR  SP    0         01   00   0      1.396 S
 #15  F00536  > 301AFFFF  R    TAR  SP    0         01   10   1      1.396 S
 #14  F00538  > E309FFFF  R    TAR  SP    0         01   00   0      1.396 S
 #13  F0053A  > 6506FFFF  R    TAR  SP    0         01   10   1      1.396 S
 #12  F005B4  > 004AFFFF  R    TAR  SD    0         01   10   0      1.396 S
 #11  F0053C  > 51CBFFFF  R    TAR  SP    0         01   00   0      1.396 S
 #10  F0053E  > FFF8FFFF  R    TAR  SP    0         01   10   1      1.396 S
  #9  F00540  > 6024FFFF  R    TAR  SP    0         01   00   0      1.396 S
  #8  F00542  > 4A03FFFF  R    TAR  SP    0         01   10   1      1.396 S
  #7  F00534  > 007EFFFF  R    TAR  SP    0    IP   01   00   0      1.396 S
  #6  F00536  > 301AFFFF  R    TAR  SP    0    IP   01   10   1      1.396 S
  #5  F00538  > E309FFFF  R    TAR  SP    0    IP   01   00   0      1.396 S
  #4  F0053A  > 6506FFFF  R    TAR  SP    0    IP   01   10   1      1.396 S
  #3  F005B6  > 0056FFFF  R    TAR  SD    0    IP   01   10   0      1.396 S
  #2  F0053C  > 51CBFFFF  R    TAR  SP    0    IP   01   00   0      1.396 S
  #1  F0053E  > FFF8FFFF  R    TAR  SP    0    IP   01   10   1      1.396 S
  #0  BREAK
```

(continued)

Trace Mode 2

Using trace mode 2 requires that the 40 conductor cable from the pod be connected to the Logic State Probe connector on the front panel of the ES.

Figure 12. Mode 1 and 2



Trace mode 2 would be selected if all 32 address and data bits need to be traced. The LSA bits are replaced with data bits 0-15, and certain status bits are replaced with A24-A31. IPL0-2˜, IPEND˜, BERR˜, and AVEC˜ are not traced in this mode.

SETUP

To select mode 2 operation, type in:

```
>SET 4,2
```

To view the selected mode of operation, type in:

```
>SET  (refer to line #4)
```

Disconnect the Logic State Probe option and replace with the smaller cable that comes out of the 68020 pod.

OPERATION

The ES1800 Event Monitor System is designed to use trace information to cause event actions. In the 68020, trace information is different depending on the selected mode (0, 1, 2, or 3).

The trace buffer contains 24 address bits, 16 data bits, 20 status bits, 24 timer bits, and 16 logic state bits. For mode 2, these bits trace the following:

Trace address bits:

```
          A0 through A23 on the trace card will trace
          A0 through A23 of the 68020 address bus.
```

Trace data bits:

```
          D0 through D15 on the trace card will trace
          D16 through D31 of the 68020 data bus.
```

(continued)

Trace status bits:

```
S0 through S14 on the trace card will trace:

68020 Address Bit 31          (S14)
68020 Address Bit 30          (S13)
68020 Address Bit 29          (S12)
68020 Address Bit 28          (S11)
68020 Address Bit 27          (S10)
68020 Address Bit 26          (S9)
68020 Address Bit 25          (S8)
68020 Address Bit 24          (S7)
Function code line 2          (S6)
Function code line 1          (S5)
Function code line 0          (S4)
(not used)                    (S3)
TGT/OVL                       (S2)
Read/Write                    (S1)
Byte/Word                     (S0)
```

Trace logic state bits:

```
LS0 through LS15 on the trace card will trace
D0 through D15 of the 68020 data bus.
```

When viewing the trace, all information will be in logical areas for easy viewing. For example, even though the logic state bits are tracing the lower data bits, the lower data bits will show in the data field of the trace. The only item that is affected by this arrangement is the way the comparators are used for setting up the Event Monitor System.

Examples

When setting up the Event Monitor System, comparators are used to qualify the event actions. Please refer to page 8-2 for a complete description. The comparators used are:

- Two address comparators for use with the address area of the trace: AC1 and AC2

- Two data comparators for use with the data area of the trace: DC1 and DC2
- Two status comparators for use with the status area of the trace: S1 and S2
- One logic state comparator for use with the lower data bits of the trace: LSA
- One count limit comparator for use with the internal count register: CL

When assigning values to AC1 and AC2, single values and ranges are acceptable. DC1, DC2 and LSA may have masked values. The CL register can be between 1 and $FFFF.

To set the Event Monitor System to break on a 32 bit address, please remember that you will need to use a status comparator (S1 or S2) *and* an address comparator (AC1 or AC2). For example, to break on the address value $FFFF3040, use the following procedure:

```
>AC1=$FF3040
>S1=$7F80 DC $807F
>WHEN AC1 AND S1 THEN BRK
>RBK
```

Notice the AC1 comparator is looking for the lower 24 bits (A0-A23) and the S1 comparator (S7-S14) is looking for the upper 8 address bits (A24-A31). The "DC $807F" is a mask that will cause the Event Monitor System to ignore the other information being traced on those particular status lines when determining if a comparator has been reached. However, the trace will still display that information. It may be simpler to use binary. In this case type the following:

```
>S1=%0111111110000000  DC  %1000000001111111
```

S1 will be displayed in hexadecimal unless the global default (DFB) or the S1 default base has been altered. Please refer to page 5-85.

(continued)

The following chart may aid in determing the status values to select.

```
 S14  S13  S12  S11  S10  S9  S8  S7  S6  S5  S4  S3  S2  S1  S0
  |    |    |    |    |    |   |   |   \   |   /       |   |   |
 A31  A30  A29  A28  A27  A26 A25 A24   F.CODES        T/O W/R W/B
 |              |                |                |              |
 |              |                |                |              |
 |---------------------------------------------------------------|
```

Another example of breaking on a 32 bit address using address $12345678 would be:

```
>AC1=$345678
>S1 =$900 DC $807F
>WHEN AC1 AND S1 THEN BRK
>RBK
```

To set up a comparator using the data bus, remember to use the data comparators (DC1 and DC2) and the logic state comparator (LSA). For example, when trying to break on the data pattern $12345678, use the following procedure:

```
>DC1=$1234
>LSA=$5678
>WHEN DC1 AND LSA THEN BRK
>RBK
```

Notice the LSA comparators (S1 and S2) is used for the lower data bits (D0-D15) and DC1 is used for the upper data bits (D16-D32).

The status comparator may be set up with a variety of different conditions to further qualify your particular event action. The acceptable statuses are as follows:

```
BYT or WRD
RD  or WR
TAR or OVL
Any one of the function codes  SC0
                               SC1/UD
                               SC2/UP
                               SC3
                               SC4
                               SC5/SD
                               SC6/SP
                               SC7/CPU
```

When setting up a status comparator, the following examples are acceptable:

```
S1=BYT
S1=RD+OVL
S2=SC4+RD+BYT
```

As you can see, multiple choices for the status comparators are allowed, however, only *one* from each category should be used. For example, you should not set the status comparator to BYT+WRD, RD+WR, or SC3+SC4 because these types of statuses would not happen simultaneously.

Therefore, S1 and S2 can equal the following:

```
                         STATUS MNEMONIC TABLE

S1   =    BYT    +    RD    +    TAR    +    SC0
          WRD         WR         OVL         SC1/UD
                                             SC2/UP
                                             SC3
                                             SC4
                                             SC5/SD
                                             SC6/SP
                                             SC7/CPU
(Only one out of each category)
```

(continued)

```
                          STATUS COMPARATOR BREAKDOWN

 14    13    12    11    10    9   8   7   6   5   4    3   2   1   0
  |     |     |     |     |    |   |   |   \   |   /        |   |   |
 A31   A30   A29   A28   A27  A26 A25 A24   0=SC0          0=OVL |   |
                                            1=SC1/UD       1=TAR |   |
                                            2=SC2/UP           0=WR  |
                                            3=SC3              1=RD  |
                                            4=SC4                 0=WRD
                                            5=SC5/SD              1=BYT
                                            6=SC6/SP
                                            7=SC7/CPU
```

```
                    STATUS COMPARATOR BIT REPRESENTATION

 SC0=00000000 DC FF8F
 SC1/UD=00000010 DC FF8F
 SC2/UP=00000020 DC FF8F
 SC3=00000030 DC FF8F
 SC4=000000040 DC FF8F
 SC5/SD=00000050 DC FF8F
 SC6/SP=00000060 DC FF8F
 SC7/CPU=00000070 DC FF8F
 TAR=00000004 DC FFFB
 OVL=00000000 DC FFFB
 RD=00000002 DC FFFD
 WR=00000000 DC FFFD
 BYT=00000001 DC FFFE
 WRD=00000000 DC FFFE
```

Example of mode 2 display:

```
>SET
ES SETUP: SEE MANUAL FOR DETAILS...
SET #X, #Y - SET ITEM X TO VALUE CORRESPONDING TO Y
LD 0;SAV 0   LOAD/SAVE SETUP FOR CURRENTLY SELECTED USER

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

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

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

(continued)

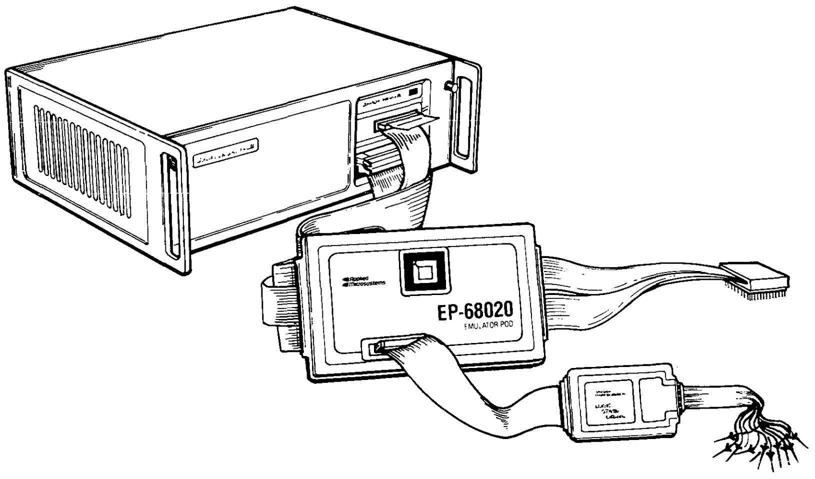
Example of mode 2 display:

```
>DRT
LINE  ADDRESS      DATA     R/W        FC   DS  SZ  OCS    TIME
 #20  FFF00506  >  00C0FFFF  R   TAR   SP   01  10   1     1.215 S
 #19  FFF00508  >  206DFFFF  R   TAR   SP   01  00   0     1.215 S
 #18  FFF0050A  >  041AFFFF  R   TAR   SP   01  10   1     1.215 S
 #17  FFF21C72  <  FFFBFFFB  W   TAR   SD   01  00   0     1.215 S
 #16  FFF21C74  <  00400040  W   TAR   SD   01  10   1     1.215 S
 #15  FFF21C6E  <  FFF2FFF2  W   TAR   SD   01  00   0     1.215 S
 #14  FFF21C70  <  04220422  W   TAR   SD   01  10   1     1.215 S
 #13  FFF0050C  >  2268FFFF  R   TAR   SP   01  00   0     1.215 S
 #12  FFF0050E  >  000CFFFF  R   TAR   SP   01  10   1     1.215 S
 #11  FFF2041A  >  FFF2FFFF  R   TAR   SD   01  00   0     1.215 S
 #10  FFF2041C  >  0422FFFF  R   TAR   SD   01  10   1     1.215 S
  #9  FFF00510  >  4A29FFFF  R   TAR   SP   01  00   0     1.215 S
  #8  FFF00512  >  0007FFFF  R   TAR   SP   01  10   1     1.215 S
  #7  FFF2042E  >  FFFBFFFF  R   TAR   SD   01  00   0     1.215 S
  #6  FFF20430  >  0040FFFF  R   TAR   SD   01  10   1     1.215 S
  #5  FFF00514  >  6A02FFFF  R   TAR   SP   01  00   0     1.215 S
  #4  FFF00516  >  6106FFFF  R   TAR   SP   01  10   1     1.215 S
  #3  FFFB0047  >  44FFFFFF  R   TAR   SD   10  01   0     1.215 S
  #2  FFF00518  >  4CDFFFFF  R   TAR   SP   01  00   0     1.215 S
  #1  FFF0051A  >  0300FFFF  R   TAR   SP   01  10   1     1.215 S
  #0  BREAK
```

Trace Mode 3

Trace mode 3 requires that the 40 conductor cable from the Pod be connected to the Logic State Probe connector on the front panel of the ES.

Figure 13. Trace Mode 3



Mode 3 would be selected to trace 32 address, 16 data, and all status lines. There are 6 LSA lines available for use in this mode.

In this mode, the high address byte (A24-A31) is set up in the high 8 bits of the LSA register and the statuses are assigned as in modes 1 and 2. (Refer to page 8-31)

SETUP

To select mode 3 operation, type in:

```
>SET 4,3
```

(continued)

To view the selected mode of operation, type in:

```
  >SET  (refer to line #4)
```

Disconnect the Logic State Probe option and replace with the smaller cable that comes out of the 68020 pod. Connect the LSA pod to the 68020 pod face plate.

OPERATION

The ES1800 Event Monitor System is designed to use trace information to cause event actions. In the 68020, trace information is different depending on the selected mode (0, 1, 2, or 3).

The trace buffer contains 24 address bits, 16 data bits, 20 status bits, 16 logic state bits, and 24 timer bits. For mode 3, these bits trace the following:

Trace address bits:

```
        A0 through A23 on the trace card will trace
        A0 through A23 of the 68020 address bus.
```

Trace data bits:

```
        D0 through D15 on the trace card will trace
        D16 through D31 of the 68020 data bus.
```

Trace status bits:

S0 through S14 on the trace card will trace:

Interrupt Pending	(S14)
Autovector	(S13)
Break	(S12)
Interrupt Line 2	(S11)
Interrupt Line 1	(S10)
Interrupt Line 0	(S9)
Bus Error	(S8)
Memory Access Violation	(S7)
Memory Write Violation	(S7)
Function code line 2	(S6)
Function code line 1	(S5)
Function code line 0	(S4)
(not used)	(S3)
TGT/OVL	(S2)
Read/Write	(S1)
Byte/Word	(S0)

Trace logic state bits:

```
LS0 through LS15 on the trace card will trace
the following:

68020 Address Bit 31 on LS15
68020 Address Bit 30 on LS14
68020 Address Bit 29 on LS13
68020 Address Bit 28 on LS12
68020 Address Bit 27 on LS11
68020 Address Bit 26 on LS10
68020 Address Bit 25 on LS9
68020 Address Bit 24 on LS8
Read Modify Write Cycle (RMC) on LS7
Cache Disable (CDIS) on LS6
```

The remaining lines (LS0-LS5) come from the Logic State Analyzer pod option. To use these lines, plug the pod into the connector located on the 68020 pod face plate. Use only the first 6 lines.

(continued)

When viewing the trace, all information will be in logical areas for easy viewing. For example, even though the logic state bits are tracing the upper data bits, these data bits will be displayed in the address field of the trace. The only item that is affected by this arrangement is the way the comparators are used for setting up the Event Monitor System.

Examples

When setting up the Event Monitor System, comparators are used to qualify the event actions. Please refer to page 8-2 for a complete description. The comparators used are:

- Two address comparators for use with the address area of the trace: AC1 and AC2
- Two data comparators for use with the data area of the trace: DC1 and DC2
- Two status comparators for use with the status area of the trace: S1 and S2
- One logic state comparator for use with the lower data bits of the trace: LSA
- One count limit comparator for use with the internal count register: CL

When assigning values to AC1 and AC2, single values and ranges are acceptable. DC1, DC2 and LSA may have masked values. The CL register can be between 1 and $FFFF.

To set up a comparator using the address bus, please remember to use the address comparators (AC1 and AC2) *and* the logic state comparators (LSA). For example, when trying to break on the address $12345678, use the following procedure:

```
>AC1=$345678
>LSA=$12 DC $00FF
>WHEN AC1 AND LSA THEN BRK
>RBK
```

Notice the LSA comparator is used for the upper address bits (A24-A31) and AC1 is used for the lower address bits (A0-A23). The "DC $00FF" is the mask that will cause the Event Monitor System to ignore the information that is being traced in the remaining LSA bits.

The following chart may aid in setting up the LSA comparator:

```
LS15 LS14 LS13 LS12 LS11 LS10 LS9 LS8 LS7 LS6 LS5 LS4 LS3 LS2 LS1 LS0
  |    |    |    |    |    |    |   |   |   |   |   |   |   |   |   |
A31  A30  A29  A28  A27  A26  A25 A24 RMC CDC LS5 LS4 LS3 LS2 LS1 LS0
|                   |                 |           |                 |
|-------------------------------------------------------------------|
```

To setup the Event Monitor System to break on cache disable, use the following example:

```
>LSA = 40 DC $FFBF
>WHEN LSA THEN BRK
>RBK
```

The status comparators (S1 and S2) may be set up with a variety of different conditions to further qualify your particular event action. The acceptable statuses are as follows:

(continued)

```
BYT or WRD
RD  or WR
TAR or OVL
Any one of the function codes  SC0
                               SC1/UD
                               SC2/UP
                               SC3
                               SC4
                               SC5/SD
                               SC6/SP
                               SC7/CPU

BER
Any one of the interrupt priorities, IP0 through IP7
AV (autovector)
IP (interrupt pending)
```

When setting up a status comparator, the following examples are acceptable:

```
S1=BYT
S1=RD+OVL
S2=BER+TAR
S2=SC4+RD+BYT
S1=IP4
```

As you can see, multiple choices for the status comparators are allowed, however, only *one* from each category should be used. For example, you should not set the status comparator to BYT+WRD, RD+WR, or SC3+SC4 because these types of statuses would not happen simultaneously.

Therefore, S1 and S2 can equal the following:

```
                        STATUS MNEMONIC TABLE

S1 = BYT  +  RD  +  TAR  +  SC0   +  BER  +  IP0  +  AV  +  IP0  +  AV  +  IP
     WRD     WR     OVL     SC1/UD          IP1
                            SC2/UP          IP2
                            SC3             IP3
                            SC4             IP4
                            SC5/SD          IP5
                            SC6/SP          IP6
                            SC7/CPU         IP7
(Only one out of each category)
```

```
                      STATUS COMPARATOR BREAKDOWN

14    13    12     11    10    9   8   7   6   5   4    3   2   1   0
 |     |     |      \    |    /    |        \  |  /         |   |   |
1=IP  1=AV            0=IP0      1=BER       0=SC0        0=OVL |   |
                      1=IP1                  1=SC1/UD     1=TAR |   |
                      2=IP2                  2=SC2/UP         0=WR  |
                      3=IP3                  3=SC3            1=RD  |
                      4=IP4                  4=SC4                0=WRD
                      5=IP5                  5=SC5/SD             1=BYT
                      6=IP6                  6=SC6/SP
                      7=IP7                  7=SC7/CPU
```

(continued)

```
                    STATUS COMPARATOR BIT REPRESENTATION

IP=4000 DC BFFF
AV=2000 DC DFFF
IP0=0 DC FF1F
IP1=00000200 DC F1FF
IP2=00000400 DC F1FF
IP3=00000600 DC F1FF
IP4=00000800 DC F1FF
IP5=00000A00 DC F1FF
IP6=00000C00 DC F1FF
IP7=00000E00 DC F1FF
BER=00000100 DC FEFF
SC0=00000000 DC FF8F
SC1/UD=00000010 DC FF8F
SC2/UP=00000020 DC FF8F
SC3=00000030 DC FF8F
SC4=000000040 DC FF8F
SC5/SD=000000050 DC FF8F
SC6/SP=000000060 DC FF8F
SC7/CPU=000000070 DC FF8F
TAR=00000004 DC FFFB
OVL=00000000 DC FFFB
RD=00000002 DC FFFD
WR=00000000 DC FFFD
BYT=00000001 DC FFFE
WRD=00000000 DC FFFE
```

Example of mode 3 display:

```
>SET
ES SETUP: SEE MANUAL FOR DETAILS...
SET #X, #Y - SET ITEM X TO VALUE CORRESPONDING TO Y
LD 0;SAV 0   LOAD/SAVE SETUP FOR CURRENTLY SELECTED USER

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

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

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

(continued)

Example of mode 3 display:

```
>DRT
LINE  ADDRESS      DATA  R/W       FC    IPL     LSA - O    TIME
 #20  FFF21C5E   < FFFB  W   TAR   SD    0  IP   %111111  500.183 MS
 #19  FFF21C60   < 0040  W   TAR   SD    0  IP   %111111  500.184 MS
 #18  FFF21C5A   < FFF2  W   TAR   SD    0  IP   %111111  500.184 MS
 #17  FFF21C5C   < 0422  W   TAR   SD    0  IP   %111111  500.185 MS
 #16  FFF21C56   < 0000  W   TAR   SD    0  IP   %111111  500.186 MS
 #15  FFF21C58   < 0000  W   TAR   SD    0  IP   %111111  500.186 MS
 #14  FFF21C52   < 0000  W   TAR   SD    0  IP   %111111  500.187 MS
 #13  FFF21C54   < 0000  W   TAR   SD    0  IP   %111111  500.188 MS
 #12  FFF21C4E   < 0000  W   TAR   SD    0  IP   %111111  500.188 MS
 #11  FFF21C50   < 0008  W   TAR   SD    0  IP   %111111  500.189 MS
 #10  FFF21C4A   < 0000  W   TAR   SD    0  IP   %111111  500.190 MS
  #9  FFF21C4C   < 00FE  W   TAR   SD    0  IP   %111111  500.190 MS
  #8  FFF21C46   < 0000  W   TAR   SD    0  IP   %111111  500.191 MS
  #7  FFF21C48   < 0003  W   TAR   SD    0  IP   %111111  500.192 MS
  #6  FFF21C42   < 0000  W   TAR   SD    0  IP   %111111  500.192 MS
  #5  FFF21C44   < 0003  W   TAR   SD    0  IP   %111111  500.193 MS
  #4  FFF21C3E   < 0014  W   TAR   SD    0  IP   %111111  500.193 MS
  #3  FFF21C40   < 000C  W   TAR   SD    0  IP   %111111  500.194 MS
  #2  FFF21C3A   < 0000  W   TAR   SD    0  IP   %111111  500.195 MS
  #1  FFF21C3C   < 0001  W   TAR   SD    0  IP   %111111  500.195 MS
  #0  BREAK
```

Counting Events

```
Registers:
                                        Value Type - 16 bit integer
        CL
        CL.1
        CL.2
        CL.3
        CL.4

        CL=<EXP>
        CL<.group>=<EXP>

Events:

        CNT
        RCT
        TOC
```

Events can be defined to count bus cycles selectively. There is one hardware counter and there are four count registers, one register for each group. Each group has its own count limit register (CL). The Event Monitor System has only one counter. The hardware counter is automatically loaded with the count limit register for group 1 when entering run mode.

The count, **CNT**, action decrements the hardware counter. When the count reaches zero, the CL event becomes true. If all other conditions specified in the WHEN/THEN clause are satisfied, the appropriate action is taken.

Whenever the reset count, **RCT**, action is specified, the count comparator value for the specified group is loaded into the hardware counter. When switching groups, the current value of the hardware counter is passed along as a global count value unless a **RCT** action is specified in the same list of events that causes the group switch.

(continued)

The toggle count, **TOC**, command provides a way to turn counting on and off. When a **TOC** event is detected, the count is toggled to the opposite state, either on or off. You can specify an event that starts and stops the counter each time it is detected or specify any number of events that toggle the counter on and off.

The current value of the counter cannot be read. You can only detect when you have reached a limit.

This table describes the count conditions immediately before an after a group change.

	New Group		
Previous Group	Nothing Specified	CNT	TOC
Nothing Specified	No cycles counted	Count only qual'd cycles	Count nothing until first TOC
CNT	No cycles counted	Count only qual'd cycles	Count nothing until first TOC
TOC (Not counting)	No cycles counted	Count only qual'd cycles	Count nothing until first TOC
TOC (Tracing)	No cycles counted	Count only qual'd cycles	Count all until first TOC

This table describes initial count conditions (always group 1).

Action Specified	Count Condition
Nothing	Count nothing
CNT	Count only Qual'd CNT events
TOC	Count nothing until TOC event

Examples

Count the times that the specified data are written to a specific address. Break if the data are written 20 times.

```
 >CL=#20
 >S1=WR
 >AC1=4020; DC1=$XXF3
 >WHEN AC1 AND DC1 AND S1 THEN CNT
 >WHEN CL THEN BRK
 >RBK
R>
```

(continued)

To cause a break to occur after $400 writes have lapsed:

```
S1 = WR
CL = 400
WHEN S1 THEN CNT
WHEN CL THEN BRK
```

Look for a read from a specific I/O port. After it is found go to group 2, load the group 2 counter register value into the hardware counter and set a group 2 address comparator to count every bus cycle (all addresses). Break after 100 bus cycles.

```
 >AC1='IOport
 >S1=RD
 >WHEN AC1 AND S1 THEN GRO 2, RCT
 >CL.2=#100
 >AC1.2=0 TO -1
 >2 WHEN AC1 THEN CNT
 >2 WHEN CL THEN BRK
 >RBK
R>
```

Trigger Signal

The trigger signal is an output that is available from the BNC connector labelled TRIG on the back panel of the ES1800 chassis and from pin 19 of the optional LSA pod. When a **TGR** event is detected, the trigger signal is asserted, and remains so for the duration of the specified bus cycle. If a trigger event is specified for more than one consecutive bus cycle, the signal stays high for the duration of the consecutive bus cycles.

The trigger signal can be used as a pulse output for triggering other diagnostic equipment. It can also be used in conjunction with a counter/timer for timing subroutines.

Examples

Trigger a scope when reading data from a UART.

```
>AC1='DATA_PORT
>S1=RD
>WHEN AC1 AND S1 THEN TGR
```

Determine the duration of a subroutine using the trigger pulse. The trigger pulse can be the input to a counter/timer or a scope. The duration of the subroutine can be determined from the pulse width displayed on the scope or the counter/timer readout.

```
 >AC1=2500                      Start of subroutine
 >AC1.2=AC1+38E                 End of subroutine
 >DC1.2=XXXX                    Detect any data pattern

 >WHEN AC1 THEN TGR, GRO 2      Go to Group 2 when subroutine is entered
 >2 WHEN DC1 THEN TGR           Trigger during all cycles while in Group 2
 >2 WHEN AC1 THEN GRO 1         Go back to Group 1 when last instruction
 >RUN                           in subroutine is executed.
R>
```

(continued)

Special Interrupts

```
Registers:

        SIA                     Value Type - 32 Bit Integer

Events:

        FSI
```

The force special interrupt action, **FSI**, provides a way to jump to a specified address when a specific event is detected.

The special interrupt address register, SIA, should be set prior to entering run mode if you are using the **FSI** event. It defines the address your program vectors to when the **FSI** is executed.

When an **FSI** event is detected, an `FSI ACTIVE` message is displayed on the screen. You may also see some unusual cycles in the trace memory at the address where the **FSI** occurred. These are internal cycles that get traced as the execution address is changed. They are not purged from the trace memory because of the need for speed when executing an **FSI**.

The **FSI** event can be used as a fast way to make a patch to your code. It is also used to write soft shutdown routines for machinery that cannot be halted using a simple breakpoint.

The **FSI** routine residing at the SIA address should terminate with a return from exception **(RTE)** instruction. Execution will resume 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 **RTE** instruction.

Examples

Make a patch using overlay memory.

```
 >MAP 1000
 >AC1=8F36
 >WHEN AC1 THEN FSI
 >SIA=1000
 >ASM SIA                           Single line assembler - patch code
     .                              can be assembled here.
     .
     .
 >RUN
R>
```

Assume the program needs to break at a certain address, but the machine cannot be turned off until a soft shutdown routine is executed. Set SIA to the address of the soft shutdown routine. Use an **FSI** action at the break address, then set a breakpoint at the end of the soft shutdown routine.

```
 >SIA='SHUT_down
 >AC1=$7F4E2
 >AC2='SHUT_down + 4E
 >WHEN AC1 THEN FSI
 >WHEN AC2 THEN BRK
 >RBK
R>
```

Changing Event Groups

The four event groups provide a way to detect sequential events. When emulation is entered, event monitoring always begins in group 1. The example below describes a common use of the EMS group structure.

You may wish 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, you would 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
>'Module_B =8750 LEN 408
>'Sub_X =8934 LEN 56

>AC1='Module_A
>AC2='Module_B

>WHE AC1 OR AC2 THE GRO 2

>AC1.2='Sub_X

>2 WHEN AC1 THE TRC
>2 WHE NOT AC1 THE GRO 1
```

The **TRC/TOT** and **CNT/TOC** actions interact in a specific way when event groups are switched. The following state transition tables describe the actions taken when each of the different event combinations are specified.

Previous Group	**New Group**		
	Nothing Specified	*TRC*	*TOT*
Nothing specified	Trace all cycles	Trace only qualified cycles	No trace until first TOT
TRC	Trace all cycles	Trace only qualified cycles	No trace until first TOT
TOT OFF (not tracing)	Trace all cycles	Trace only qualified cycles	No trace until first TOT
TOT ON (tracing)	Trace all cycles	Trace only qualified cycles	No trace until first TOT

Previous Group	**New Group**		
	Nothing Specified	*CNT*	*TOC*
Nothing specified	No cycles counted	Count only qualified cycles	No count until first TOC
CNT	No cycles counted	Count only qualified cycles	No count until first TOC
TOC OFF (not counting)	No cycles counted	Count only qualified cycles	No count until first TOC
TOC ON (counting)	No cycles counted	Count only qualified cycles	No count until first TOC

(continued)

Shortcuts for Setting Up

This information is intended as a helpful guide to aid in setting up the Event Monitor System. Please read all of this information before using it. You can use the examples as is or you may find another combination of commands to accomplish your task.

These examples assume you are familiar with macros, (page 5-117) and, for repetitive use, know how to save macros to the EEPROM (page 5-25). Though the following macros use GD0 and GD1, you may use any of the general purpose registers (page 5-87).

MODE 0

Mode 0 is relatively easy to set up when using the Event Monitor System. To set up the address comparators, simply use the lower 24 bits of the address. To set up the data comparators, use the following macro:

```
 _1=DC1=GDO>> #16
 GDO=XXXXXXXX (32 bit data pattern you are using)
 _1
```

Set GD0 to a 32 bit data pattern and execute macro #1. The data comparators will only utilize the upper 16 bits. This macro will shift the data pattern to the right 16 places and place the new value into DC1.

The Event Monitor System can now be set up. For example:

```
     >WHEN DC1 THEN BRK
```

MODE 1

Mode 1 uses the LSA comparator and the data comparators to set up a full 32 bit data value. Use the following macro to set up a data comparator:

```
_1=LSA=GD0;DC1=GDO>>#16
GDO=XXXXXXXX (32 bit data pattern you are using)
_1
```

Set GD0 to a 32 bit pattern and execute macro #1. The data comparator will utilize the upper 16 bits and the LSA comparator will utilize the lower 16 bits.

The Event Monitor System can now be set up. For example:

```
    >WHEN DC1 AND LSA THEN BRK
```

MODE 2

(Please refer to Mode 1 for setting up the data comparator.)

To set up the address comparator, you will need to use an address comparator and a status comparator.

```
_1=AC1=GD1;S1=((GD1>>#17)&$7F80) DC $807F
GD1=XXXXXXXX (32 bit data pattern you are using)
_1
```

This macro will place the lower 24 bits into AC1. Next, GD1 will be shifted 17 (decimal) times to align adddress bits 24-31 to S1 bits 7-14. Next, it is "ANDED" with $7F80 to zero out the lower bits. Finally, a don't care mask is added so the S1 comparator will not expect to include bits 0-6 in the event comparison. This manipulated information will be placed into S1.

The Event Monitor System can now be set up. For example:

```
    >WHEN AC1 AND S1 THEN BRK
```

(continued)

To extend the address to a range, after the macro has been executed, use the following syntax:

```
AC1=AC1  LEN XXXXXX
```

You will not be able to extend the range to the upper 8 bits since they are residing in a status comparator. Ranges are not allowed in the status comparators.

To include any of the other status bits (i.e. BYT/WRD or TAR/OVL), after the macro has been executed, use the following syntax:

```
S1=S1+BYT
S1=S1+TAR+IP3
```

MODE 3

```
_1=AC1=GD0;LSA=((GD0>>#16)&$FF00) DC $FF
GD0=XXXXXXXX (32 bit data pattern you are using)
_1
```

This macro will place the lower 24 bits into AC1. Next, GD0 will be shifted 16 (decimal) times to align adddress bits 24-31 to LSA bits 8-15. Next, it is "ANDED" with $FF00 to zero out the lower bits. Finally, a don't care mask is added so the LSA comparator will not expect to include bits 0-7 in the event comparison. This manipulated information will be placed into the LSA comparator.

The Event Monitor System can now be set up. For example:

```
    >WHEN AC1 AND LSA THEN BRK
```

To extend the address to a range, after the macro has been executed, use the following syntax:

```
AC1=AC1  LEN XXXXXX
```

You will not be able to extend the range to the upper 8 bits, since they are residing in an LSA comparator. Ranges are not allowed in LSA comparators.

APPENDIX A

Table of Contents

ES Language Mnemonics

Page

ES LANGUAGE MNEMONICS

ES Language Commands

Mnemonic	Description	Page
>	emulation pause mode prompt	*4-27*
R>	emulation run mode prompt	*4-27*
RETURN	Return	*4-6*
/	repeat previous command line	*4-7*
;	statement separator	*4-6*
*	repeat command	*4-6*
CTRL Q	start screen scrolling (can be changed)	*4-31*
CTRL S	stop screen scrolling (can be changed)	*4-31*
CTRL X	delete line	*4-31*
CTRL R	reprint current line	*4-31*
CTRL Z	reset the emulator	*4-31*
ESC ESC	escape transparent	*4-31*
$	hexadecimal	*4-11*

Mnemonic	Description	Page
#	decimal	*4-11*
%	binary	*4-11*
\	octal	*4-11*
=	equals	*4-12*
()	parentheses	*4-8*
@	indirection	*4-9*
*	multiplication or repeat	*4-17*
/	division	*4-17*
+	addition	*4-17*
-	subtraction	*4-17*
-	negation	*4-16*
&	bitwise AND	*4-17*
^	bitwise OR	*4-17*
<<	shift left	*4-17*
>>	shift right	*4-17*
!	inverse, bitwise NOT	*4-17*
:	memory block attribute	*6-15*
.	increment Memory Mode address	*6-46*
.	execute macro 2	*5-110*
,	decrement Memory Mode address	*6-46*
,	execute macro 1	*5-110*
?	help menu	*4-20*
?	error query	*4-7*
_	define/execute macro	*5-88*
'	symbol definition (single quote)	*4-10*

Mnemonic	Description	Page
.B	dot operator, byte mode	*4-28*
.W	dot operator, word mode	*4-28*
.L	dot operator, long word mode	*4-28*
A(0-6)	address register (0-6)	*5-69*
ABS	absolute value	*4-17*
AC1, AC2	address comparators 1 and 2	*7-3, 8-3*
AND	logical event AND	*4-16*
ALL	status constant	*5-63*
ASM	line assembler	*6-34*
BAS	display base value	*5-79*
BFMT	bus error register	*5-75*
BMO	block move	*6-24*
BRK	break	*7-15, 8-11*
BTE	bus timeout enable switch	*5-12*
BUS	display status of lines	*6-82*
BYM	byte mode	*4-28*
BYT	byte access status	*7-5, 8-4*
CAAR	cache address (68020)	*5-70*
CACR	cache control (68020)	*5-70*
CAS	continuous address strobe switch	*5-13*
CCT	serial port control	*5-35*
CDS	cache disable switch (68020)	*5-14*
CES	clear WHEN/THEN statements	*7-14, 8-10*
CL	count limit	*7-8, 8-4*
CLK	read target system clock	*6-79*
CLM	clear memory map	*5-62*

Mnemonic	Description	Page
CLR	clear microprocessor data registers	*5-76*
CLT	clear trace memory (68020)	*5-102*
CMC	clear macros	*5-121*
CNT	count event	*7-20, 8-54*
COM	Communication with target	*5-47*
CPU	CPU space	*5-63*
CPY	copy switch	*5-15*
CRC	target cyclic redundancy check	*6-80*
CYC	cycle (68010)	*5-72*
D(0-7)	data registers (0-7)	*5-69*
DB	display memory block	*6-17*
DBP	disable bus error switch	*5-16*
DC	don't cares	*4-11*
DC1, DC2	data comparators 1 and 2	*7-4, 8-3*
DEL	delete symbol/section	*5-132*
DES	display WHEN/THEN statements	*7-12, 8-8*
DFB	default base register	*5-70, 5-85*
DFC	destination function code register (68010)	*5-69*
DIA	display ASCII character string	*5-51*
DIB	bus error register	*5-73, 5-75*
DIS	display disassembled memory	*6-39*
DM	display memory map	*5-54*
DNL	download	*5-38*
DOB	bus error register	*5-73, 5-75*

Mnemonic	Description	Page
DR	display microprocessor registers	*5-76*
DRT	display raw trace memory	5-94
DST	display raw trace memory	*5-108*
DT	disassemble trace memory	*5-103*
DTB	disassemble trace memory backward	*5-116*
DTF	disassemble trace memory forward	*5-116*
ECS	external cycle start switch (68020)	*5-17*
END	exit line assembler	*4-29*
FA	bus error register	*5-73, 5-75*
FIL	fill memory with constant data	*6-21*
FIN	find data in range	*6-19*
FMT	bus error register	*5-73*
FSI	force special interrupt	*7-25, 8-58*
FST	fast interrupt enable switch	*5-18*
FTO	fast bus timeout switch	*5-20*
GD0-7	general purpose data registers (0-7)	*5-70, 5-87*
GR0-7	general purpose range registers (0-7)	*5-70, 5-89*
GRO	event monitor system group	*7-27, 8-61*
IDX	repeat index register	*5-70, 5-113*
IIB	bus error register	*5-73*
ILG	illegal memory access attribute	*5-56*
IM	introspective mode switch	*5-21*
IP(0-7)	interrupt levels	*7-5, 8-4*
IPS	bus error register	*5-75*
IR(1-16)	bus error registers	*8-4*
IRA	internal range	*4-11*

Mnemonic	Description	Page
IR(A-G)	bus error registers	*5-75*
ISP	interrupt stack pointer	*5-70*
LD	load EEPROM data	*5-27*
LDV	load vectors	*6-9*
LEN	length	*4-11*
LIM	repeat limit register	*5-70, 5-113*
LOV	load overlay memory	*5-65*
LSA	Logic State Probe comparator	*7-4, 8-3*
LST	decrements address in memory mode	*6-46*
LWM	long word mode	*6-15*
MAC	display macros	*5-118*
MAP	define overlay memory map	*5-55*
M	enter memory mode	*6-42*
MMD	access status register	*5-70, 5-81*
MMP	memory mode pointer	*5-70, 5-83*
MMS	memory mode status	*5-70, 5-81*
MOD	modulo	*4-17*
MSK	bus error register (68010)	*5-73*
MSP	master stack pointer (68020)	*5-70*
NOT	logical event NOT	*4-15*
NXT	increment address in memory mode	*6-46*
OCS	operand cycle start	*5-17*
OFF	disable switches	*5-9*
ON	enables switches	*5-9*
OR	logical event OR	*4-16*

Mnemonic	Description	Page
OVE	overlay memory enable	*5-63, 5-70*
OVL	overlay memory access	*7-5, 8-4*
OVO	MMS/MMD attribute	*5-81*
OVS	overlay memory speed	*5-67*
PC	program counter register	*5-69, 5-73, 5-75*
PPT	peek/poke trace switch	*5-22*
PUR	clear symbolic memory	*5-133*
R(0-14)	bus error registers	*5-73*
RBK	run with breakpoints	*6-5*
RBV	load reset vectors and run breakpoints	*6-5*
RCT	reset count limit	*7-20, 8-54*
RD	read access status	*7-5, 8-4*
RET	execute line feed and return	*5-136*
REV	display firmware revision	*5-135*
RNV	run with new vectors	*6-5*
RO	read only attribute	*5-56*
RST	reset target system	*6-14*
RUN	run emulation	*6-5*
RW	read/write attribute	*5-55*
S1, S2	status comparators 1 and 2	*7-5, 8-4*
SAV	save EEPROM data	*5-25*
SBA	bus error register	*5-75*
SC(0-7)	space code status	*7-5, 8-4*
SD	supervisor data	*5-63*
SEC	display section	*5-129*

Mnemonic	Description	Page
SET	set/display system parameters	*5-3*
SF(0-3, 10-17, 20, 40-49)	special functions	*6-50*
SFC	source function code register (68010)	*5-69*
SIA	special interrupt address	*7-25, 8-60*
SLO	slow interrupt enable switch	*5-18*
SP	supervisor program	*5-63*
SPD	view bus speed info switch	*5-23*
SR	status register	*5-69, 5-73, 5-75*
SSP	supervisor stack pointer register	*5-69*
SSW	bus error register	*5-73, 5-75*
STA	search raw trace for address	*5-114*
STD	search raw trace for data	*5-114*
STS	search raw trace for status	*5-114*
STP	step and stop	*6-8*
SYM	display symbols	*5-128*
SZ	value display	*4-28*
TAD	tri-state address switch	*5-24*
TAR	target access	*5-113*
TCE	trace acquisition switch	*5-24*
TCT	terminal control	*5-34*
TGO	MMS/MMD attribute	*5-81*
TGR	enable trigger output	*7-24, 8-58*
TGT	target system memory attribute	*5-56*
THE	then	*7-2, 8-2*
TIM	Select Timer Frequency	*5-107*

Mnemonic	Description	Page
TO	to	*4-11*
TOC	toggle counting	*7-21, 8-55*
TOT	toggle trace memory	*7-17, 8-12*
TRA	transparent mode	*5-32*
TRC	trace event	*7-17, 8-12*
TST	test register for repeats	*5-70, 5-91*
UD	user data	*5-63*
UP	user program	*5-63*
UPL	upload	*5-43*
UPS	upload symbols	*5-45*
USP	user stack pointer register	*5-69, 5-70*
VBL	verify block data	*6-22*
VBM	verify block move	*6-27*
VBR	vector base register (68010/20)	*5-69, 5-70*
VFO	verify overlay memory	*5-66*
VFY	verify serial data	*5-42*
VM	valid memory address status	*7-5, 8-4*
VP	valid peripheral address status	*7-5, 8-4*
WAI	wait	*6-11*
WDM	word mode	*4-27*
WHEN/THEN	WHEN/THEN Statements	*7-2, 8-2*
WR	write access status	*7-5, 8-4*
WRD	word access status	*7-5, 8-4*

Mnemonic	Description	Page
X	don't care	*4-11*
X	exit memory mode	*6-45*
XRA	external range	*4-11*

APPENDIX B

Table of Contents

Error Messages

ERROR MESSAGES

Error Messages

Following is a list of error messages that the Emulator may display due to target-type errors.

CRC ERROR IN BOOT	Indicates a PROM error on the memory control board, U32 located in the Emulator chassis has a PROM malfunction. Addresses 1FE000 to 1FFFFF (level 1) or E000 to FFFF (for testing) on the 6809 side.
CRC ERROR IN DISASSEMBLER	Indicates a PROM error on the Emulator card, U72. Addresses 1C000 to 1FFFF (level 1) or C000 to FFFF (for testing) on the 6809 side.
CRC ERROR IN ESL	Indicates a PROM error on the Emulator card, located in the Emulator chassis. Addresses 4000 to FFFF on the 6809 side.
EMULATOR CRC ERROR	Indicates a PROM error on the Emulator card. Addresses 8000 to FFFF on the 680XX side.

NOTE: A cyclic redundancy check (CRC) error message displays whenever an internal hardware error to the Emulator has occurred. A PROM that is not firmly seated or has one of its legs folded under may cause a CRC error. If after checking for this a PROM malfunction still exists, call the factory, as this may indicate a serious problem.

MEMORY ACCESS VIOLATION	Indicates that the target program has attempted to access an area of target that is mapped illegal (ILG). **DM** will help determine which areas are mapped as illegal. **DRT** will help determine where the program was making the access.
MEMORY WRITE VIOLATION	Indicates that the target program has tried to write to the RAM overlay in an area that is mapped read only (**RO**). **DM** will help determine which areas are mapped read only. **DRT** will help determine which address the program was writing to.
PROCESSOR HUNG	Indicates that the Emulator is forcing a break (a single step, stopping, or a breakpoint), but the target processor is not responding to the 6809. In the Emulator, this means the "run to pause" routine was interrupted. Issue a reset (CTRL-Z) and look at the RAW trace (**DRT**). In the 68000/08/10, a *PROCESSOR-HUNG* error can occur if an odd value is present in the **SSP** or **VBR** when the break occurred. When the Emulator goes into run mode, the **SSP** is forced to an even value. In the 68010 and 68020, VBR will also be

	forced to an even value. If the program forces the VBR register to an odd address when the Emulator attempts to go into pause mode, a *PROCESSOR HUNG* error occurs. However, if the stack register in the 68020 has been changed to an odd value during run, an *UNUSED VECTOR 010* message may display after a couple of single steps (see below).
UNUSED VECTOR XXX	Indicates a processor-generated error that the Emulator is unable to correct. The Emulator returns control to the user with a $\boxed{>}$ prompt to wait for further commands. The number refers to the vector offset listed in the Exception Vector Assignments from Motorola. Please refer to the appropriate Motorola *Microprocessor User's Manual*.
WARNING - NO TARGET VCC	Indicates that there is no target power. Target power is checked before data strobes; therefore data strobes may or may not be present. The operating system prompt returns after this message is displayed, permitting user input.

Data Strobe Messages

These messages display whenever the data strobes are halted for more than 220 ms, making the user aware of status conditions on the target system.

BUS TIMEOUT WAITING FOR DSACK	(68020) Indicates that the present bus cycle has not been terminated. Possible causes include program execution accessing non-existent memory or address space. This message does not necessarily indicate an error. If the bus cycle is terminated by the target after the message is displayed, program execution will continue. In the event the target does not terminate the cycle, use CTRL Z, then examine raw trace (using **DRT**) to determine the fault.
BUS TIMEOUT WAITING FOR DTACK	(68000/08/10) Indicates that the present bus cycle has not been terminated. Possible causes include program execution accessing nonexistent memory or address space. This message does not necessarily indicate an error. If the bus cycle is terminated by the target after the message is displayed, program execution will continue. In the event the target does not terminate the cycle, use CTRL Z, then examine raw trace (using **DRT**) to determine the fault.
DOUBLE BUS FAULT	Indicates that the processor is halted from the processor side due to a double bus fault.

(68008) Double bus faults may be caused by:

1. Stack memory in non-valid memory space - no DTACK returned.
2. Stack memory in valid memory but set to an ODD address: example: SSP=7FFFF should be SSP = 80000 or 7FFFE
3. Attempting to access non-valid memory.

EXCESSIVE DMA TIME	Indicates that the target bus request, bus grant, or bus grant acknowledge signals have been asserted for longer than 150 ms. This message does not necessarily indicate an error. If the other bus master releases the bus after this message is displayed, program execution will continue. In the event this condition continues in the target, use CTRL Z to break emulation.
NO BUS ACTIVITY	Indicates no data strobe activity. Possible causes include the processor executing a STOP instruction or, in a 68020 system, the processor is executing out of cache. This message indicates a status and does not necessarily indicate an error.
NO TARGET CLOCK	Indicates that there is no clock present at the probe tip. The target system is expected to provide a clock signal.

NO TARGET VCC	Indicates that there is no VCC present at the probe tip. The target system is expected to provide VCC via the probe tip. Check the VCC pin for power. The probe tip power is checked before data strobes. Therefore, data strobes may or may not be present as well as target power.
PROCESSOR EXECUTING STOP INSTRUCTION	For the 68000/08/10 processor only. See *NO BUS ACTIVITY.*
TARGET BUS ERROR IS LOW	Indicates that a bus error signal on the probe tip is held low by the target system. NOTE: This error requires a solid low from the target, whereas *BUS ERROR* and *DOUBLE BUS FAULT* both require the bus error pin to be negated.
TARGET HALT ASSERTED	Indicates that the halt has been asserted from the target side for longer than .5 seconds.
TARGET RESET ASSERTED	Indicates that the target reset has been asserted for longer than about .5 seconds.

APPENDIX C

Table of Contents

Serial Data Formats

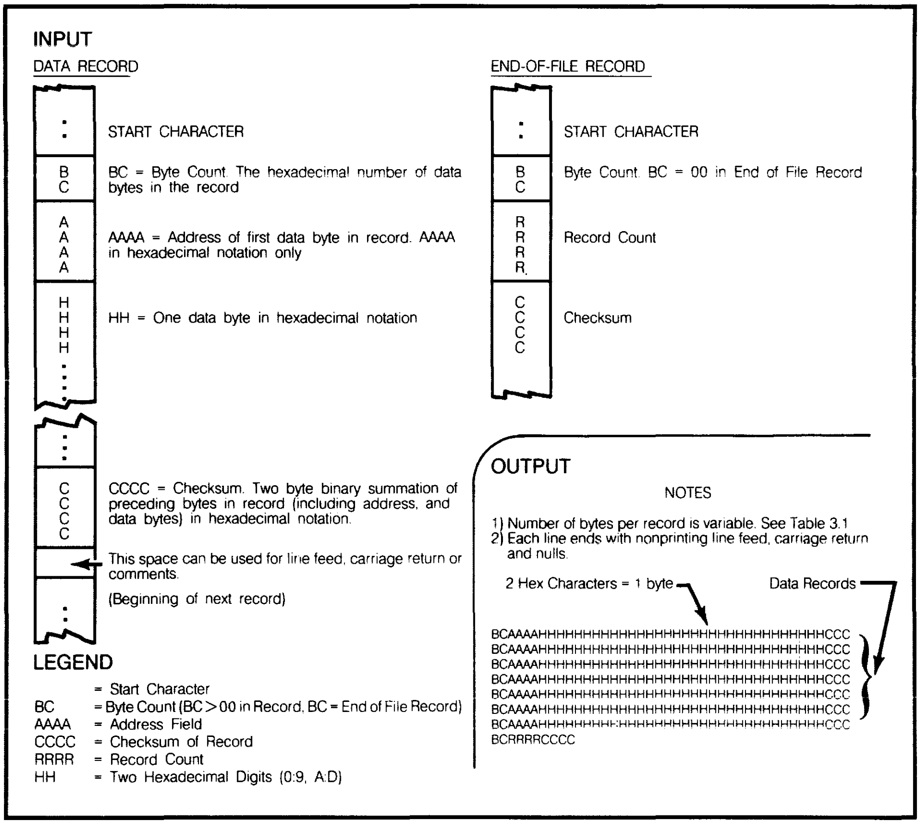
Appendix C

SERIAL DATA FORMATS

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

MOS Technology Format

Figure 14. Specifications for MOS Technology Data Files Copyright 1983, Data I/O Corporation; reprinted by permission.



INPUT

DATA RECORD

; START CHARACTER

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

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

HH = One data byte in hexadecimal notation

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

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

(Beginning of next record)

END-OF-FILE RECORD

START CHARACTER

Byte Count. BC = 00 in End of File Record

Record Count

Checksum

LEGEND

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

OUTPUT

NOTES

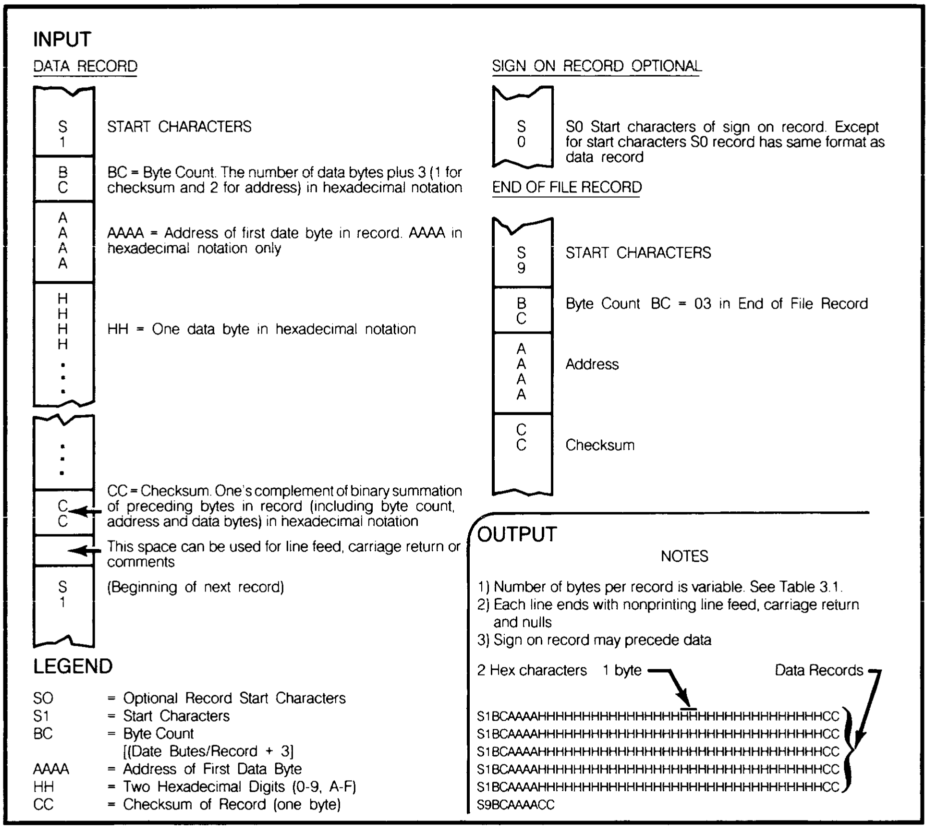
1) Number of bytes per record is variable. See Table 3.1
2) Each line ends with nonprinting line feed, carriage return and nulls.

2 Hex Characters = 1 byte

```
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCC
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCC
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCC
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCC
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCC
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCC
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCC
BCRRRRCCCC
```

Motorola Exorcisor Format

Figure 15. Specifications for Motorola Exorciser/16-BM Data Files/ Copyright 1983, Data I/O Corporation; reprinted by permission.



INPUT

DATA RECORD

START CHARACTERS

BC = Byte Count. The number of data bytes plus 3 (1 for checksum and 2 for address) in hexadecimal notation

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

HH = One data byte in hexadecimal notation

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

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

(Beginning of next record)

SIGN ON RECORD OPTIONAL

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

END OF FILE RECORD

START CHARACTERS

Byte Count BC = 03 in End of File Record

Address

Checksum

LEGEND

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

OUTPUT

NOTES

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

2 Hex characters 1 byte Data Records

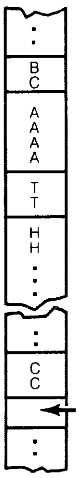
```
S1BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
S1BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
S1BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
S1BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
S1BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
S9BCAAAACC
```

Intel Intellec Format

Figure 16. Specifications for Intel Intellec/8/MDS Data Files/ Copyright 1983, Data I/O Corporation; reprinted by permission.

INPUT

DATA RECORD



START CHARACTER

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

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

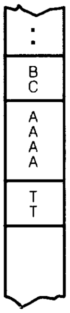
TT = Record Type (00)

HH = One data byte in hexadecimal notation

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

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

END OF FILE RECORD



START CHARACTER

Byte Count. BC = 00 in End of File Record

Address

TT Record Type (01)

LEGEND

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

OUTPUT

NOTES

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

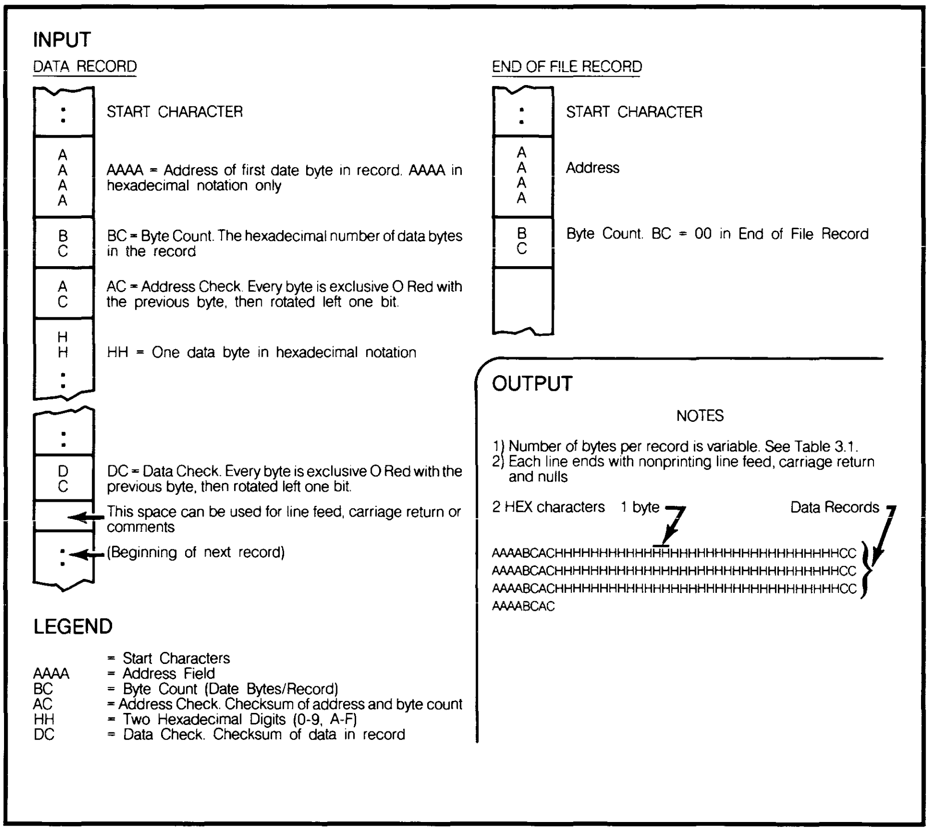
2 Hex characters 1 byte

Data Records

```
BCAAAATTHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
BCAAAATTHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
BCAAAATTHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
BCAAAATT
```

Signetics/Absolute Object File Format

Figure 17. Specifications for Signetics/Absolute Object Data Files Copyright 1983, Data I/O Corporation; reprinted by permission.



INPUT

DATA RECORD

: START CHARACTER

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

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

AC = Address Check. Every byte is exclusive O Red with the previous byte, then rotated left one bit.

HH = One data byte in hexadecimal notation

DC = Data Check. Every byte is exclusive O Red with the previous byte, then rotated left one bit.

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

(Beginning of next record)

END OF FILE RECORD

: START CHARACTER

AAAA Address

BC Byte Count. BC = 00 in End of File Record

LEGEND

:	= Start Characters
AAAA	= Address Field
BC	= Byte Count (Date Bytes/Record)
AC	= Address Check. Checksum of address and byte count
HH	= Two Hexadecimal Digits (0-9, A-F)
DC	= Data Check. Checksum of data in record

OUTPUT

NOTES

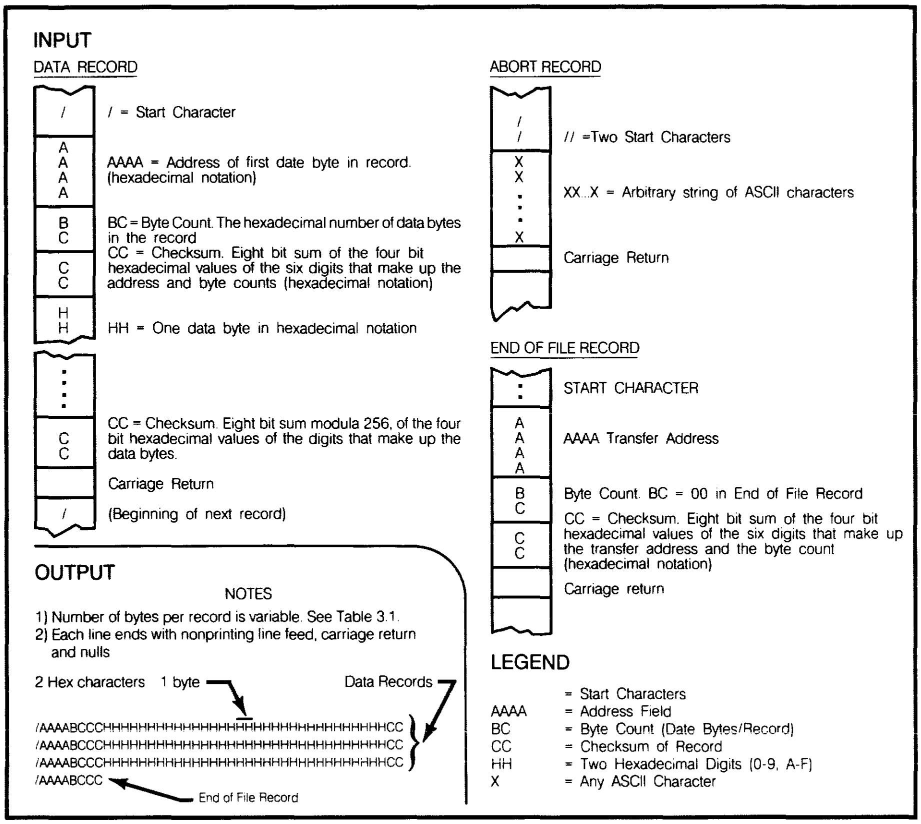
1) Number of bytes per record is variable. See Table 3.1.
2) Each line ends with nonprinting line feed, carriage return and nulls

2 HEX characters 1 byte   Data Records

```
AAAABCACHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
AAAABCACHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
AAAABCACHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
AAAABCAC
```

Tektronix Hexadecimal Format

Figure 18. Tekhex/Copyright 1983, Tektronix; reprinted by permission



INPUT

DATA RECORD

/ = Start Character

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

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

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

HH = One data byte in hexadecimal notation

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

Carriage Return

(Beginning of next record)

ABORT RECORD

// =Two Start Characters

XX..X = Arbitrary string of ASCII characters

Carriage Return

END OF FILE RECORD

START CHARACTER

AAAA Transfer Address

Byte Count. BC = 00 in End of File Record

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

Carriage return

OUTPUT

NOTES

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

2 Hex characters 1 byte

Data Records

```
/AAAABCCCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
/AAAABCCCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
/AAAABCCCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
/AAAABCCC
```

End of File Record

LEGEND

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

Extended Tekhex Format

Copyright 1983, Tektronix; reprinted by permission

Extended Tekhex uses three types of message blocks:

1. The data block contains the object code.
2. The symbol block that contains information about a program section and the symbols associated with it. This information is only needed for symbolic debug.
3. The termination block contains the transfer address and marks the end of the load module.

NOTE

Extended Tekhex has no specially defined abort block. To abort a formatted transfer, use a Standard Tekhex abort block.

Each block begins with a six-character header field and ends with an end-of-line character sequence. A block can be up to 255 characters long, not counting the end-of-line character. The header field has the format shown in the following table.

ITEM	NUMBER OF ASCII CHARACTERS	DESCRIPTION
%	1	A permit sign specified that the block is in Extended Tekhex format.
Block Length	2	The number of characters in the block: a two-digit hex number. This count does not include the leading % or the end-of-line.
Block Type	1	6 = data block 3 = symbol block 8 = termination block
Checksum	2	A two-digit hex number representing the sum, mod 256, of the values of all the characters in the block, except the leading %, the checksum digits, and the end-of-line. The following table gives the values for all characters that may appear in Extended Tekhex message blocks.

Character Values for Checksum Computation

CHARACTERS	VALUES (DECIMAL)
0..9	0..9
A..Z	10..35
$	36
%	37
. (period)	38
_ (underscore)	39
a..z	40-65

VARIABLE-LENGTH FIELDS

In Extended Tekhex, certain fields may vary in length from 2 to 17 characters. This practice enables you to compress your data by eliminating leading zeros from numbers and trailing spaces from symbols. The first character of a variable-length field is a hexadecimal digit that indicates the length of the rest of the field. The digit 0 indicates a length of 16 characters.

For example, the symbols **START**, **LOOP**, and **KLUDGESTARTSHERE** are represented as **5START**, **4LOOP**, and **0KLUDGESTARTSHERE**. The values **0**, **100H**, and **FF0000H** are represented as **10**, **3100**, and **6FF0000**.

DATA AND TERMINATION BLOCKS

If you do not intend to transfer program symbols with your object code, you do not need symbol blocks. Your load module can consist of one or more data blocks followed by a termination block. The following table gives the format of a data block and a termination block.

Extended Tekhex Data Block Format

ITEM	# OF ASCII CHARACTERS	DESCRIPTION
Header	6	Standard header field Block Type = 6
Load Address	2 to 17	The address where the object code is to be loaded: a variable-length number.
Object	2n	n bytes, each represented as two hex digits.

Extended Tekhex Termination Block

Header	6	Standard header field Block type = 8.
Transfer Address	2 to 17	The address where program execution is to begin: a variable-length number.

SYMBOL BLOCKS

A symbol used in symbolic debug has the following attributes:

1. The symbol itself: 1 to 16 letters, digits, dollar signs, periods, a percent sign, or symbolize a section name. Lower case letters are converted to upper case when they are placed in the symbol table.

2. A value: up to 64 bits (16 hexadecimal digits).

3. A type: address or scalar. (A scalar is any number that is not an address.) An address may be further classified as a code address (the address of an instruction) or a data address (the address of a data item). As symbolic debug does not currently use the code/data distinction, the address/scalar distinction is sufficient for standard applications of Extended Tekhex.

4. A global/local designation. This designation is of limited use in a load module, and is provided for future development. If the global/local distinction is not important for your purposes, simply call all your symbols global.

5. Section membership. A section may be thought of as a named area of memory. Each address in your program belongs to exactly one section. A scalar belongs to no section.

The symbols in your program are conveyed in symbol blocks. Each symbol block contains the name of a section and a list of the symbols that belong to that section. (You may include scalars with any section you like.) More than one block may contain symbols for the same section. For each section, exactly one symbol block should contain a section definition field, which defines the starting address and length of the section.

If 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 Block Format

ITEM	NUMBER OF ASCII CHARACTERS	DESCRIPTION
Header	6	Standard header field Block Type = 3
Section Name	2 to 17	The name of the section that contains the symbols defined in this block: a variable-length symbol.
Section Definition	5 to 35	This field must be present in exactly one symbol block for each section. This field may be preceded or followed by any number of symbol definition fields. The table on the next page gives the format for this field.
Symbol	5 to 35	Zero or more symbol definition fields as described in the next table.

Extended Tekhex Symbol Block: Section Definition Field

ITEM	NUMBER OF ASCII CHARACTERS	DESCRIPTION
0	1	A zero signals a section definition field.
Base	2 to 17	The starting address of the Address section: a variable-length number.
Length	2 to 17	The length of the section: a variable-length number, computed as 1 + (high address base address).

Extended Tekhex Symbol Block: Symbol Definition Field

ITEM	NUMBER OF ASCII CHARACTERS	DESCRIPTION
Type	1	A hex digit that indicates the global/local designation of the symbol, and the type of value the symbol represents: 1 = global address 2 = global scalar 3 = global code address 4 = global data address 5 = local address 6 = local scalar 7 = local code address 8 = local data address
Symbol	2 to 17	A variable-length symbol.
Value	2 to 17	The value associated with the symbol: a variable-length number.

The following figures show how the preceding tables of information might be encoded in Extended Tekhex. The information for the Extended Tekhex Symbol Block illustration could be encoded in a single 96-character block. It is divided into two blocks for purposes of illustration.

Figure 13. Extended Tekhex Data Block

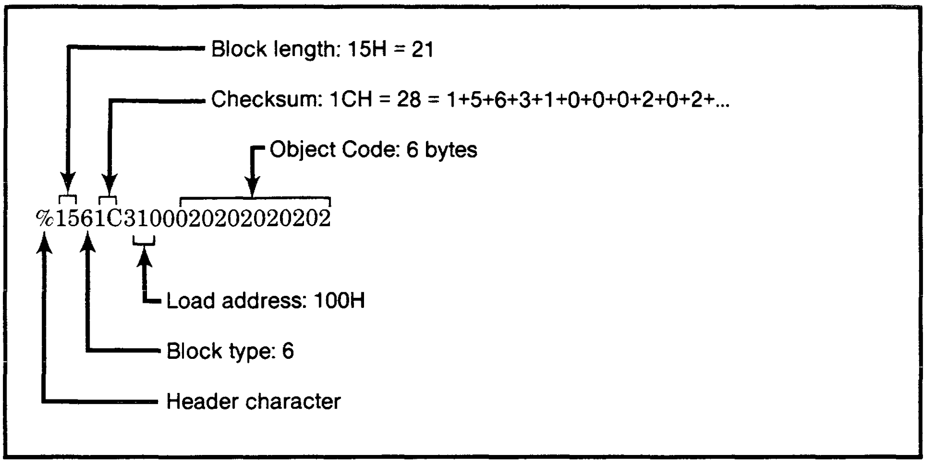


Figure 14. Extended Tekhex Termination Block

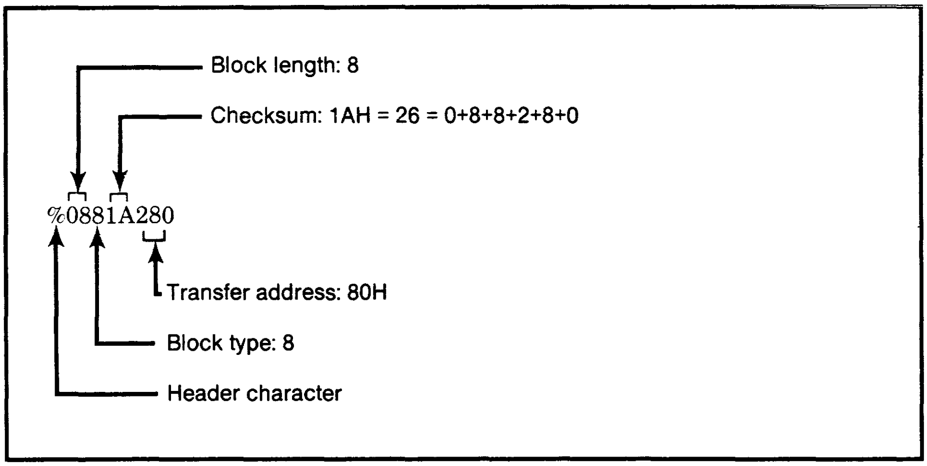
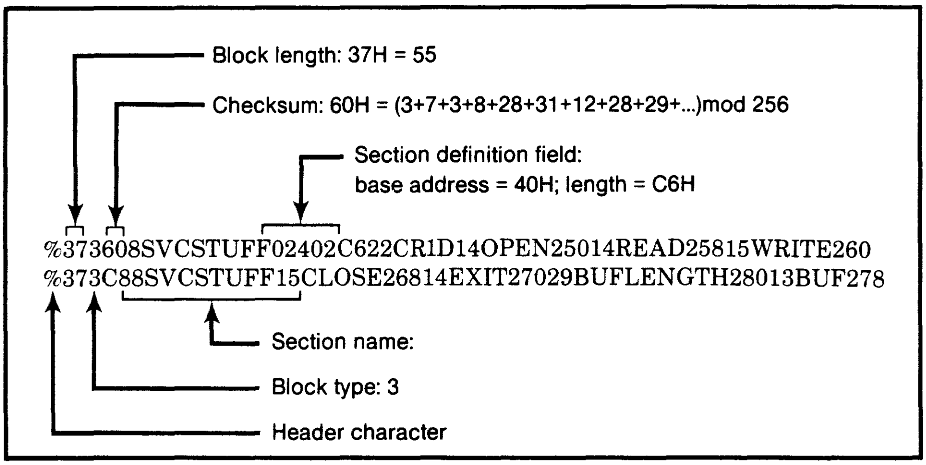


Figure 15. Extended Tekhex Symbol Block



```
%373608SVCSTUFF02402C622CR1D14OPEN25014READ25815WRITE260
%373C88SVCSTUFF15CLOSE26814EXIT27029BUFLENGTH28013BUF278
```

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
Section name:
Block type: 3
Header character

Motorola S-Record Format

S-RECORD CONTENT

When viewed by the user, S-records are essentially character strings made of several fields which identify the record type, record length, memory address, code/data, and checksum. Each type of binary data is encoded as a 2-character hexadecimal number: the first character representing the high-order 4 bits, and the second the low-order 4 bits of the byte.

The 5 fields which comprise an S-record are: type, length, address, code/data and checksum.

The fields are composed as follows:

FIELD	PRINTABLE CHARACTERS	CONTENTS
type	2	s-record type -- S0, S1, etc.
record length	2	The count of the character pairs in the record, excluding the type and record length.
address	4, 6, or 8	The 2-, 3-, or 4-byte address at or which the data field is to be loaded into memory.
code/data	0-2n	From 0 to n bytes of executable code, memory-loadable data, or descriptive information. For compatibility with teletypewriters, some programs may limit the number of bytes to as few as 28 (56 printable characters in S-record).
checksum	2	The least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields.

Each record may be terminated with a CR/LF/NULL. Additionally, an S-record may have an initial field to accommodate other data such as line numbers generated by some time-sharing systems.

Accuracy of transmission is ensured by the record length (byte count) and checksum fields.

S-RECORD TYPES

Eight types of S-records have been defined to accommodate the several needs of the encoding, transportation, and decoding functions. The various Motorola upload, download, and other file-creating or debugging programs, utilize only those S-records which serve the purpose of the program. For specific information on which S-records are supported by a particular program, the user's manual for that program must be consulted.

An S-record format module may contain S-records of the following types:

S0	The header record for each block of S-records. The code/data field may contain any descriptive information identifying the following block of S0-records. Under VERSAdos, the resident linker's IDENT command can be used to designate module name, version number, revision number, and description information which will make up the header record. The address field is normally zeros.
S1	A record containing code/data and the 2-byte address at which the code/data is to reside.
S2	A record containing code/data and the 3-byte address at which the code/data is to reside.
S3	A record containing code/data and the 4-byte address at which the code/data is to reside.
S5	A record containing the number of S1, S2, and S3 records transmitted in a particular block. This count appears in the address field. There is no code/data field.
S7	A termination record for a block of S3 records. The address field may optionally contain the 3-byte address of the instruction to which control is to be passed. There is no code/data field.

S8	A termination record for a block of S2 records. The address field may optionally contain the 3-byte address of the instruction to which control is to be passed. There is no code/data field.
S9	A termination record for a block of S1 records. The address field may optionally contain the 2-byte address of the instruction to which control is to be passed. Under VERSAdos, the resident linker's ENTRY command can be used to specify this address. If not specified, the first entry point specification encountered in the object module input will be used. There is no code/data field.

Only one termination record is used for each block of S-records. S7 and S8 records are usually used only when control is to be passed to a 3- or 4- byte address. Normally, only one header record is used, although it is possible for multiple header records to occur.

CREATION OF S-RECORDS

S-record-format programs may be produced by several dump utilities, debuggers, VERSAdos' resident linkage editor, or several cross assemblers or cross linkers. ON EXORmacs, the Build Load Module (MBLM) utility allows an executable load module to be built from S-records; and has a counterpart utility in BUILDS, which allows an S-record file to be created from a load module.

Several programs are available for downloading a file in S-record format from a host system to an 8-bit microprocessor-based or 16-bit microprocessor-based system. Programs are also available for uploading an S-record file to or from an EXORmacs system.

Example: Shown below is a typical S-record-format module, as printed or displayed:

```
S00600004844521B
S1130000285F245F2212226A000424290000082337CA
S11300100002000080008262900185381234100181З
S113002041E9000084E42234300182342000824A952
S107003000144Ed492
S9030000FC
```

The module consist of one S0 record, four S1 records, and an S9 record.

The S0 record is comprised of the following character pairs:

S0	S-record type S0, indicating that it is a header record.
06	Hexadecimal 06 (decimal 6), indicating that six character pairs (OR ASCII bytes) follow.
00+	
00	Four-character 2-byte address field, zeros in this example.
48	
44+	ASCII H, D, and R - "HDR".
52	
1B	The checksum.

The first S1 record is explained as follows:

S1	S-record type S1, indicating that it is a code/data record to be loaded/verified at a 2-byte address.
13	Hexadecimal 13 (decimal 19), indicating that 19 character pairs, representing 19 bytes of binary data, follow.
00+	Four-character 2-byte address field; hexadecimal address
00	0000, where the data which follows is to be loaded.

The next 16 character pairs of the first S1 record are the ASCII bytes of the actual program code/data. In this assembly language example, the hexadecimal opcodes of the programs are written in sequence in the code/data fields of the S1 records:

OPCODE	INSTRUCTION	
285F	MOVE.L	(A7) +,A4
245F	MOVE.L	(A7) +,A2
2212	MOVE.L	(A2),D1
226A0004	MOVE.L	4(A2),A1
24290008	MOVE.L	FUNCTION(A1),D2
237C	MOVE.L	#FORCEFUNC,FUNCTION(A1)
o	(The balance of this code is continued in the code/data fields of the remaining S1 records, and stored in memory location 0010, etc.)	
2A	The checksum of the first S1 record.	

The second and third S1 records each also contain $13 (19) character pairs and are ended with checksums 13 and 52 respectively. The fourth S1 record contains 07 character pairs and has a checksum of 92.

The S9 record is explained as follows:

S9	S-record type S9, indicating that it is a termination record.
03	Hexadecimal 03, indicating that three character pairs (3 bytes) follow.
00	The address field, zeros.
FC	The checksum of the S9 record.

Each printable character in an S-record is encoded in hexadecimal (ASCII in this example) representation of the binary bits which are actually transmitted.

SUBJECT INDEX

Applied
Microsystems
Corporation

Applied Microsystems Corporation maintains a worldwide network of direct sales offices committed to quality service and support. For information on products, pricing, or delivery, please call the nearest office listed below. If you are unsure which office to contact, call 1-800-426-3925 for assistance.

CORPORATE OFFICE
Applied Microsystems Corporation
5020 148th Avenue Northeast
P.O. Box 97002
Redmond, WA 98073-9702
(206) 882-2000
1-800-426-3925
TRT TELEX 185196
Fax (206) 883-3049

EUROPE
Applied Microsystems Corporation
Chiltern Court
High Street
Wendover
Aylesbury, Bucks
44 (0) HP22 6EP England
296-625462
Telex 265871 REF WOT 004
Fax 44 (0) 296-623460

JAPAN
Applied Microsystems Japan, Ltd.
Nihon Seimei
Nishi-Gotanda Building
7-24-5 Nishi-Gotanda
Shinagawa-Ku
Tokyo T141, Japan
3-493-0770
Fax 3-493-7270

U.S. REGIONAL SALES OFFICES
Western Region
Applied Microsystems
Corporation of Washington
3333 Bowers Avenue
Suite #220
Santa Clara, CA 95054
(408) 727-5433
Fax (408) 727-9011

Applied Microsystems
Corporation of Washington
2101 Business Center Drive
Suite #140
Irvine, CA 92715
(714) 476-3177
Fax (714) 476-8546

Central Region
Applied Microsystems Corporation
Suite #142
Richardson, TX 75081
(214) 235-8827
Fax (214) 238-0719

Eastern Region
Applied Microsystems
Corporation of Washington
6 Cabot Place
Stoughton, MA 02072
(617) 341-3121
Fax (617) 341-0245

P/N 922-00002-04
August 1988