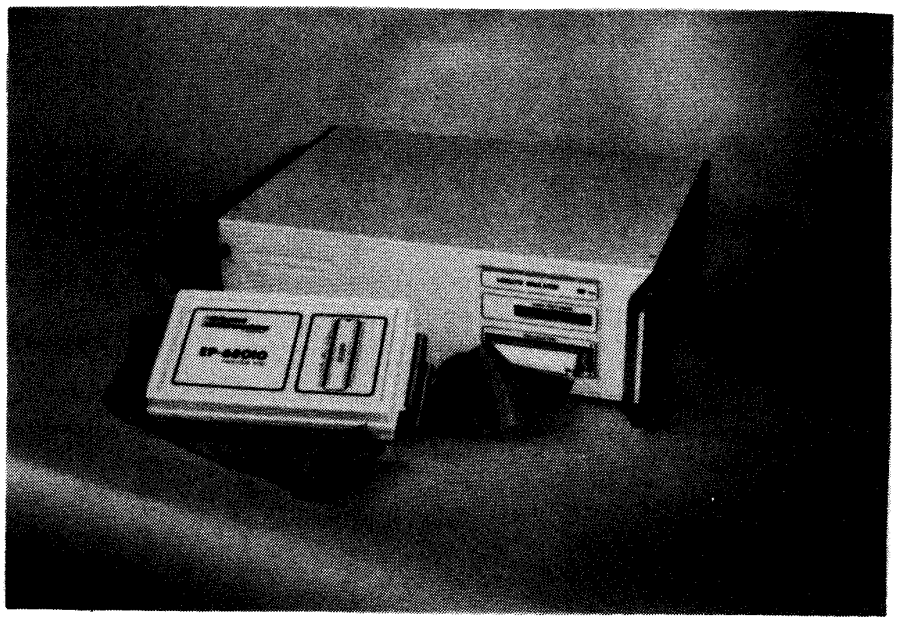


ES 1800 SATELLITE EMULATOR
OPERATOR'S MANUAL
FOR 68000/68008/68010 MICROPROCESSORS

Applied Microsystems CORPORATION

CHANGING YOUR IDEAS ABOUT EMULATION

5020 - 148th Avenue NE
P.O. Box C-1002
Redmond, WA 98073-1002
(206) 882-2000
(800) 426-3925

August, 1984

920-11435-00

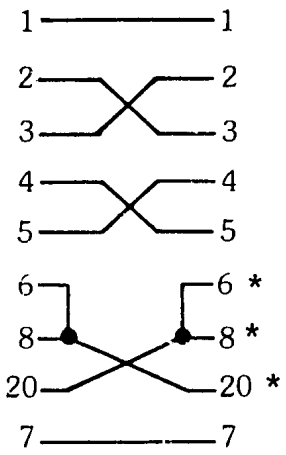
SET-UP CHECKLIST
68000/68008/68010

Please read this checklist completely before using your new Applied Microsystems' Satellite Emulator.

1. **Are you going to operate in static or dynamic RAM?** If you are using dynamic RAM, check the ON-OFF switches. See CAS and TAD, page 3-19.

2. **Have you reviewed the specifications for the serial interface port?** See Sections 2.3.3 and 2.3.4.

3. **If using communications without a modem, you may need a null modem cable.** If you purchase 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.

4. **You may wish to protect the 64-pin adapter on the Probe Tip Assembly** by installing a low-cost, round-pin CPU socket (male-female) onto the adapter. If a pin is then broken on the CPU socket, it is easier to replace because of its common usage.

5. At a minimum, **you should review sections applicable to the steps listed here, plus:**

   - **Section 1—Introduction**
   - **Section 2—Installation and Set-up** (safety information, Help menus, sample first-time emulation sequences, etc.).

If you experience difficulty in setting up your Satellite Emulator, call Customer Service for ES Products at 1-800-426-3925.

PLACE CHECKLIST INSIDE FRONT COVER FOR FUTURE REFERENCE

5020 148th AVenue NE
P.O. Box C-1002
Redmond, WA 98073-1002
Toll Free Service: 1-800-426-3925

Applied Microsystems CORPORATION

August, 1984

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

MEMORY - REGISTER COMMANDS:             DR - DISPLAY ALL CPU REGISTERS
  DB(.BWL) 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
  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 = 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

MEMORY MAPPING:                         OVE = SP+SD+UP+UD+CPU; OVS = 0-7
  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 + IP0 .. IP7 + VP + VM + BER
  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   2 WHEN AC1 AND NOT DC2 THEN CNT
  CL.2 = 24;AC2.2 = $F000 LEN $400          WHEN CL.2 OR AC2.2 THEN BRK
```

Applied Microsystems Corporation has made every effort to document this product accurately and completely. However, Applied Microsystems assumes no liability for errors or for any damages that result from use of this manual or the equipment it accompanies. Applied Microsystems reserves the right to make changes to this manual without notice at any time.

Information that is unique to the MC68010 is marked with a solid bar (▮) in the margin.

TABLE OF CONTENTS

SECTION 4. OPERATION

SECTION 5. PROGRAMMING THE EVENT MONITOR SYSTEM

QUICK INDEX TO OPERATORS

LIST OF FIGURES

LIST OF TABLES

LIST OF EXAMPLES

Most examples occur within the section that discusses the operator in the example. The only examples listed here are those that have been placed separately from their section because more than one operator is illustrated.

SECTION 1
INTRODUCTION

1.1 SYSTEM CONCEPT

The Applied Microsystems ES 1800 Satellite Emulator (See Fig. 1-1.) is a controllable microprocessor emulation system. It operates in conjunction with your host computer system or as a standalone system controlled by a CRT terminal. All system configurations provide powerful hardware and software debugging capability as well as hardware/software integration support.

The Satellite Emulator is transparent to the normal operation of the "target system" (your hardware). Emulation is performed in real time - no additional microprocessor cycles are required as a result of the emulation process. No target system addresses or I/O ports are needed or used and no program or software objects are required in the target system address space. There are no hidden quirks. You will have no difficulty using the Satellite Emulator with your target system, even when critical timing constraints are present. The emulator operates at speeds up to the specified clock rate and will also single-step the microprocessor under program or operator control.

Standard features of the Satellite Emulator include an Event Monitor System, Trace Memory and Disassembly and special test functions.

1.1.1 Components

The Satellite Emulator consists of a mainframe, an emulator pod assembly, a Null Target Software Simulation Tool, and an optional Logic State Analysis pod assembly.

MAINFRAME. The mainframe houses the emulation control board, the memory controller board, the RAM Overlay board, the controller board, the trace and break board, and the power supply, as shown in Figure 1-2. There are no external panel controls except the power switch on the rear panel. The emulation control board configures the Satellite Emulator for use with specific microprocessors. It resides in the mainframe and contains the electronics unique to the specific device it emulates.

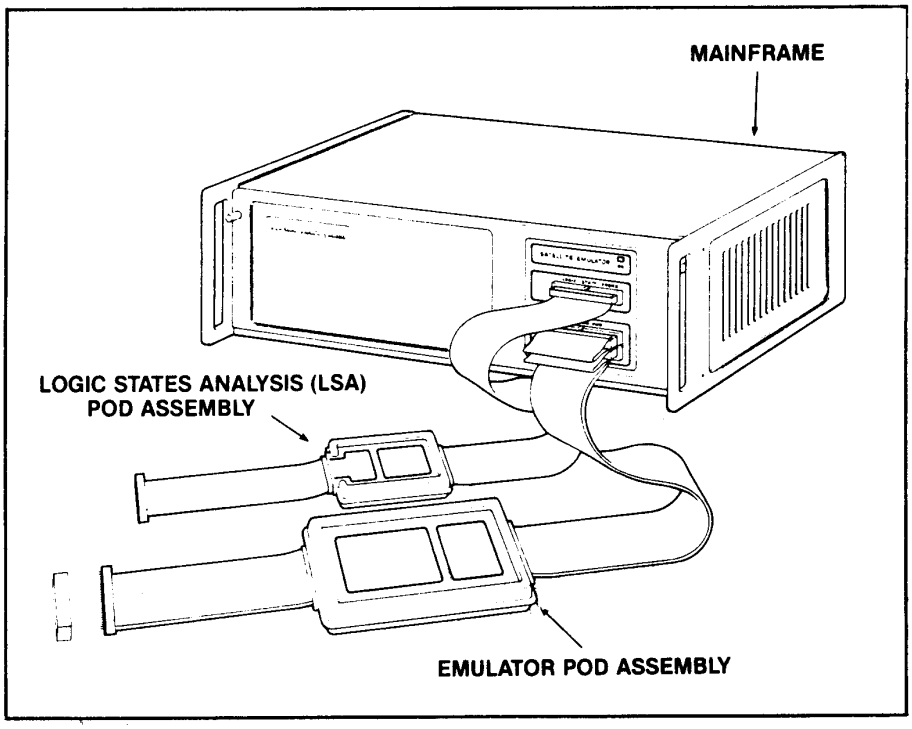


Figure 1-1.
The Satellite
Emulator

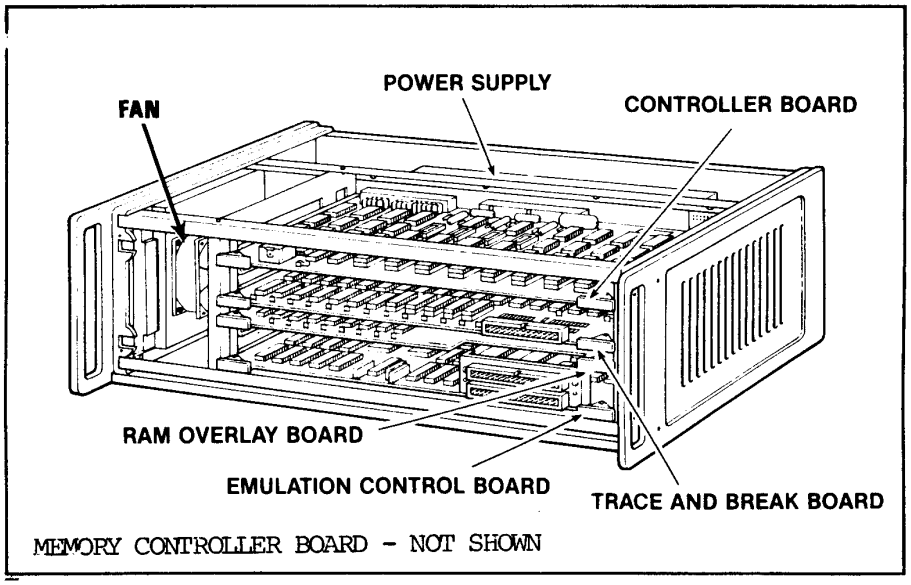


Figure 1-2. Mainframe Components

EMULATOR POD ASSEMBLY. The emulator pod assembly consists of the pod, a probe and two cables:

- The 40-inch ribbon cable connects the assembly to the mainframe; the 11-inch ribbon cable connects the assembly to the target system.
- The pod contains the emulating microprocessor and associated circuitry (line buffers, etc.).
- A dual in-line package (DIP) connector on the probe plugs into the target system's microprocessor chip socket.

The emulator pod assembly is connected internally to the mainframe via the emulation control board (see Figure 1-2).

OPTIONAL LOGIC STATE ANALYZER POD ASSEMBLY. The Logic State Analyzer (LSA), via the optional LSA pod assembly, provides 16 additional input lines to the Satellite Emulator, giving you access to signals other than the bus signals.

1.1.2 The Target System

The target system is your hardware. The emulator pod assembly is connected to the target system by removing the target system microprocessor from its socket and plugging the probe connector in its place. The emulator then functions as a replacement for the microprocessor that was removed, providing a rich variety of control and analysis capabilities at the same time.

Once connected, the emulator is able to communicate with the environment that the target system provides for the target system microprocessor; the emulator may read or write to the microprocessor registers or memory locations and it may execute programs contained in the target system memory. It makes no assumptions about the environment provided by the target system; if the target system microprocessor works correctly with the target system, the emulator will also, provided that the microprocessor manufacturer's design specifications are complied with.

1.1.3. The Host System

The host system may be a development system, minicomputer, or automatic test equipment system. The Satellite Emulator connects to a host system via a serial port (labeled "COMPUTER") on the rear panel of the emulator mainframe. A second serial port (labeled "TERMINAL") is provided for connection to a CRT terminal.

The host system can be used to control the emulator or as a source of data. This is described in section 1.1.4.

1.1.4 System Configurations

There are two system configurations: standalone, and standalone with host data files. See Figure 1-3.

STANDALONE. In this configuration, the Satellite Emulator is controlled directly by a CRT terminal, with no external data sources or output devices. The terminal serial port on the rear panel is the input source for control commands you key in on a CRT terminal. See Figure 1-3a.

STANDALONE WITH HOST DATA FILES. In this configuration, the Satellite Emulator is still under the direct control of the CRT terminal. In addition, the computer serial port is connected to a host system for access to the host's data files. Or, the computer serial port can be connected to a printer for dumping data from the emulator to create hard copies. You also have available a "transparent mode," wherein the Satellite Emulator allows communication between the computer and terminal ports or output devices connected to these other ports. Essentially, the transparent mode uses the emulator as an interface or conduit between the two ports. See Figure 1-3b.

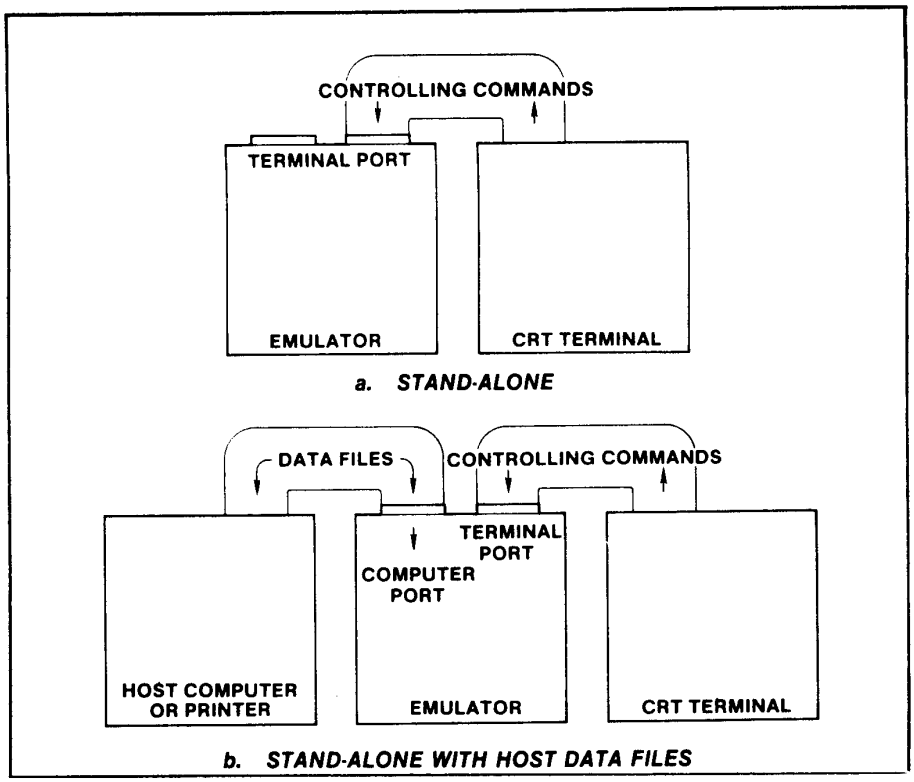


Figure 1-3.
System Configurations

1.1.5 System Features

Table 1-1 summarizes the system features of the emulator. These features can be combined in various ways to form an emulation system that fulfills your exact needs. Section 3 gives a more detailed description of how these are combined.

Table 1-1. Feature Summary

FEATURE	DESCRIPTION
Help Menu	Provides you with a display of examples on a CRT terminal. See Section 2.
EEPROM Storage	Allows two users to store complete sets of unique, user-defined operating parameters; interface parameters, register values, switch settings, Event Monitor System parameters, and the memory map. Parameters can be accessed and changed at any time during an emulation session. See Section 2.
Emulation	Lets the emulator become the target system microprocessor and execute the program and functions of the target system. See Section 4.
Trace Memory	Functions as a history of target system program execution. It records each bus cycle and can output to a display the last 2046 machine cycles. See Section 4.
Disassembler	Allows you to display the contents of the Trace Memory history in a form similiar to your program listing. Output can be to your CRT terminal, a printer, or your host computer. See Section 4.
Event Monitor System	Allows you to specify event detectors that will cause specified actions to occur when the events are encountered in the target system program. Some of the available actions are: • break emulation • qualify trace data • increment or reset the pass counter • trigger an oscilloscope or other instrument • switch to other event detectors • interrupt to a user routine See Section 5.

Logic State Analyzer	Provides external logic signal recording (pod assembly option) and event detection capability (16 inputs to a 16 x 2048-bit memory). See Section 5.
Overlay Memory (options up to 512K-byte total)	Memory, locatable in 2K-byte segments, that can be mapped into the address space of the target system. When a portion of the target system program is loaded into it, the program can be edited, positioned as desired, and the program executed as if it resided completely in the target system. See Section 4.
Memory Mode	Provides fast and easy examination and modification to target system memory locations. See Section 4.
Null Target Software Simulation Tool	Allows you to execute your software without connecting the emulator to your target system. See Section 5.
Downloading	Loads target system memory space with data from a host system. See Section 6.
Uploading	Dumps data from the target system address space to one of the Satellite Emulator's serial ports. See Section 6.
Diagnostic Functions	A large number of diagnostic functions and routines that can be used in both engineering and manufacturing environments to turn on and test your microprocessor system hardware. Features include memory tests, oscilloscope synchronization, and signature analysis stimuli. See Section 7 for a complete list and detailed descriptions.

1.2 DOCUMENTATION

This manual gives you information necessary for setting up and operating the Satellite Emulator.

This first section of the manual introduces the Satellite Emulator and provides information on features, options, specifications, warranty, and service. The remaining sections are organized as follows:

- Section 2, Installation and Set-Up: procedures for setting up the physical connection, interface parameters, initial checkout of the emulation system, pre-operational procedures for setting up the system, such as accessing the Help Menu and EEPROM storage of parameters and a sample first-time emulation sequence.

- Section 3, System Syntax and Parameters.
- Section 4, Operation: procedures for emulation, Memory Mode, Overlay Memory, Trace Memory, and error codes.
- Section 5, Programming the Event Monitor System: procedures for programming the Event Monitor System to your specific needs.
- Section 6, Interfacing and Communications: procedures for communicating between the Satellite Emulator and other units in an emulation system, such as uploading and downloading and setting system controls.
- Section 7, Diagnostic Functions: descriptions of and procedures for using the built-in diagnostic functions of the Satellite Emulator.
- Section 8, Maintenance and Troubleshooting: procedures for routine maintenance and basic troubleshooting of the Satellite Emulator.
- Appendices: serial data formats, glossary, cross-reference of number bases.
- Index

1.3 68000 APPLICATIONS

Your Satellite Emulator is configured for 68000 family microprocessors with the appropriate emulation control board and emulator pod assembly. The following table lists the microprocessors currently supported by the ES series emulators and the emulation control board and emulator pod assembly used with each. New devices may be added as support becomes available. Contact your Applied Microsystems Corporation representative when you need additional support.

Table 1-2
Applications

DEVICE	EMULATION CONTROL BOARD	EMULATOR POD ASSEMBLY
Motorola:		
68000	ES-68000B	ES-68000P
68008	ES-68008B	ES-68008P
68010	ES-68010B	ES-68010P
Zilog:		
Z8000	ES-Z8000B	ES-Z8000P
Z8001	"	ES-Z8001P
Z8002	"	ES-Z8002P
Z8003	"	ES-Z8003P
Intel:		
8086	ES-8086B	ES-8086P-86
8088	"	ES-8086P-88
80186	"	ES-80186P-186
80188	"	ES-80186P-188

1.4 OPTIONS

The following options are available for your emulator. Contact your Applied Microsystems Corporation representative for information on prices and ordering.

- Overlay Memory Expansion: available for adding Overlay Memory from 32K-bytes up to 512K-bytes total.
- Logic State Analyzer (LSA) Pod Assembly: provides 16 input lines and one trigger output line. The pod assembly gives you access to signals other than bus signals which are recorded simultaneously with the bus signals into the Trace Memory. These signals also become part of the Event Monitor System.
- Carrying Case: fits mainframe, one pod assembly, and LSA pod assembly.
- Symbolic Debug (described in appendix C).

Other options are available to configure your ES 1800 Satellite Emulator mainframe for use with other microprocessor families. See your sales representative for more information.

1.5 SPECIFICATIONS

Table 1-3 lists the specifications of the Satellite Emulator. Figure 1-4 shows the dimensions of the mainframe and emulator pod assembly.

Table 1-3. Specifications

INPUT POWER

Standard:
90 to 130 VAC
47 to 440 Hz
consumption less than 130W

Optional:
180 to 260 VAC
47 to 440 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, noncondensing

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.).

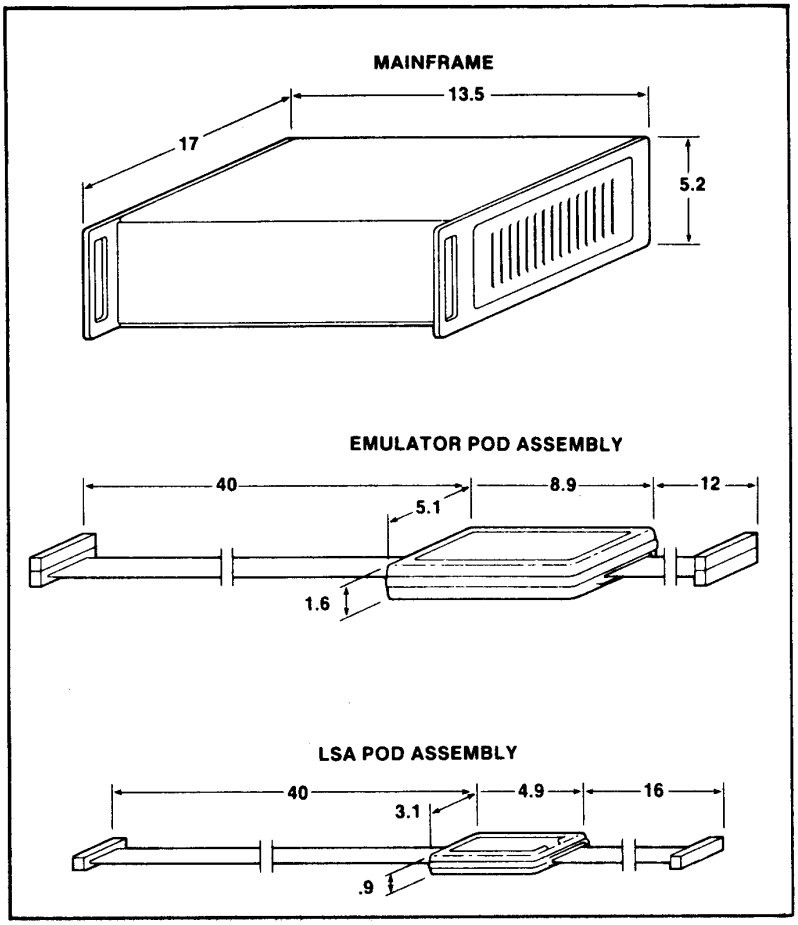


Figure 1-4. Dimensions

1.6 LIMITED WARRANTY

Applied Microsystems Corporation warrants that the equipment accompanying this document is free from defects in material and workmanship, and will perform to applicable published Applied Microsystems' specifications for one year from the date of shipment. THIS WARRANTY IS IN LIEU OF AND REPLACES ALL OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING THE WARRANTY OF MERCHANTABILITY AND THE WARRANTY OF FITNESS FOR PARTICULAR PURPOSE. In no event will Applied Microsystems be liable for special or consequential damages as a result of any breach of this warranty provision. The liability of Applied Microsystems shall be limited to replacing or repairing, at its option, any defective unit which is returned F.O.B. to Applied Microsystems' plant. Equipment or parts which have been subject to abuse, misuse, accident, alteration, neglect, unauthorized repair, or improper installation are not covered by this warranty. Applied Microsystems shall have the right to determine the existence and cause of any defect. When items are repaired or replaced, the warranty shall remain in effect for the balance of the warranty period or for 90 days following date of shipment by Applied Microsystems, whichever period is longer.

Extended warranty programs are available by contract.

1.7 SERVICE

If the unit is to be returned to Applied Microsystems for repairs, a repair authorization number will be issued by Applied Microsystems Customer Service for ES products. Call 1-800-426-3925 to obtain the necessary return shipment information.

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

SECTION 2
INSTALLATION AND SET-UP

2.1 UNPACKING AND INSPECTION

The Satellite Emulator was inspected and tested for any electrical and mechanical defects before it was shipped and adjusted for the line voltage you requested. The emulator was carefully packed to prevent any possible damage and should arrive in perfect operating condition. Carefully inspect it for any damage that may have occurred in transit. If any physical damage is noted, file a claim with the carrier and notify Applied Microsystems. Also check to make sure each unit of the Satellite Emulator system is present:

- the emulator mainframe
- the pod assembly for 680XX microprocessors
- the 680XX emulation control board
- the mainframe power cord
- Null Target Software Simulation Tool
- an extra probe tip
- the 68000/68008/68010 Operator Manual
- Optional equipment you may have ordered:
  - Overlay Memory
  - Logic State Analyzer pod assembly
  - a carrying case
  - Symbolic Debug

The following paragraphs describe how to properly set up an emulation system around the Satellite Emulator.

CAUTION:
DO NOT OPERATE THE EMULATOR UNTIL YOU HAVE COMPLETED THE PROCEDURES IN SECTIONS 2.2 THROUGH 2.5.

2.2 OPERATING VOLTAGE AND GROUNDING

The Satellite Emulator is normally set for operation on 90 to 140 VAC 50/60 HZ. It is also available for operation on 180 to 240 VAC 50/60 HZ, if so specified when ordered.

The emulator is supplied with a three-wire cord fastened to a three-terminal polarized plug for connection to a power source and a protective ground. The ground terminal of the plug is connected internally to the metal chassis parts of the emulator. Electric shock protection is provided when the plug is connected to a mating outlet with a protective ground contact that is properly grounded.

WARNING:
FAILURE TO PROPERLY GROUND THE SYSTEM WILL CREATE A SHOCK HAZARD

The emulator has three types of grounds. The first is the chassis ground that is connected to the metallic enclosure of the unit. The second type is the AC protective ground. This ground is derived from the third (green) wire of the AC power cord. It is

tied to the chassis ground at the power input filter of the emulator. The third ground is the signal ground. This is used as a common reference for all DC voltages and is the ground employed by the logic circuits. The signal ground is tied to the chassis ground (and thus to the AC ground) by means of a jumper at the power supply terminal strip.

NOTE:

Any target system connected to a Satellite Emulator should ideally have independent signal and chassis grounds that can be disconnected from each other when the target system is connected to the emulator. If the target system's signal ground is permanently tied to its chassis ground, a ground loop will exist. In some cases this will cause unwanted currents to flow through the emulator signal ground and may result in electrical noise on data, address, and control lines.

Total elimination of ground loops may not be practical if the system also contains peripherals that tie a signal ground to a chassis ground. When the signal and chassis grounds can't be separated, a low resistance strap between the emulator chassis and the target system chassis can reduce noise on the signal lines.

2.3 SYSTEM INTERFACING

The Satellite Emulator will be connected to the target system, a CRT terminal, and/or a host system. Two points must be considered: (1) the physical connection between the emulator and the CRT terminal or host system and (2) maintaining proper grounds throughout the system.

2.3.1 The Rear Panel

The rear panel of the Satellite Emulator is shown in Figure 2-1. The two serial ports are labeled **TERMINAL** and **COMPUTER**, to signify which is used for connection to a CRT terminal and which is for a host system, printer, or other source of data files. Be sure all peripherals are connected to the proper serial port.

Also on the rear panel is a BNC connector for connecting to an oscilloscope trigger, the main power switch, a line fuse, and the AC power connection.

The line fuse may be replaced if necessary. It is removed by turning the fuse holder counterclockwise with a small screwdriver. Replace with a 3-amp slow-blow fuse for 110-volt operation.

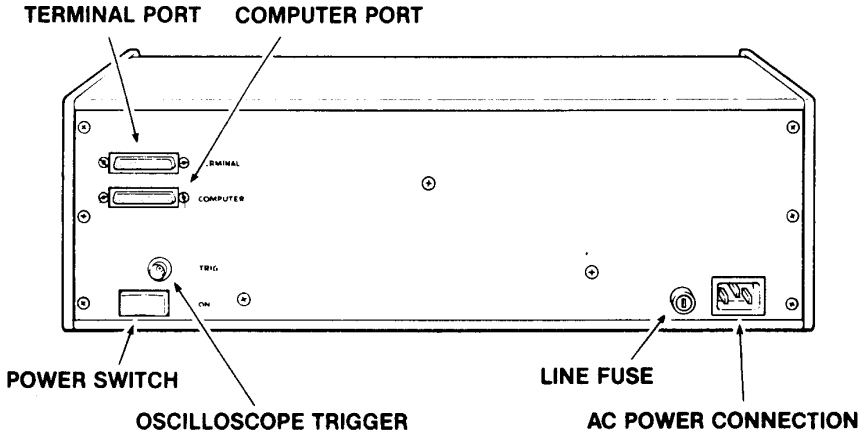


Figure 2-1
Rear Panel

2.3.2 The Side Panel The side panel contains the cooling fan for the emulator. See Figure 1-2 for the location.

CAUTION:
DO NOT BLOCK THE FAN OPENING WHEN THE POWER IS ON. THIS WILL CAUSE THE EMULATOR TO OVERHEAT.

2.3.3 Serial Port Connector Pin Assignment Figure 2-2 shows the pinout of the serial port connectors. Both ports use the same pin assignment. Table 2-1 lists the signals present on each pin. Pins without signals shown are not connected within the emulator. All pin assignments and voltage levels conform to Electronics Industries Association (EIA) RS232C standards.

Physically, there is no difference between the two ports. However, there are many software constraints making it important that peripherals are connected to the emulator at the correct port.

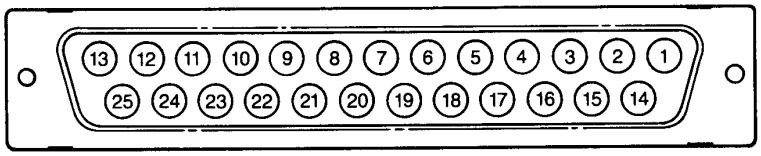


Figure 2-2
Serial Port
Connector
Pinout

The minimum connection to another unit consists of pins 1, 2, 3, and 7. Pins 4 and 5, Request to Send and Clear to Send, need not be connected unless other units connected to the emulator are using them.

You must be familiar with the pin configurations of your own equipment, as pins 2 and 3 vary and pins 1 and 7 are sometimes tied together.

CAUTION:
CHECK HOST AND CRT CONFIGURATIONS BEFORE CONTINUING

Table 2-1.
Serial Port

Connector Pin Signals

PIN	NAME	DESCRIPTION
1	Protective Ground	Connected in the emulator to the logic ground.
2	Serial Data Out*	This signal is driven to nominal ± 12 volt levels by an RS232C compatible driver.
3	Serial Data In*	Data will be accepted on this pin if the voltage levels are as specified by RS 232C specifications and follows the format outlined in Section of this manual.
		***NOTE:** **You should be familiar with the pin configuration of your own system. Some systems receive on pin 2 and some on pin 3. It may be necessary for you to rewire the cable connecting the units.**
4	Request to Send (Output)	This signal is driven to nominal ± 12 volt levels by an RS232C compatible driver; it signals other equipment that the emulator is ready to accept data on this port.
5	Clear to Send (Input)	This input to the emulator indicates that other equipment in the system is ready to accept data. This signal is terminated such that the emulator will operate with it disconnected.
6	Not Used	
7	Signal Ground	This pin is connected in the emulator to the system logic ground. Note, however, that this ground is connected to the emulator probe ground pin; when the emulator is connected to the target system, the target system logic ground and the emulator logic ground are connected together, and to the ground system of equipment plugged into the serial ports.
9 to 25	Not Used	

2.3.4 Setting Interface Parameters

A thumbwheel switch on the controller card selects the initial power-on interface parameters, set up in user-defined groups. After power-up, you can override the switch setting with software commands described in Section 3.5 of this manual. To select parameters, turn BOTH knobs to the left and remove the front panel of the emulator to expose the card cage, as shown in Figure 2-3. The controller card is the top card in the card cage.

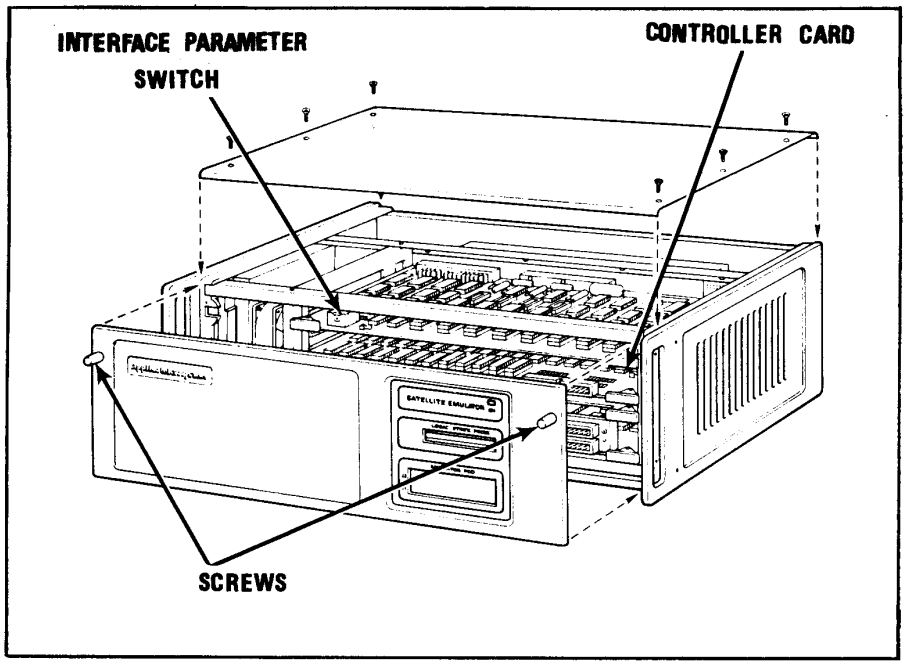


Figure 2-3
Top and Front Panel Removal

Refer to Table 2-2. The term "Factory Default" is used to denote an 8-bit word, one stop bit, and no parity. "User 0" and "User 1" refer to two operators. This allows two operators to each define their own power-up parameters, store them in the EEPROM (see Section 3), and recall them on power-up, depending on the switch position. "Terminal Control" and "Computer Control" determine which port will be active on power-up.

Table 2-2.
Interface Parameter Switch Settings

POSITION	FUNCTION	POSITION	FUNCTION
0	Factory Default 9600 baud	6	Factory Default 300 baud
1	User 0 Terminal Control	7	Factory Default 1200 baud
2	User 1 Terminal Control	8	Factory Default 2400 baud
3	User 0 Computer Control	9	Factory Default 4800 baud
4	User 1 Computer Control	A	Factory Default 7200 baud
5	Factory Default 110 baud	B	Factory Default 19,200 baud
		C,D,E,F	Reserved for factory use

Factory Default = 8-bit word, one stop bit, no parity

2.4 PHYSICAL CONNECTION

Connection to a host system will vary with the application. Contact Customer Service for ES Products if you require additional information for your host system.

2.4.1 Connection to CRT Terminal

You may need to consult your CRT terminal manual to correctly connect it to the Satellite Emulator. Standard parameters are:

- 9600 baud rate
- 8-bit word length
- one stop bit
- no parity
- full duplex
- no echo
- XON and XOFF are recognized.

Refer to the table above if you need to use a baud rate other than 9600.

Connect the CRT terminal to the TERMINAL port of the emulator. Make sure your connector pin assignment is compatible with the emulator.

On some CRT terminals, it may be necessary to turn the power off, then on, to ensure all switches are read by the CRT terminal hardware.

2.4.2 Connection to a Target System

To connect the Satellite Emulator to a target system, the procedure is as follows:

1. Verify that the target system power supply voltages are correct.

2. Install the proper emulation control board in the mainframe as shown in Figure 2-4. See the table below to determine the correct board for the microprocessor you are working with (your emulator will arrive from the factory with the correct board installed if you ordered only one family support; Z8000, 68000, etc.).

Table 2-3.
Model Numbers

DEVICE	EMULATION CONTROL BOARD	EMULATOR POD ASSEMBLY	SWITCH SETTING ON MCB (FIG. 2.4)
Motorola:			
68000	ES-68000B	ES-68000P	Left
68008	ES-68000B	ES-68008P	Centered
68010	ES-68010B	ES-68010P	Centered
Zilog:			
ES-Z8000B	ES-Z8000P		Right
Z8001	ES-Z8000P	ES-Z8001P	Right
Z8002	ES-Z8000P	ES-Z8002P	Right
Z8003	ES-Z8000P	ES-Z8003P	Right
Intel:			
8086	ES-8086B	ES-8086P	Centered
8088	ES-8086B	ES-8088P	Centered
80186	ES-8086B	ES-80186P	Centered
80188	ES-8086B	ES-80188P	Centered

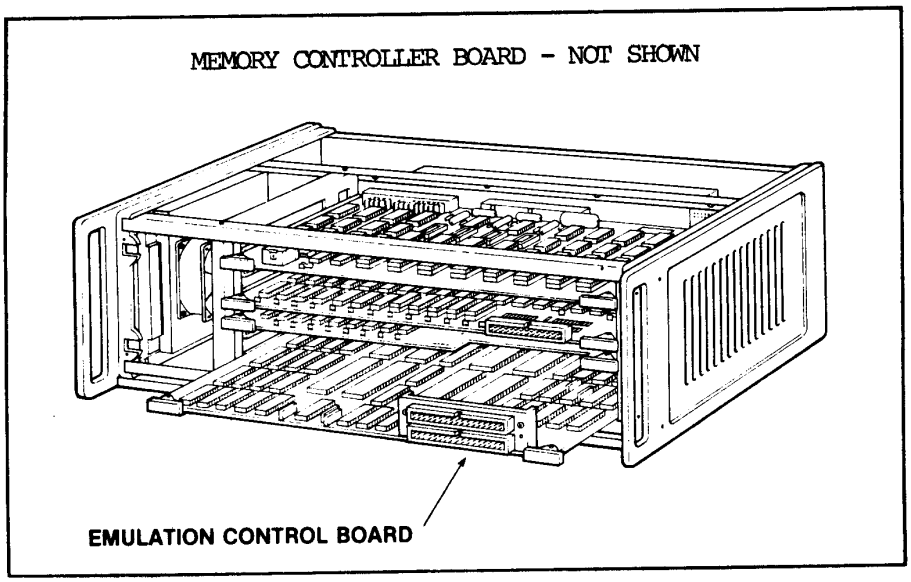


Figure 2-4
Installing the Emulator Control Board

3. Connect the emulator pod assembly to the mainframe as shown in Figure 2-5.

4. With target system power off, remove the target system microprocessor from its socket and plug in the DIP header, as shown in Figure 2-6.

CAUTION:

NOTE CORRECT PIN 1 ORIENTATION

5. The next section gives power-up procedures.

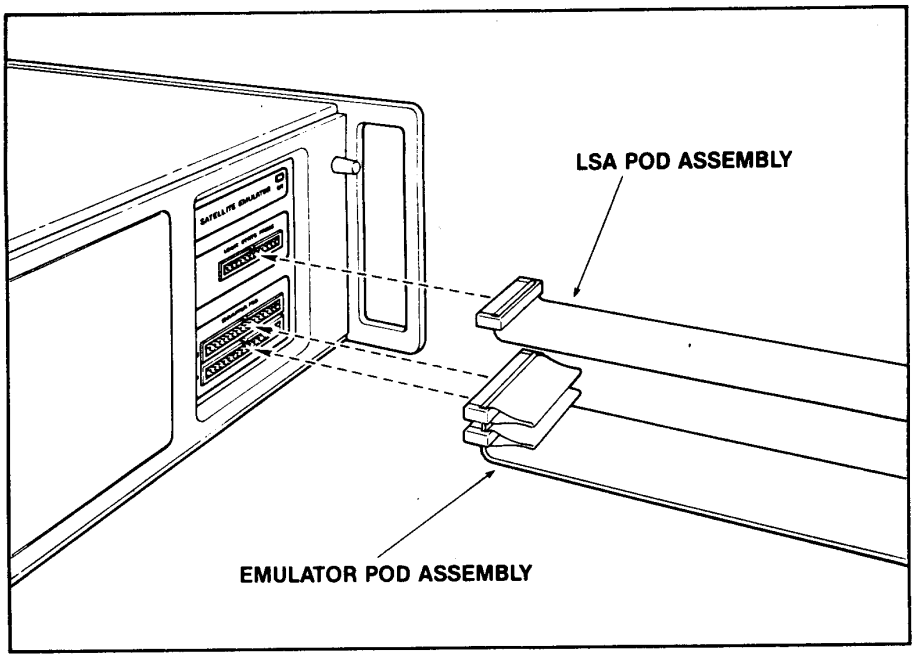


Figure 2-5.
Connecting the
Pod Assemblies
To the Mainframe

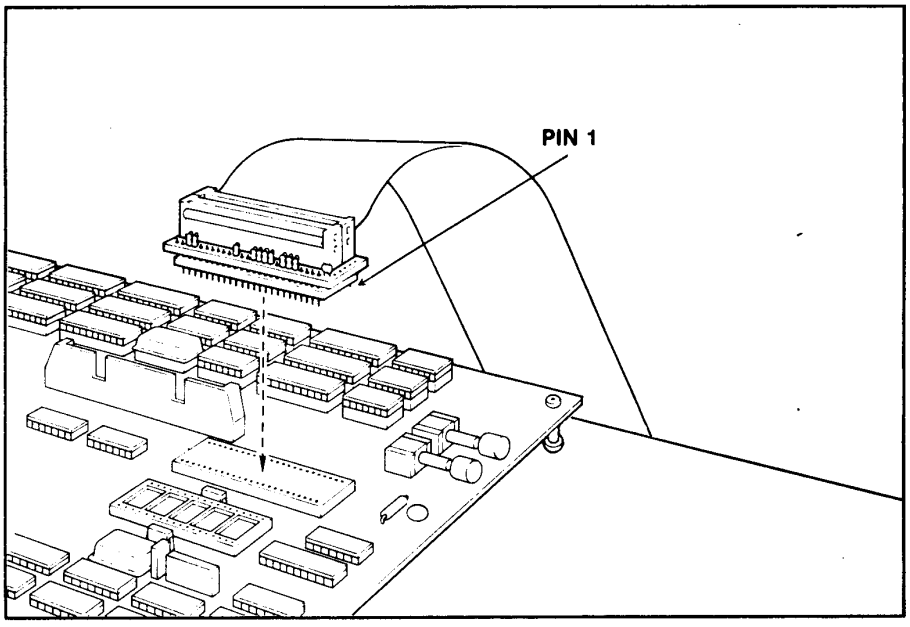


Figure 2-6.
Installing the
DIP Header Plug

2.5 SYSTEM POWER-UP AND CHECKOUT

With the emulator properly connected to a CRT terminal and your target system, first turn on the CRT terminal, then the target system, and finally the emulator. When the power is first applied to the Satellite Emulator and its clock begins operating, a Power-on-reset operation occurs during which the following functions are performed:

1. The microprocessors in the mainframe and pod are both reset.

2. The Trace Memory, Event Monitor System, and registers are cleared.

3. The emulator program starting address is cleared to zero (the default starting address).

4. If the interface parameter switch is in any of positions 1 through 4, the parameters and register values stored in the EEPROM are loaded. See Section 3.

You should see this display at the top of the screen:

COPYRIGHT 1983
APPLIED MICROSYSTEMS CORPORATION
SATELLITE EMULATOR V X.X
USER = n, SW = n
XXK AVAILABLE OVERLAY

- When the three-line header appears, the emulator is in a self-test mode, checking items 1 through 4 above. When the test is complete, the > prompt tells you that the emulator is ready to receive your instructions. (Always make sure that the > prompt shows before you type in a command or you will lose one character and the command will fail. You then must re-enter the command).

NOTE:

If the > does not appear, turn off all equipment, check the connections and then repeat the power-on sequence: terminal, target system and emulator. If the > prompt still does not appear, contact your Applied Microsystems representative.

- The user number (USER = n) and software number (SW = n) display the current settings on the interface parameter switch.

- You should verify that your system clock is operating correctly.

2.6 PRE-EMULATION CHECK LIST AND THE HELP MENU

As mentioned previously, before beginning emulation some of the features associated with it must be set up. First, review the Help feature.

1. At any time after the emulator is receiving power, you can call up the Help Menu by entering ?. This feature of the Satellite Emulator is two built-in display pages that summarize the operators used and the input form of each. Figure 2-7 shows the two displays. To access the first display (Figure 2-7a), key in:

   ?

   To move to the second display, enter:

   <return>

To return to the first help screen, type ? again.

To move out of the Help Menu after the first page (without viewing the second page), enter any character other than <return>. The emulator will return a > prompt and you can enter your next command.

The Help Menu can be accessed at any time as long as the emulator responds to input characters and it has not just transmitted a "?".

Figure 2-7.
The Help Menu

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

TRACE HISTORY:                          DTB - DISASSEMBLE PAGE BACKWARD
  DT - DISASSEMBLE MOST RECENT LINE     DTF - DISASSEMBLE PAGE FORWARD
  DT X TO Y - DISASSEMBLE BLOCK         DRT (X) - DISPLAY PAGE RAW TRACE (FROM X)

MEMORY - REGISTER COMMANDS:             DR - DISPLAY ALL CPU REGISTERS
  DB(.BWL) 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 = SC0,SC1,..,SC7,SP,SD,UP,UD      DEFINES STATUS LINES FOR MEMORY ACCESS
  .B, .W, .L DETERMINES DATA LENGTH     M X - VIEW/CHANGE MEMORY AT X
  A0.B, .W, .L - BYTE, WORD, LONG       X - EXIT MEMORY MODE

MEMORY MAPPING:                         OVE = SC0+..+SC7/SP+SD+UP+UD; OVS = 0-7
  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
  LD/SAV (X) - LOAD/SAVE 0=SETUP,1=REGS,2=EVENTS,3=MAP,4=SWITCHES (DEFAULT=ALL)
```

a. *FIRST PAGE OF HELP MENU*
Access by entering (?)

```
                        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
                     + SP/SD/UP/UD/SC0-SC7 + IP0 .. IP7
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  2 WHEN AC1 AND NOT DC2 THEN CNT
CL.2 = 24;AC2.2 = $F000 LEN $400         WHEN CL.2 OR AC2.2 THEN BRK
 >
```

b. *SECOND PAGE OF HELP MENU*
Access by entering a <cr>
Skip by entering any other character

2. The first time you use your emulator, it must be powered up with the interface parameter switch in position 0 or positions 5 through B. These are pre-set factory interface parameters. Your CRT terminal must be set to match these the first time the emulator is used. They can be user-defined

after the unit is powered up, using the SET command described later in this section.

2.6.1 Parameter Set-Up and EEPROM Storage Overview

The Satellite Emulator contains an interface parameter switch that allows you to power up the emulator with one of eight sets of factory-defined parameters, or one of four sets of user-defined parameters. These user-defined and other display and interfacing defaults are defined with SET commands. All the data defined with the SET commands can be stored in an EEPROM (Electrically Erasable Programmable Read Only Memory) located on the controller board. The EEPROM can also store register values, parameters for the Event Monitor System, terminal characteristics and the memory map for the Overlay Memory.

Set-Up SET

SET commands are used to configure Satellite Emulator interface and display parameters. A menu display, shown in Figure 2-8, shows the general syntax for the commands and what parameters are in effect. To access this display, enter:

>SET<return>

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

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

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

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

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

Figure 2-8. Display Format

The following example shows the key sequence for entering SET commands. Table 3-5 at the end of Section 3 shows which parameters can be defined, and which SET commands require the reset character. The reset character is Ctrl Z, unless changed in your system by you or a previous user.

Some of the parameters set via SET will go into effect immediately. Others will require you to enter a reset character first. You will be prompted for these by the display **"YOU MUST RESET ME TO INSTALL THIS VALUE IN H/W."**

Note that the SET menu display shows what is in effect currently if you have not yet changed any parameters. If you have changed some

but not yet entered the reset character, it will show what will be in effect after the reset character is input.

The generalized key sequence to alter the interface parameters or the CRT display format, is:

>SET<select number>, <value>[,value][,value]<return>

The select number selects which attribute will be altered. The values entered correspond to the selections displayed in Table 3-5. Remember to use decimal-based numbers when entering the select number.

NOTE:

When scrolling, XOFF (Ctrl S) is used to stop the screen and XON (Ctrl Q) turns the scrolling on again. You may need to change the defaults for use in the transparent mode, for instance. Like all codes specified and displayed by the SET command, those new values can be stored in EEPROM. XON and XOFF are set as follows:

>SET 3, $10, $12

In this example, XON has been changed to 10_{16} and XOFF has been changed to 12_{16}.

Load and Save

The EEPROM is partitioned into space for two users (0 and 1). Each user's space is partitioned into five groups:

LD

- 0 = system set-up (defined via SET)

SAV

- 1 = all the registers in the system and Event Monitor System event comparators
- 2 = Event Monitor System WHEN/THEN statements
- 3 = RAM Overlay map
- 4 = Software Switch Settings (see Section 3.5)

A user's number is determined by the SET operator described in the previous section. Parameters selected with the SET commands are stored in the EEPROM with the SAV (Save) command. Once parameters have been stored via SAV, they can be called up with the Load command. When you first receive the machine or when converting it from another microprocessor family, initialize the EEPROM to the proper data, you should execute a:

>SAV (no argument)<return>

The entire contents of the EEPROM of the appropriate user will be loaded into the Satellite Emulator automatically on power-up if the interface parameter switch is in positions 1, 2, 3, or 4. When the switch is in any other position, you must key in a Load command to access the data in the EEPROM.

You can selectively Load or Save the groups of data with the EEPROM.

- To Load or Save all the groups, key in the Load and Save command.
- To Load or Save only one group of data, such as just the registers, you will key in the group number in addition to the command.
- To Load or Save a combination of data groups, such as the parameters and the registers, you will have to enter two or three commands.

Run With Vectors RNV

The program counter and stack pointer are loaded to their starting values, and emulation is started when you enter RNV <return> (See section 4.2 for detailed instructions).

You will see an R> (the RUN prompt). Most commands can be executed from the R>. However, if your command fails and you see a ?, follow these steps:

- Enter ? to get the error message telling you why the command failed.
- If the command failed because it cannot be executed from the R>, enter STP. When the > appears, re-enter the command.
- If the command failed for any other reason, follow the appropriate measures to correct.

Sample sequence for first-time emulation:

```
>RNV          Loads target system vectors, begins program execution
>STP;DTB      Stops program execution and disassembles one page
>STP;DT       Steps (executes) one instruction and disassembles it
>AC1 = <address>; WHEN AC1 THEN BRK
                    Sets breakpoint address
>RBK; WAIT; DTB
              Runs program, waits until breakpoint, then dis-
              assembles one page
```

Here is another sequence you can try. You will be unfamiliar with the syntax at this point, but type in the commands to get the feel of the emulator. Observe what happens on the screen and read the comment in the right-hand column before going on to type in the next command. (System syntax is explained in Section 3, and emulation is explained beginning in Section 4.)

OVERVIEW:

Stop emulation. Display all the CPU registers, then display just one register, then change a register and display all the registers again to view the change. Display a block of memory in byte form, word form, or long word form, whichever is most convenient. Go into memory mode to display memory data and then change it.

STP	Stop emulation
DR	Display all the CPU registers.
A1.W=8844	Change the lower word of register A6 to the value 8844 hex (assuming the default base is 16).
A6=$1F34CD22	Change the 32-bit length register A6 to the value 1F34CD22 hex
D3.B=#16	Change the lower byte of data register D3 to 16 decimal or 10 hex. Note that the prefix $ means hexadecimal, the prefix # means decimal, / denotes octal, and % denotes binary.
DR	Display all the CPU registers again to verify they were changed as expected.
A3	Typing in just the name of a register will return the value of that register.
DB.B 100 TO 140	Display a block of memory from location 100 to 140 (hex). The .B means in byte form.
DB.W 100 LENGTH 40	Display a block of memory from location 100 with the length of 40 in word form.
DB.L $100 LEN $40	Display a block of memory. Notice the prefix $ is optional since DFB = 16 and that the short form of LENGTH is used.
MAP 4000 LEN 1000 :RW	Map a block of memory to reside in RAM Overlay and make it read/ write.
MMS=SP	There are four different memory spaces in the 680XX: Supervisor Program, Supervisor Data, User Program, and User Data. The ES is now pointing to the supervisor program memory command.
M.W 4000	This is the command to inspect and change memory.
<return> <return> <return>	Notice that each time you hit the <return> that the memory location is incremented by one word and the next word is displayed.

. . .	A period (at the beginning of the line only) without a <return> will increment the address of the word being displayed as well.
, , ,	The comma decrements the address of the word being displayed.
1111	To change the value at that location, simply type in the new value.
2222, 3333, 4444	To change values in successive locations, simply separate the values by commas.

This is how to patch data in your program.

(For patching instructions, you would use the single line assembler, ASM.)

, , , , ,	Back up to see that you did indeed change the values stored at those locations.
X	Exit the memory mode.

NOTE: Some emulator features that enhance emulation may be set up prior to emulation. However, if you are working with the emulator for the first time, we recommend following the sequence of commands given in Section 4. This sequence starts with the most basic commands.

The advanced features that enhance emulation are:

- Event Monitor System - the Event Monitor System allows you to select events within emulation that will cause specified actions to occur. These events and resultant actions may be defined prior to emulation and are described in Section 5.

- Memory Mode - allows you to examine and change contents of the target system memory.

- Overlay Memory - you may wish to map your target system program memory and fill it with data. This is described in Section 4.5.

- Trace Memory - special conditions can also be set for the Trace Memory. These are described in Section 4.6.

SECTION 3
SYSTEM SYNTAX AND PARAMETERS

3.1 INTRODUCTION

This section explains how to use ESL, the Satellite Emulator control language. The information here will be used in conjunction with that given in Sections 4-7.

NOTE:

If you are reading this manual for the first time, you should familiarize yourself with the contents of this chapter. Then you can then refer to specific sections when you need to use them. New users do not need to read all of the information word for word.

The Satellite Emulator operates in response to command statements made up of operators and arguments. Operator refers to the command mnemonic or symbol used (RUN, FIL, etc.). Argument refers to any additional value you must enter as part of the command sequence, such as an address range or base value. Essentially, the command operators form a control language, much like higher-level computer languages. And, like a computer languag, the operators and arguments may be combined in various ways to form many complex command "sentences." The Satellite Emulator accepts operators and arguments when they are logically combined into a statement. Statements can be up to 79 characters long.

The control software recognizes over one hundred mnemonics described in Sections 3, 4, 5, and 6. You have two options in entering the mnemonics. Since the software recognizes the first three letters and last two digits, you can enter just these, as in GRO for group, or you can enter GROUP. Any letters included after the first three are disregarded: GROABCD would also be recognzed as GRO. Note that the limitation on the last two digits only refers to those included in operator mnemonics. Other numerical values are not limited.

In the following discussion many examples have been included to illustrate the test. Some conventions have been adopted for ease of explanation.

- When an angle bracket <> encloses an expression, it is required entry; for example, <address range> or <value>.

- When square brackets [] are used to enclose an expression, it is an optional entry; for example, [base value].

3.2 STANDARD CHARACTERS

Standard characters appear throughout all the operations of the Satellite Emulator.

3.2.1 The Prompt Character >

When the Emulator is ready to accept a command statement, the prompt character (>) appears on the left margin of the CRT terminal screen. In the examples, it should be understood that you do not type in the prompt character; it was already supplied by the command interpreter, indicating readiness for another input line.

3.2.2 The Run Prompt R>

The Run prompt appears on the CRT terminal to notify you that the emulator is in Run mode (emulating).

3.2.3 Spacing

Space characters (the space bar) are used to improve readability. Normally, you may enter them at your discretion except as required to separate two named items (such as "NOT AC2"). So the statement:

>GD4 = GD4 + #8 * GD2 <return>
can also be written as:

>GD4=GD4+#8*GD2<return>.

Lower-case characters are converted to upper-case except in the Transparent Mode or when using Symbolic Debug.

3.2.4 Utility Operators

The utility operators are used to separate, execute, edit, and repeat other commands. These operators are:

<RETURN> **RETURN.** The <return> is used to terminate statements and execute commands. It must be entered after every statement. It is also used to scroll through addresses while you are in the Memory Mode. On some CRT terminals, the key may be labeled **ENTER.**

/ **REPEAT PREVIOUS COMMAND LINE.** When this operator (/) is the first character of a line, it repeats the previous command line. When it appears anywhere else on a line, it signifies arithmetic division.

; **STATEMENT SEPARATOR.** The semicolon (;) is used to separate command statements that are strung together on one line.

, **ARGUMENT SEPARATOR.** Just as the semicolon separates command statements, the comma (,) is used to separate arguments when more than one argument is required to form a command statement. The comma is also used to decrement addresses when you are in the Memory Mode, where it will be the only operator on a line.

CNTL X **DELETE LINE.** The CNTL (control key) X command will delete that line.

CNTL R **REPRINT CURRENT LINE.** CNTL R will reprint the line you just entered. This will be useful to you when you are making a hard copy. Don't confuse this with the / operator - the command is not repeated, only reprinted.

.B
.W
.L

DOT OPERATORS. The .B, .W, and .L operators used with the 680XX are also recognized by the Satellite Emulator. The .B and .W operators are used to refer to the lower byte or word of a long word value. The resulting value is formed by sign extending the byte or word value to 32 bits.

- When @ is used, .L is the default extension regardless of system default, unless you specify .B or .W.
- When commands such as DB, FIL, or FIN are used, size is determined by SZ < .B, .W or .L>

You may temporarily override system default at any time by keying in these extensions.

3.3 NUMBERS AND BASE VALUES

There are three basic types of values used in the emulator: Normal, Don't Cares, and ranges.

- Normal values are simple integer numbers.
- Don't Care values consist of two normal values separated by the Don't Care operator DC. Don't Care values are best envisioned in binary form. The value to the right of DC should have some bits

set. These bits are used as a mask so that every bit set in the right side value causes the corresponding bit position on the left value to be ignored. Don't Care values are useful when you are working with the Event Monitor system to monitor bit logic and are described in Section 5.

o Range values consist of two normal values separated by one of the range operators TO or LEN. Range values are useful for referring to blocks of memory and are also described in Section 5. Also, XRA and IRA can be prefixed to the arguments to define external or internal ranges, respectively. The default is IRA.

The base value operators are used to set the numeric base you want to work with or to temporarily change the base in effect. On power-up the default base is hexadecimal (unless another default base has been loaded by the EEPROM on power-up).

3.3.1 HEXADECIMAL ($) DECIMAL (#) BINARY (%) OCTAL (\)

These operators tell the emulator what base a value is in. The format is $n, #n, %n, or \n, where n is any numeric value. The operator preceding n tells the Satellite Emulator that n is in that base. They are used any time you want to enter a value in other than the default base. Values not preceded by one of these operators are presumed by the emulator to be in the default base.

The following numbers show the format for the different bases:

- $270F - hexadecimal
- #9999 - decimal
- \23417 - octal
- %10011100001111 - binary

3.3.2 Default Base DFB

The DFB operator is used to display the system default base or change the default base in effect (factory default is hexadecimal). The Satellite Emulator will attempt to work with any base you set, though decimal, hexadecimal, octal, or binary are the most meaningful. Numbers without a base prefix are assumed to be in the default base. If any number larger than 16 (hexadecimal) or smaller than 2 (binary) is assigned, the Satellite Emulator will assume the base to be hexadecimal. The following example shows the key sequences for assigning default bases.

- To display the default base in effect:
  >DFB<return>
- To set the default base to binary:
  >DFB = #2<return>
- To set the default base to decimal:
  >DFB = #10<return>
- To set the default base to octal:
  >DFB = #8<return>
- To set the default base to hexadecimal:
  >DFB = #16 <return>

- The same format as shown above is used to set the emulator to any other base desired between 2 and 16.

.3.3 Display Base BAS

This operator displays the base currently in effect for a specific register, as shown in the following example. Displayed bases are always shown in decimal:

- #16 = hexadecimal
- #10 = decimal
- #8 = octal
- #2 = binary

If it is necessary to have a specific register value displayed in other than the default base, you can assign it a "private" display base of any number between 2 and 16. Be careful when setting private display bases to unusual bases such as 4,7, or 11. The Satellite Emulator will operate correctly but the results may be confusing. The example also shows how to set private display bases.

If the base value is set to other than hexadecimal, decimal, octal or binary, the emulator will display a ? when you ask it to display the base in effect--there are symbols only for the four most common bases.

- To display the current default base:
  >DFB<return>

- To display the base of a specific register:
  >BAS GD3<return>

  GD3 is the name for a specific register that you need to know the base of. The emulator may respond with #16 to show that the register base is hexadecimal. Note, however, that though the register is hexadecimal, the base is displayed in decimal: #16 = hexadecimal, #8 = octal, #10 = decimal, #2 = binary, etc.

- If a register has no private display base assigned, the result of this command will be
  DEFAULT:#n
  where n is the current default base.

- To set a private display base:
  >BAS GD3=2<return>

  This sets the display base of GD3 to binary but does not affect any other values or the default base (it only affects GD3). The next time you display the base of GD3, the CRT terminal will respond with:

  #2

  The value of GD3 will always be displayed in binary until you key in a different display base or the Satellite Emulator is reset. The private display base of any register may be assigned the value 0 to cause that value to be displayed in the default base.

3.4 ARITHMETIC OPERATORS

Arithmetic operators can be divided into three groups.

- Assignment operators are used to assign values.
- Single-argument operators assign a property to a single argument.
- The two-argument operators include the more common arithmetic symbols and operators for more specific arithmetic operations. Each of these groups have some specific characteristics. Table 3-1 lists the arithmetic commands and tells which of the three groups each falls in.

Table 3-1.
Arithmetic Operations

GROUP	OPERATOR	NAME
Assignment Operators:		
	=	Equal
	()	Parentheses
	@	Indirection
Two Argument Operators:		
	*	Multiplication
	+	Addition
	/	Division
	-	Subtraction
	MOD	Modulo
	&	Bitwise AND
	^	Bitwise OR
	<<	Shift Left
	>>	Shift Right
Single Argument Operators:		
	!	Inverse
	-	Negation*
	ABS	Absolute Value

*Note that the minus sign (-) signifies subtraction when it is part of a two-argument statement and negation when it is part of a single-argument statement.

3.4.1 Assignment Operators

Assignment operators assign a value or property to an argument. They also extend expressions to include values obtained from combinations of other expressions, or values stored in the target system memory address space.

Generally, the form taken by the result of an operation will be the form of the left-hand argument: a Don't Care value times a normal value will be a Don't Care value. There are two exceptions to this:

1. When a normal value appears on the left and a Don't Care value on the right, the result will include Don't Care bits;

2. When a normal value appears on the left and an internal or external range appears on the right, the result will be a range.

=

EQUAL. The equal sign passes the quantity defined on its right to the entity on its left. All operations to its right will be performed before the equality is considered. The entity on the left should be a single entity.

- The equal sign is used as follows:

  >GD3 = $47FF<return>

  The emulator does not display anything in response to this entry, but the value you entered at the right ($47FF) is now assigned to GD3.

- It is also used as follows:

  >GD3 = $121 + $4<return>

  This would first add $4 to $121. GD3 is then assigned the value $125.

()

PARENTHESES. The emulator recognizes parentheses, just as they are treated in algebraic equations: all operations within the parentheses are performed first and a single value derived.

NOTE

There is no set number of levels of parentheses that the Satellite Emulator can work with. The only limitation is that statements can be no more than 79 characters long. Whatever level of complexity you can handle within this limitation will be handled easily by the emulator.

@

INDIRECTION. The "at" sign is used to express indirection. Indirection allows expressions to include values obtained from, or stored to, the target system memory address space. The @ operator causes the command interpreter to consider the value of the expression following to be an address of a target system word; the word is accessed and that word--from the target system address space--becomes the value of the expression.

It is possible to use more than one @ operator in an expression. If two are used, the Satellite Emulator will access the expression following the operators and look at the address pointed to; the value at that address is then also considered to be an address, and that address is accessed and displayed. This gives a means to display a quantity that is pointed to by some other quantity located in the target system memory. Refer to Example 3-1 for illustration of this operator.

Just as with parentheses, the Satellite Emulator is capable of dealing with many levels of indirection. However, again due to the limitation that statements not exceed 79 characters, you will probably not deal with more than 70 levels of indirection at one time.

The dot operators (.B, .W, or .L) can also be used with @.

Example 3-1.
Parentheses and
Indirection

- The following two examples help explain using Parenthesis and Indirection together:

  >@GD4 + 6<return>
  >@(GD4 + 6)<return>

  Both contain the indirection operator and the same argument, GD4. In the first example, the indirection operator would be applied to GD4: the command interpreter accesses the target system location pointed to by GD4, adds six to the value stored there, and then will display the final result. Instead, if you wanted to see the location stored in six locations above the address pointed to by GD4, you would use the second example, using the parentheses to signify that GD4 + 6 is one entity.

- It is also possible to use indirection in an assignment function:

  >@(GD4 + 6) = #10 <return>

  This example assigns the number ten to the target system memory location which is found six bytes above the location pointed to by GD4.

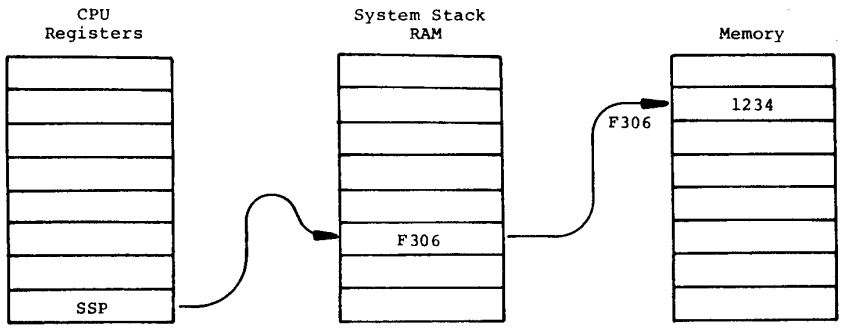
- The following example is also legal:

  >@(GD4 + 6) = @(GD4 + 8)

  Here, a quantity offset eight bytes from the location pointed to by GD4 is copied to a location offset six bytes from the location pointed to by GD4. This is a target-to-target move.

- You can use more than one @ operator in a statement. In this example the dual indirection is used to access a table of data that is pointed to by the system stack pointer.

  >@@ SSP
  >12345678



- The dot operators can also be applied to @. The command:

  >@.WA3 = @.BDO.W

  would cause the following sequence to occur:

1. The lower word of D0 is collected and sign extended to 32 bits.

2. That value is used as an address to memory where a byte value is read.

3. The value just read from memory is sign extended from 8 to 32 bits.

4. The contents of A3 are used as an address where the low-order word of the 32-bit value above is stored.

3.4.2 Two-Argument Operators

The two-argument operators involve an arithmetic or logical operation between two values. The following table lists the two-argument operators and the combinations. It is set up as a matrix, showing what operations are valid. Refer to this table in the following discussion of the individual operators.

NOTE

Normal refers to simple arithmetic values. DC means Don't Care bits are included, IRA is an internal address range, and XRA is an external address range. They are explained in detail in Section 4.

Table 3-2. Two-Argument Operation Validities

LEFT-HAND ARGUMENT	RIGHT-HAND ARGUMENT	OPERATION	RESULT
Normal	Normal	* / MOD	Valid
		& ^	Valid
		<< >>	Valid
		+ -	Valid
Normal	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.
		<< >>	Invalid
		+ -	Don't Care bits are passed to the left hand argument.
Normal	IRA XRA	* / MOD	Invalid
		& ^	Invalid
		<< >>	Invalid
		+ -	The endpoints of the range will be altered by the value of the normal expression.
DC	DC	* / MOD	Invalid
		& ^	Invalid
		<< >>	Invalid
		+ -	Don't Care bits are ANDed
DC	DC	* / MOD	Don't Care bits are kept
		& ^	Valid
		<< >>	Don't Care bit positions are shifted
		+ -	Don't care bits are kept
IRA, XRA	Normal	* / MOD	Invalid
		& ^	Invalid
		<< >>	Invalid
		+ -	The endpoints of the range will be altered by the value of the normal expression.

* **MULTIPLICATION.** An asterisk is used to denote multiplication. Multiplication is algebraic--the value to the left is multiplied by the value to the right of the * operator. And, as in an algebraic equation, multiplication has precedence over addition or subtraction in the same equation or statement unless the operator separator (;) is used. Multiplication can't be performed on address ranges.

+ **ADDITION.** Addition is denoted with the addition sign. Like multiplication, it operates just as in an algebraic equation. Addition can be performed on address ranges and Don't Cares.

/ **DIVISION.** Division follows the same principles as multiplication. It has precedence over addition and subtraction when all are contained in one equation or statement. Be careful not to confuse the slash operator used for division with the slash used for Repeat Previous Command Line. Both use the same key, but, when the slash is used to repeat command statements it will be the first character on a line. When used for division it is between two arguments - it cannot be the first character on a line. Division can't be performed on address ranges or Don't Cares.

Example 3-2. Multiplication and Addition

Here's an easy example:

>GD4 = GD4 + #8 *GD2<return>

=	the equal operator
GD4	(again)
+	the addition operator
#8	a number, written with the decimal prefix
*	the multiplication operator
GD2	a variable name representing another register
<return>	the return symbol

The effect of this statement is to read the current value of register GD4, add to this value the product of 8 and the value contained in GD2, and assign this sum to GD4, thus changing the value it contains.

- **SUBTRACTION.** Subtraction is much the same as addition. It is denoted by the minus sign (-). The minus sign is also used to denote negation or two's complement.

MOD **MODULO.** The result of this operation is the remainder after the value on the left has been divided by the value on the right. See the following example.

- **>29 MOD 4**
  result = 1

- **>38 MOD 6**
  result = 2

>> **SHIFT LEFT AND SHIFT RIGHT.** These two operations are a movement of
<< the bits of a number. For example, a right shift of n places has the effect of dividing by 2^n and a left shift of n places has the effect of multiplying by 2^n. See the following example.

- A binary shift left:
  >00100000<<1
  result = 01000000

- A binary shift right:
  >0001000000000000>>1
  result = 100000000000

& **BITWISE AND.** Bitwise And operator (&) functions as a logical AND. The & operator infers the ANDing of the bits that form the two arguments. Refer to example 3-3 and Table 3-2.

^ **BITWISE OR.** The Bitwise OR operator (^) functions as a logical inclusive OR. The operator infers the ORing of the bits that form the arguments.

Table 3-3. Bitwise Operator Validities

AND &			OR ^		
INPUT		OUTPUT	INPUT		OUTPUT
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	1	1	1	1

Example 3-3
Bitwise And
Bitwise Or

- Bitwise And:
  >%00101101 & %10011100
  result = %00001100

- Bitwise Or:
  >%00101101 ^ %10011100
  result = %10111101

3.4.3 Single-Argument Operators

Single-argument operators assign a property to the number directly following the operator. The following table summarizes the operators and valid combinations.

Note that a single-argument operator can even be used before a parenthetical operation, and the value within the parentheses will be treated as a single value.

! — **INVERSE/ONE'S COMPLEMENT.** The exclamation mark is used to signify that the following number or value is to be inverted. The inverse is the one's complement; the inverse of %0010 would be %1101. Address ranges can also be inverted: an internal into an external and vice versa.

- — **NEGATION AND TWO'S COMPLEMENT.** The minus sign is also used for negating a number when used as single-argument operator. This operator forms the two's complement of its argument.

ABS — **ABSOLUTE VALUE.** The ABS operator converts the following value to its absolute, positive value; a negation value would become positive, a positive value would remain unchanged.

Table 3-4.
Single-Argument
Operators

OPERATOR	ARGUMENT	RESULT
!	Normal	Valid
	DC	Don't care bits are not affected
	IRA	Complement (IRA becomes XRA)
	XRA	Complement (XRA becomes IRA)
ABS	Normal	Valid
	DC	Don't care bits are not affected
	IRA	Invalid
	XRA	Invalid
-	Normal	Valid
	DC	Don't care bits are not affected
	IRA	Invalid
	XRA	Invalid

3.5 PARAMETER SET-UP AND EEPROM STORAGE

As mentioned in the overview in Section 2, you may set system parameters with the SET command. These are listed in Table 3-5, which follows.

You can also set an additional 11 parameters using software switches. Ten of these relate to emulation and one determines whether or not you can produce hard copy during an emulation session. These are listed after Table 3-5 under ON and OFF.

Table 3-5
SET Select Numbers

SEQUENCE	DESCRIPTION/RESULT	RESET CHARACTER REQUIRED
SET #1,#0<return>	Select User 0	No
SET #1,#1<return>	Select User 1	No
SET #2,<return>	Specify Reset character (default = $1A)	No
SET #3,$n,$m<return>	Set XON, XOFF (defaults = $11, $13)	No
SET #10,#0<return>	Set CRT terminal baud rate to 50	Yes
SET #10,#1<return>	75 baud (CRT terminal)	Yes
SET #10,#2<return>	110 baud (CRT terminal)	Yes
SET #10,#3<return>	134.5 baud (CRT terminal)	Yes
SET #10,#4<return>	150 baud (CRT terminal)	Yes
SET #10,#5<return>	300 baud (CRT terminal)	Yes
SET #10,#6<return>	600 baud (CRT terminal)	Yes
SET #10,#7<return>	1,200 baud (CRT terminal)	Yes
SET #10,#8<return>	1,800 baud (CRT terminal)	Yes
SET #10,#9<return>	2,000 baud (CRT terminal)	Yes
SET #10,#10<return>	2,400 baud (CRT terminal)	Yes
SET #10,#11<return>	3,600 baud (CRT terminal)	Yes
SET #10,#12<return>	4,800 baud (CRT terminal)	Yes
SET #10,#13<return>	7,200 baud (CRT terminal)	Yes
SET #10,#14<return>	9,600 baud (CRT terminal)	Yes
ET #10,#15<return>	19,200 baud (CRT terminal)	Yes
SET #11,#1<return>	CRT terminal data frame has 1 stop bit	Yes
SET #11,#2<return>	2 stop bits (CRT terminal)	Yes
SET #12,#0<return>	CRT terminal parity (send and receive) none	Yes
SET #12,#1<return>	Parity even (CRT terminal)	Yes
SET #12,#2<return>	Parity odd (CRT terminal)	Yes
SET #13,#n<return>	Set CRT terminal lines per page; n=5 to 255	No
SET #14,$n<return>	Specify an 8-bit Reset character. The reception of this character from any port in any mode resets the emulator	No
SET #15,$n,$m<return>	CRT terminal transparent mode escape sequence; n and m are arbitrary character codes; 7-bit ASCII values only	No
SET #20,#n<return>	Select computer baud rate; n0 to 15 as in SET #10,n above	No
SET #21,#1<return>	Computer data frame has 1 stop bit	Yes
SET #21,#2<return>	2 stop bits (computer)	Yes
SET #22,#n<return>	Select computer parity as in SET #12,n above	Yes
SET #23,$n,$m<return>	Set computer transparent mode escape characters as in SET #15,n,m	No

Table 3-5 (cont.)

	DESCRIPTION/RESULT	RESET CHARACTERS REQUIRED
SET #24,$n,$m,$o <return>	Command terminator for Download; n, m, and o are arbitrary 7-bit ASCII character codes	No
SET #25,#n<return>	Determine maximum number of data bytes in an Upload record; n = 1 to 27	No
SET #26,#n<return>	Select serial data format for Upload and Download; 0=Intel 1=MOS 2=Motorola 3=Signetics 4=Tektronix 5=Extended Tekhex	No
SET #27<return>	Set acknowledge character (default = $06)	No

Example 3-4 shows how to Load or Save parameters. The loader checks the validity of the stored data before transferring it to the Satellite Emulator memory.

The system will save what is shown on the SET menu. The parameters shown do not necessarily have to be in effect at the time they are saved. This allows you to use one system to set up default parameters for another system.

NOTE

A SAV Operation may take up to two minutes. Do not interrupt the process.

Example 3-4.
Load and Save

- To Load all the system parameters:
  >LD<return>
- To Load only one section:
  >LD <n><return>
  The section to be Loaded is denoted by n.
- To Save all system parameters:
  >SAV<return>
  Remember this may take up to two minutes.
- To Save only one section:
  >SAV <n><return>
  Again, n is the section number.

ON
OFF

The switches used to control emulation and hard copy parameters are enabled using ON and disabled using OFF.

- To turn on the PPT switch
  >ON PPT

- To turn off all switches
  >OFF -1

- To turn on three switches
  >ON PPT + CAS + BTE

- To display switches
  >ON
  or
  >OFF

Peek Poke Trace
PPT

PPT makes it possible to trace all Reads (Peeks) and Writes (Pokes) to the Target System.

- To trace a particular memory location:

```
>ON PPT
>AC1 = 5550
>CES; WHEN AC1 THEN TRC
>ITR
>SF1, 5000 TO 5FFF

>DRT
```

This displays what happened at location 5550 during the Memory Diagnostic.

Initialize Trace
ITR

ITR causes the trace and break electronics to be initialized just as it is prior to going to RUN. Thus, even though emulation does not occur, you may record information about Peeks and Pokes to the target system.

You may wish to use this capability by tracing only Peeks or Pokes that are qualified in some way or by doing a Toggle Trace (TOT) to turn the Trace on and off under certain conditions. To do this, the ITR command must be executed after setting up the Event System, and prior to executing the command that performs Peeks and Pokes to the target system.

Bus Timeout Enable
BTE

When BTE is ON and the target program is running, an internal watchdog timer will detect if the address strobe has been active for too long a time. In most systems, this condition will be detected by hardware in the target; however, if the target fails to do so the Satellite Emulator will detect the fault. Setting BTE to off will not disable Timeout on Peek/Poke operations.

Fast Timeout
FTO

FTO is used with BTE.

- When FTO is OFF, BTE requires 35840 clock cycles.
- When FTO is ON, BTE will be shorter by a factor of 16 (2240 clock cycles).

Either of these settings will probably be much longer than the amount of delay used by the watchdog circuit in the target system.

On Peek/Poke cycles FTO is always used regardless of the setting.

Disable Bus Error on Peek or Poke
DPB
DBP

DPB makes it possible to ignore the Bus Error signal coming from the target system. This could allow the Emulator to interface with a system that had a faulty memory error detection scheme. The Emulator will generate the Bus Error when one is detected by the built-in watchdog circuit.

- When DPB is ON, Bus Errors are inhibited during Peeks and Pokes, but not during emulation.
- When DPB is OFF, Bus Errors are detected as normal.

Slow Interrupt Enable
SLO

When SLO is ON, Interrupts will not be enabled immediately upon going into emulation; instead a delay of approximately 160 clock cycles must elapse before Interrupts are enabled.

If both SLO and FST are OFF, Interrupts generated by the target system will be inhibited from reaching the MPU.

Fast Interrupt Enable
FST

If FST is ON, Interrupts will be enabled at the moment the Emulator begins executing the target program.

- If both SLO and FST are OFF, Interrupts generated by the target system will be inhibited from reaching the MPU.
- If both the SLO and FST are ON, the effect is the same as FST being ON alone.

Continuous Address Strobe CAS

CAS enables the address strobe to go to the target system while not emulating.

- If CAS is ON, address strobes continue whether emulating or not.
- If CAS is OFF, address strobes are only generated during emulation.

Tri-State Address
TAD

TAD enables tri-stating of the address bus (determining whether the bus continues to output addresses to the target system when not emulating).

- When TAD is off, addresses generated from the program being executed on the emulator card are output by the address bus to the target system.

- When TAD is ON, you can prevent these extraneous addresses from reaching the target system. This will Tri-State the address bus anytime the Emulator is not emulating or doing Peeks or Pokes.

iew Bus Speed Info
SPD

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

- If SPD is ON, the column will instead display a number that relates to the access time of devices on the bus. If one access displays a "4" and another a "5", the latter bus cycle took one more clock cycle to complete than the former. Access times greater than or equal to 10 cycles will display as "+".

Introspective Mode
IM

When IM is ON, the emulator itself becomes the "target system." This gives you the capability of writing your own scope loops and diagnostics.

While IM is ON, only addresses greater than $80000 may be overlaid. There is 3K of RAM set aside for the user in the memory map of the emulator ($7000-$7BFF). Special functions 40-49 are used to execute subroutines in internal RAM.

Copy Switch
CPY

This switch allows you to produce hard copy of your emulation session. When CPY is on, data sent to the controlling port will also be echoed to the other port. This is useful for making hard copy of emulation sessions or monitoring computer control (CCT) commands.

SECTION 4
OPERATION

4.1 INTRODUCTION

This section describes the procedures for operating the Satellite Emulator and error codes that may occur during the process. The information here presumes that you have read the previous sections.

THE SATELLITE EMULATOR WILL NOT OPERATE PROPERLY UNLESS IT HAS BEEN CORRECTLY INSTALLED AND SET UP. Information on system communications and serial interfacing (beyond initial installation) is in Section 6, Interfacing and Communications.

4.2 REGISTER OPERATORS

The register operators are used to assign values to registers within the 680XX and the emulator, and to display these values.

Values should be assigned to the necessary registers before they are used in statements. The program counter and stack pointer should be loaded before beginning emulation. This is done by entering the RNV command. Table 4-1 lists the registers recognized by the system.

Table 4-1
Registers

OPERATOR	DESCRIPTION	HOW USED
AC1, AC2	address comparator registers 1 and 2	Event Monitor System
CL	count limit comparator register	Event Monitor System
DC1, DC2	data comparator registers 1 and 2	Event Monitor System
LSA	Logic State Analyzer comparator register	Event Monitor System
S1, S2	status comparator registers 1 and 2	Event Monitor System
SIA	Special Interrupt address register	Event Monitor System
OVE	Overlay Memory enable register	Memory Mode
OVS	Overlay Memory speed register	Memory Mode
MMP	memory space pointer	Memory Mode
MMS	Memory Mode access status register	Memory Mode
MMD	Access status register for destination of block move	Memory Mode
DFB	default base register	Miscellaneous
GDn (0-7)	general purpose data register	Miscellaneous
GRn (0-7)	general purpose range register	Miscellaneous
PC	program counter register	Motorola CPU Register
USP	User stack pointer register	Motorola CPU Register
SSP	Supervisor stack pointer register	Motorola CPU Register
VBR	Vector base register (32 bit)	Motorola CPU Register
SFC	Source function code register (3 Bit)	Motorola CPU Register
DFC	Destination function code register (3 bit)	Motorola CPU Register
SR	status register	Motorola CPU Register
An (0-6)	CPU address registers	Motorola CPU Register
Dn (0-7)	CPU data registers	Motorola CPU Register
IIB	Instruction Input Buffer	Bus Error Register*
DIB	Data Input Buffer	Bus Error Register*
DOB	Data Output Buffer	Bus Error Register*
FA	Fault Address Register	Bus Error Register*
FMT	Format Register	Bus Error Register*
SSW	Special Status Word	Bus Error Register*
MSK	Internal Information	Bus Error Register*
Rn (0-14)	Internal Information	Bus Error Register*

*See Section 4.9

DR

The Display Registers command is used to display the CPU registers in a fixed format. See figure 4.1. Its format is:

>DR<return>

Registers (except event comparators) will be displayed in long word format (32-bit).

4.2.1 Loading a Register

Example:

```
>R0
>$00000000
>R0=5678
>R0
>00005678
>DR
```

4.2.2 General Registers

GD0-7 and GR0-7 are miscellaneous registers used to save key strokes when you are using simple integers, ranges, or Don't Cares. They are used as follows:

- GR0-7 integers or ranges for addresses
- GD0-7 integers or Don't Cares for data

Examples:

```
>GR0=1000 TO 2FFF
>MAP GR0
>DM

>GR0= 10 TO 20
>DB GR0

>GR0 = OLEN50
>DB GR0
>MAP GR0
>FIL GR0, $90
>DB GR0
```

These examples display only the series of keystrokes you would enter; the system response is not shown.

```
>DR

    - 0 -     - 1 -     - 2 -     - 3 -     - 4 -     - 5 -     - 6 -     - 7 -
D = 0000FFFF  0000FFFF  0000FFFF  7820FFFC  0000FFFF  0000FFFF  0000FFFF  0000FFFF
A = 0004FFFF  0000FFFF  0000FFFF  0000FFFF  0000FFFE  0000FFFF  0000FFFF
 PC=000000  VBR=000000  SFC=0     DFC=0     SSP=0000    USP=0  SR=..S7.....

>
```

Figure 4-1. Display Registers Format

4.3 EMULATION

The basic function of the Satellite Emulator is the emulation of microprocessors. When emulation is initiated, the Satellite Emulator will run the target system program transparently and in real time, just as the target system microprocessor would, or one instruction at a time. Essentially, emulation lets you "see" into the logical environment of the emulated microprocessor.

The operators associated with emulation are:

- Run - RUN
- Step and Stop - STP
- Run With Breakpoints - RBK
- Wait - WAIT

Only used to start emulation:

- Run With New Vectors - RNV
- Run With New Vectors and Breakpoints - RBV
- Load New Vectors - LDV

To reset processor:

- Reset - RST

4.3.1 Run RUN

The Run operation executes the target system program in real time until you stop it or it encounters an access violation associated with the defined memory map. The Run Prompt R> will be present during RUN. See example 4-1.

4.3.2 Step and Stop STP

The Satellite Emulator combines Step and Stop into one mnemonic. Step takes you through the target system program one instruction at a time. Stop is used to break emulation during a Run or Run With Breakpoints.

- If emulation is in progress, keying in STP will cause the Satellite Emulator to halt emulation.
- If STP is entered while emulation is not in progress, one program instruction will be executed. (If you want to execute more instructions, press the slash key (/) as many times as desired.)

See Example 4-1.

4.3.3 Run With Breakpoints RBK

Run With Breakpoints (RBK) is the same as a Run operation except that break operators within the Event Monitor System are honored and will stop program execution when encountered. The Run prompt R> will be present during RBK.

These examples show how to start emulation:

Example 4-1.
Run, Run With Breakpoints, Step, and Stop

- To initiate a Run:
  >RUN<return>
- To initiate a Run With Breakpoints:
  >RBK<return>
- To stop a Run or Run With Breakpoints:
  R>STP<return>
- To Step through emulation (emulation not currently in progress):
  >STP<return>

4.3.4 Vector Loading and Running With Vectors
LDV
RNV
RBV

Since the 680XX uses start-up vectors, three additional operators are used to accomodate them. LDV loads the program counter and stack pointer target system vector table into the 680XX registers. RNV and RBV are Run commands that preload the vectors. RNV is identical to LDV;RUN. RBV is the same as LDV;RBK. The Run prompt (R>) is present during the RNV and RBV.

- To load vectors:
  >LDV<return>

- To run with vectors:
  >RNV<return>
  or
  >LDV;RUN<return>

  Note that the two examples cause identical results. RNV loads the vectors, then starts emulation. The same can be accomplished using the two commands LDV and RUN.

- To run with vectors and breakpoints:
  >RBV<return>
  or
  >LDV;RBK

4.3.5 Reset
RST

The Reset operator (RST) can be used at any time to reset your target system except during emulation.

NOTE:

Ctrl Z differs from RST in that Ctrl Z resets the emulator, but does not reset the target system.

4.3.6 Wait
WAI

The Wait operator causes the Satellite Emulator to delay executing the command statement following it until emulation is broken for some reason (an event detector within the Event Monitor System or access violation of the memory map, for example). See the following example.

- The format for the Wait operator is:
  >RBK;WAI;<command><return>

- For example:
  >RBK;WAI;D3<return>

- Note that the semicolon is used to separate the commands.

CAUTION
THE EMULATOR MAY HANG UP WHILE USING THE WAIT OPERATOR IF EMULATION IS NOT AUTOMATICALLY BROKEN. TO ESCAPE THIS CONDITION, USE THE USER-DEFINED RESET CHARACTER

4.3.7 Cycle
CYC

CYC allows the target program to be executed in the form of individual bus cycles rather than by instruction. When instructions are broken up into bus cycles, there are only three types that are encountered while executing CYC:

- A Read cycle (also called a Peek)

- A Write cycle (also called a Poke)

- A Read-Write-Modify cycle

CYC does not actually execute the target program the way a STP command does: the individual cycles are simulated by performing Peeks or Pokes to the target system. Since the Emulator does not go to RUN while executing the Bus Cycles, the Trace Memory does not keep a record of transactions to and from the target system.

4.4 MEMORY MODE

Memory Mode allows you to examine or change the contents of the target system memory. Each address is accessed and displayed individually, with easy-to-use scrolling features. Data at each address can be displayed and/or entered in any number base you select.

The following sections explain how to enter and exit Memory Mode, use the pointer, scrolling features, long word, word and byte modes, and how to examine and change the target system memory.

4.4.1 Entering and Exiting Memory Mode MM or M MX or X

MM or M is used to enter Memory Mode. If no entry address is specified, the address will default to the value of MMP. Upon entry, the memory location is read and the address and data residing there are displayed preceding the prompt. A <return> will increment the address, (unless a LST was entered previously).

- To enter the target system memory space at a specific address:
  >M <address><return>

- To enter the target system memory space at the default address:
  >M<return>

- You should specify the data length when entering memory mode, unless current system default is desired.

  - To change the data size globally
    >SZ.B/W/L
    >MM

  - To change the data size only for memory mode
    >MM.B/W/L

(Data length is discussed further in Section 4.4.4)

- The system will respond with one of the three memory mode prompts:

  - byte mode $000000 $FF>

  - word mode $000000 $FFFF>

  - long word mode $000000 $FFFFFFFF>

- To exit Memory Mode:
  >MX<return> or
  >X<return>

4.4.2 Memory Mode Pointer MMP

The Memory Mode pointer, when invoked, will display the last address invoked in the Memory Mode since power-up. The key sequence is shown in the following example.

You can also change the Memory Mode pointer to a value you select by entering the desired value before the <return>.

- To display the last memory space address examined:
  >MMP<return>

- To change the Memory Mode pointer:
  >MMP=<address><return

4.4.3 Scrolling NXT LST , .

Once you have entered Memory Mode at a specific address, you can scroll to higher or lower addresses. The NXT and LST operators determine the default direction of sequential memory accesses. When you enter NXT after the prompt, the addresses are incremented between each access. LST entered after the prompt causes the addresses to be decremented. The power-up default is NXT. These commands are useful for storing lists of values into memory.

When a comma or period is entered in response to the Memory Mode prompt, addresses are incremented (with the period) or decremented (with the comma) and the next location displayed. These are used to temporarily override NXT and LST.

- To scroll to the next higher address:
  >NXT <return>
  or
  >.
  or
  <return>

- To scroll to the next lowest address:
  >LST<return>
  or
  >,

4.4.4 Word, Byte, Long Word Mode SZ.W or WDM SZ.B or BYM SZ.L or LWM .B .W .L

If you wish to scroll through the memory spaces one byte (8 bits) at a time, invoke SZ.B SZ.W is used to scroll in the word mode (16 bits). SZ.L scrolls you through Memory Mode in a long word mode (32 bits). The system will default to a byte mode.

SZ.W, SZ.B, and SZ.L should be considered global defaults that affect all operations, not just Memory Mode. The dot operators (.B, .W, .L) can be appended to specific commands to temporarily override the SZ.<B/W/L> operators, just as the comma and period temporarily override NXT and LST commands.

- To scroll in the byte mode:
  >SZ.B;M or M.B

- To return to the word mode:
  >SZ.W;M or M.W

- To scroll in the long word mode:
  >SZ.L;M or M.L

4.4.5 Examining and Changing Values

Now that you can access Memory Mode, work with the pointers, and scroll higher and lower in either the byte, word, or long word mode, it's time to discuss how to change values.
When you enter an address or scroll to a new address, the CRT terminal will display the value at that location. To change the value, simply enter a new value followed by a <return>. A string of values can also be entered, each separated by a comma. This will store the values to consecutive locations according to the current NXT or LST mode.

>M address <ret>
Emulator responds.
<addr> <current data> >
Enter new data or <ret>

- To change a single value at one location:
  >$47FF<return>

- To change a series of values at consecutive locations:
  >M<return>
  The emulator will respond with:
  <address> <current data>
  Then enter:
  >$1,$2,$3,$4<return>

The emulator loaded the first location with 1, the second with 2, the third with 3, and the fourth with 4. The address increments to the next location following the last word that received data (according to current NXT or LST mode).

4.4.6 Memory Mode Status

MMS
MMD
SP
SD
UP
UD
CPU
SC0-7
[TGO]
[OVO]
[NRM]

Memory Mode status (MMS) allows you to access any of the eight 680XX memory spaces: Supervisor program, Supervisor data, User program, User data, or CPU space or unused codes (SCØ, SC3 and SC4). See the following example.

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

MMD is the same as MMS except that MMD is used with Block Move and Block Verify.

In addition to specifying the memory space, you may specify target only access (TGO) or overlay only access (OVO). If you have overlay mapped and you want to see what's in the target system without destroying what you have mapped, specify TGO. If you have

the target system mapped and want to look at overlay only, specify OVO. When you use OVO, no bus cycles enter the target system.

- The general format is:

```
>MMS = SP
       SD
       UP + TGO
       UD + OVO
       CPU
```

  Only one of the possible memory spaces can be selected.

- To access Supervisor program space:
  >MMS = SP<return>

- To access Supervisor data space:
  >MMS = SD

- To access User program space:
  >MMS = UP

- To access User data space:
  >MMS = UD

- To access CPU space:
  >MMS = CPU

- To access overlay independent of the target:
  >MMS = SC5 + OVO

- To access target even if address is mapped by overlay:
  >MMS = UD + TGO

- To return to default mode (after previously specifying TGO or OVO):
  >MMS = SP + NRM

4.4.7 Displaying a Block of Memory and Finding a Memory Pattern

Two additional operators need explanation at this point. Though you cannot be in Memory Mode when you invoke them, these operators affect memory examination.

DB

DISPLAY MEMORY BLOCK. To display a block of memory, use the DB operator. The display format includes line address and hexadecimal byte data; ASCII-equivalent characters are displayed when in byte mode.

- To display a block of memory:
  >DB [.B or .W or .L] <address range> <return>

FIN

FIND MEMORY PATTERN. To find a specific bit pattern in memory the FIN operator is used.

- To find a bit pattern in memory:
  >FIN [.B or .W or .L] <range>, <data> <return>

- To find a bit pattern using Don't Cares (either form):
  FIN.W 1000 TO 2FFF, 60XX
  or
  FIN.L $C000 LEN 200,$7FFE1234 DC 1F80000

  The emulator will return:
  $<address> = <data>
  to indicate where the bit pattern has been found.

4.5 MEMORY MAPPING AND THE OVERLAY MEMORY

Memory mapping is used in conjunction with the Overlay Memory. If you wish to use the Overlay Memory during emulation, you will have to define the memory map first.

The Overlay Memory is RAM in the Satellite Emulator with appropriate address and control logic. It is locatable in 2K-byte segments throughout the system. Size of the Overlay Memory ranges from 32K-bytes to 512K-bytes, depending on the option you selected at time of purchase.

The Overlay Memory can be mapped into the address space of a target system so you can load the target system program into it; the target system program can then be edited, positioned in the target system address space as desired, and the program is executed in real time as if it resided totally in the target system. Overlay Memory is also useful for checking programs not yet committed to PROM. When programming data is correct, it can be uploaded to a PROM programmer.

```
  DB.B 1000 TO 1100
001000  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
001010  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
001020  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
001030  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
001040  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
001050  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
001060  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
001070  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
001080  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
001090  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0010A0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0010B0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0010C0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0010D0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0010E0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0010F0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
001100  FF                                              .
  >

  DB.W 1000 LEN $4F
001000    FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF
001010    FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF
001020    FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF
001030    FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF
001040    FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF
  >DB.L 1000 LEN $4F
001000    FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
001010    FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
001020    FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
001030    FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
001040    FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
  >
```

Figure 4-2.
Display Memory Block Format

4.5.1 Memory Block Attributes

The first step in using the Overlay Memory is assigning one of four attributes to the memory ranges. The ranges specified must fall on 2K-byte boundaries. If you specify a range that does not, the Satellite Emulator will expand the range until the endpoints fall on such a boundary.

The following paragraphs describe the attributes of the four types of memory blocks possible.

NOTE:

The memory block attribute operators, :RO, :RW :TGT, and :ILG, are always preceded by a colon.

:RO

READ ONLY. Memory blocks marked with this attribute are write-protected: no target system write cycle can change the data. Note, however, that the emulator can write to that space. This is most often used to emulate instruction memory that would be placed in ROM or PROM. If a write cycle is made to a memory block that has this attribute, a write access violation breakpoint stops program execution and displays a message to that effect.

:RW **READ/WRITE.** Memory blocks marked with this attribute are available for read or write access. No error breakpoints ever occur as a consequence of access to these blocks.

:TGT **TARGET.** Memory blocks marked with this attribute are assigned to the target system. All memory accesses marked target by the microprocessor go directly to the target system memories (if any).

:ILG **ILLEGAL.** Memory blocks marked with this attribute are illegal for all types of access. Normally these blocks are useful for marking memory that should never be referenced by the program at all. In other words, if the program references these addresses, there is something wrong with the program. If this ever occurs, a memory access violation breakpoint will stop program execution and display a message to that effect.

4.5.2 Memory Mapping Operators

Three operators are used in conjunction with the memory type operators for memory mapping.

MAP **SET MEMORY MAP.** The Set Memory Map Operator (MAP) is used in conjunction with the memory type operators (Read Only, Read/Write, etc.) to define the memory map of the system.

- The general format for setting up a block of overlay memory is:
  >MAP <range>: [memory type]<return>

- To set a memory space as Read Only, $3000 bytes long, responding to addresses 0 to $2FFF:
  >MAP 0 to $2FFF:RO<return>
  This would contain 6 blocks of 2K bytes each.

- The same format is used for setting any other of the four memory types. If the memory type argument is not supplied, the default is read/write (:RW).

DM **DISPLAY MEMORY MAP.** This operator allows you to display the memory map currently in effect.

- To display the memory map in effect:
  >DM<return>

CLM **CLEAR MEMORY MAP.** This operator clears the memory map currently in effect. Be sure you are ready to clear it before invoking the operator.

- To clear the memory map in effect:
  >CLM<return>

```
  DM
MEMORY MAP:
OVERLAY ENABLED FOR (OVE=) SUP+USR+PGM+DTA,  SPEED (OVS) = 0
MAP $000000 TO $000FFF : TGT
MAP $001000 TO $0027FF : RW
MAP $002800 TO $002FFF : TGT
MAP $003000 TO $0047FF : RO
MAP $004800 TO $008FFF : TGT
MAP $009000 TO $009FFF : ILG
MAP $00A000 TO $FFFFFF : TGT
  >
```

Figure 4-3.
Display Memory Map Format

4.5.3 Overlay Memory Operators

OVE

OVERLAY MEMORY ENABLE. The OVE operator allows you to load values that determine which 680XX memory status space the Overlay Memory responds to. The current value is shown when the memory map is displayed. Factory default is SD + SP + UD + UP.

- The general format for OVE is:
  >OVE = SD + UP + UD<return>
  >OVE = ALL (enabled for all spaces, 5 for 68000/68008, 8 for 68010)

OVS

OVERLAY MEMORY SPEED. OVS is a variable that causes the Overlay to return a DTACK to the MPU. The current value of OVS is displayed when the memory map is displayed. OVS can be loaded with the values 0 through 7 as shown in the next example.

- To set OVS:
  >OVS = (0 through 7)<return>

NOTE

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

Table 4-2
OVS Values

VALUE	DESCRIPTION
OVS = 0	No DTACK; DTACK supplied by target system
OVS = 1	No delay, AS returned to 680XX as DTACK
OVS = 2	+1 cycle delay
OVS = 3	+2 cycles delay
OVS = 4	+3 cycles delay
OVS = 5	+4 cycles delay
OVS = 6	+5 cycles delay
OVS = 7	+6 cycles delay

Load Overlay Memory
LOV

LOV. LOV loads Overlay Memory with your target system program. The data is automatically verified during the operation. (The target is not written to.)

The key sequence for loading the Overlay Memory is given in the following example. The argument specifies the address range of the target system memory from which to move data to the Emulator Overlay Memory. Note that the Overlay Memory may also be loaded via the block move command.

The key sequence is:

>LOV X TO Y<return>
or
>LOV X LEN W<return>

Verify Overlay Memory
VFO

VFO. VERIFY OVERLAY MEMORY. VFO is used to verify that the program you have loaded into Overlay Memory matches the program in your target system memory. The following example shows the key sequence.

- The key sequence is:
  >VFO X TO Y<return>

- If any differences occur, the emulator will return:
  <address> = XX NOT YY

  The <address> is where the misverify occurred. XX denotes the data present in Overlay Memory and YY is the data at that location in target system memory.

Fill Operator
FIL

FILL. FIL is used to fill the memory space of the emulator or target system with a constant. The constant may be written to overlay memory and target read/write memory.

The format for the Fill operator is:

FIL.B x TO Y, Z
or
FIL.L x LEN W,Z

The first argument specifies the address range to be filled with the constant specified in the second argument. The second argument may be byte, word, or long word depending on the global default or the type of extension (.B, .W, or .L).

Verify Block Data
VBL

VERIFY BLOCK DATA. The VBL operator is used in conjunction with the FIL operator. Once the Overlay Memory has been filled with constant data (via FIL), this data can be verified with the VBL command. The key sequence shown in the following sequence is much like the key sequence for FIL.

The general format is:

>VBL <.B .W or .L> <address range>, <argument><return>

The VBL operator verifies that the address range contains the argument.

Clear Overlay
CLM

CLEAR OVERLAY MEMORY MAP. The CLM operator clears all addresses and data from the Overlay Memory map. Clear the map and data by typing CLM <return>, enter the new map, then load new data.

The format for clearing Overlay Memory is:
>CLM<return>

Block Move
Verify Block Move
BMO
VBM

The Block Move operator moves a block of data from one location within the emulator or target system memory, or Overlay Memory to another via a source/destination format. The space you move data into should be designated as writeable. You can also specify the 68010 status space (Supervisor program, Supervisor data, User program, User data).

The format for a block move is:
>BMO<source range>, [source space,]
<destination start address>,[destination]<return>

"Space" refers to the 680XX status space: SP, SD, UP, or UD.

The source space for a Block Move is determined by the value in MMS. The destination space is determined by the value stored in MMD.

- You can set MMS and MMD prior to execution:

```
>MMS = SD
>MMD = SCØ
>BMO = GR1,$1000
```

- Byte or word may be specified. The space code may be typed on the same line.

```
>BMO.W 300 LEN 40,SP,10300,UP
```

- You can verify byte data between the target and overlay memory:

```
>VBM.B 2000 TO 3FFF,SPTGO,2010,SP+OVO
```

Verify Block Move has the same syntax as Block Move except that VBM only verifies that the source and destination blocks are identical.

4.6 THE TRACE MEMORY AND DISASSEMBLER

The Trace Memory records the history of the program execution. It may be used in conjunction with a disassembly routine to format the trace data. The mnemonics provided by the disassembled display ensure more rapid analysis of your data.

During emulation, the activity of the executing program is recorded continuously and stored in the Trace Memory. At any point in the process, the program execution can be stopped. The address, data, and control bus of the last series of cycles can be displayed and scolled on a CRT terminal or output to a printer. The entire contents or a "window" of cycles occurring between specified bus or instruction cycles can be dumped. If something unexpected happens during program execution, the Trace Memory provides a record that can be reviewed to determine what happened. The Event Monitor System can be used to qualify, start recording of data into the Trace Memory, and stop the recording process.

By enabling the PPT switch, all PEEK or POKE operations will be traced in addition to normal emulation being traced.

The Trace Memory is 72 bits wide and 2048 words deep; two words are used for marks, leaving 2046 words. It cannot be accessed by the user during emulation.

A trace counter supplies the address to the Trace Memory and can be incremented with each cycle. It is a 12-bit counter (only eleven are used) with count mode logic. It has three modes of operation:

- count never
- count every bus cycle (only available during a Run mode)
- count every bus cycle when qualified by trace directive from state decode logic (only available during a Run mode)

A Disassembler is available for use with the Trace Memory. This allows you to display or print Trace Memory in an easy-to-read format similiar to a program listing. Figure 4-4 shows printouts of the Trace Memory and of the Trace Memory with Disassembly. The 4 line header may be garbled because no linefeed is sent from the disassembler; however the rest of the printout will be legible.

A "page" of Trace Memory is defined as the number of lines on the CRT terminal, less three. All scrolling is done by pages, with both raw Trace Memory data and disassembled data.

4.6.1 Display Raw Trace DRT

DRT can be used to display Trace Memory data in bus cycles if you do not wish to use Disassembly to display instruction cycles.

Invoking DRT causes the Satellite Emulator to display a page of bus cycles of the Trace Memory. More or less cycles may be displayed by specifying an address or address range in an argument following the operator. If a single address is specified, the system will display the specified address and the previous 20 bus cycles. The Trace Memory holds 2,046 cycles; therefore, 2,046 is the highest number allowable as input. Note that the raw Trace Memory contains the 16-bit status and control word (described in Section 5.2.6).

- To display the last page of bus cycles:
  >DRT<return>

- To display a specific line number and the previous 20 cycles:
  >DRT<address><return>

- To display a range of line numbers:
  >DRT<range><return>

Note that the range is a range of bus cycles, not the address recorded in the Trace Memory.

RAW TRACE

```
LINE ADDRESS DATA R/W     FC        IPL      LSA  - 8  7  -  Ø
 #20 00100A> 3080  R  OVL SP         Ø       %11111111 %11111111
 #19 00100C> B03C  R  OVL SP         Ø       %11111111 %11111111
 #18 00100E> 0039  R  OVL SP         Ø       %11111111 %11111111
 #17 001010> 6C02  R  OVL SP         Ø       %11111111 %11111111
 #16 001012> 5E40  R  OVL SP         Ø       %11111111 %11111111
 #15 001014> 5740  R  OVL SP         Ø       %11111111 %11111111
 #14 001016> 41FA  R  OVL SP         Ø       %11111111 %11111111
 #13 001018> 03F0  R  OVL SP         Ø       %11111111 %11111111
 #12 00101A> 3140  R  OVL SP         Ø       %11111111 %11111111
 #11 00101C> FF00  R  OVL SP         Ø       %11111111 %11111111
 #10 00101E> 1140  R  OVL SP         Ø       %11111111 %11111111
  #9 001308 <0037  W  OVL SD         Ø       %11111111 %11111111
  #8 001020> FFF7  R  OVL SP         Ø       %11111111 %11111111
  #7 001022> 60E0  R  OVL SP         Ø       %11111111 %11111111
  #6 0013FF <  37  W  OVL SD         Ø       %11111111 %11111111
  #5 001024> 4E71  R  OVL SP         Ø       %11111111 %11111111
  #4 001004> B03C  R  OVL SP         Ø       %11111111 %11111111
  #3 001006> 0039  R  OVL SP         Ø       %11111111 %11111111
  #2 001008> 6C02  R  OVL SP         Ø       %11111111 %11111111
  #1 00100A> 3080  R  OVL SP         Ø       %11111111 %11111111
  #Ø BREAK
```

DISASSEMBLED TRACE

```
>DTB
SEQ#  ADDR   OPCODE MNEMONIC  OPERAND FIELDS   BUS CYCLE DATA
-------------------------------------------------------------------------
0041 001008 6C02     BGE.S    $00100C
0040 00100A 3080     MOVE.W   D0,(A0)  001408<0036
0039 00100C B03C0039 CMP.B    #$39,D0
0036 001010 6C02     BGE.S    $001014
0035 001012 5E40     ADDQ.W?? #7,D0
0034 001014 5740     SUBQ.W   #3,D0
0033 001016 41FA03F0 LEA.L    $001408(PC),A0
0031 00101A 3140FF00 MOVE.W   D0,-$100(A0)  001308<003A
0029 00101E 1140FFF7 MOVE.B   D0,-9(A0)  0013FF<3A
0026 001022 60E0     BRA.S    $001004
0023 001004 B03C0039 CMP.B    #$39,D0
0021 001008 6C02     BGE.S    $00100C
0019 00100C B03C0039 CMP.B    #$39,D0
0017 001010 6C02     BGE.S    $001014
0016 001012 5E40     ADDQ.W?? #7,D0
0015 001014 5740     SUBQ.W   #3,D0
0014 001016 41FA03F0 LEA.L    $001408(PC),A0
0012 00101A 3140FF00 MOVE.W   D0,-$100(A0)  001308<0037
0010 00101E 1140FFF7 MOVE.B   D0,-9(A0)  0013FF<37
0007 001022 60E0     BRA.S    $001004
0004 001004 B03C0039 CMP.B    #$39,D0
```

MEMORY DISASSEMBLER PROGRAM

```
>DIS 1000
001000 303C003A             MOVE.W   #$003A,D0
001004 B03C0039             CMP.B    #$39,D0
001008 6C02                 BGE.S    $00100C
00100A 3080                 MOVE.W   D0,(A0)
00100C B03C0039             CMP.B    #$39,D0
001010 6C02                 BGE.S    $001014
001012 5E40                 ADDQ.W   #7,D0
001014 5740                 SUBQ.W   #3,D0
001016 41FA03F0             LEA.L    $001408(PC),A0
00101A 3140FF00             MOVE.W   D0,-$100(A0)
00101E 1140FFF7             MOVE.B   D0,-9(A0)
001022 60E0                 BRA.S    $001004
001024 4E71                 NOP
```

Figure 4-4.
Trace Memory Format

NOTE

If a double question mark (??) appears on a line of disassembled output, this indicates that the disassembler does not know if the instruction was executed. This could be due to any of three reasons.

- The preceding conditional branch is jumping around this instruction.
- The instruction is a single-word type.
- The instruction does not cause data to appear on the bus.

4.6.2 Disassemble Trace DT

This operator will cause the Trace Memory to be disassembled and output to the controlling port (computer or terminal). If no range argument is specified, the last instruction executed is disassembled. The output of the DT operator in this instance has its linefeed suppressed. Thus, by repeating the operators Step and Disassemble Trace, a continuous Disassembly is formed (>STP;DT<return>). Some information can't be disassembled because of qualifiers such as TOT or TRC (within the Event Monitor System).

- To disassemble the last instruction executed:
  >DT<return>

- To disassemble a range:
  >DT<single value or range><return>
  The single or range values are sequence numbers where 0 is the number for the most recent instruction. Entering a single value will disassemble that value and the previous page. A range disassembles that range of sequence numbers.

- To initiate a continuous Disassembly:
  STP;DT<return>

4.6.3 Disassemble Previous and Following Trace DTB DTF

These two operators will scroll you through the disassembled Trace Memory a page at a time. This key sequence is also shown in Example 4-19.

NOTE

The sequence numbers in DT, DTB, and DTF correlate with the line numbers displayed by DRT, which are bus cycle numbers.

- To disassemble the previous page:
  >DTB<return>

- To disassemble the following page:
  >DTF<return>

4.7 SOFTWARE DEBUGGING WITHOUT TARGET SYSTEM HARDWARE

An added feature of the Satellite Emulator is its ability to debug software without being physically connected to your target system. This is accomplished with the Null Target Software Simulation Tool.

The procedure consists of mapping memory space and loading the Overlay Memory with your program. With the Null Target attached to the emulator pod assembly, you can now use the features of the emulator to execute program code and test modules.

4.8 ERROR HANDLING AND CODE

When an error occurs, the system will print a question mark (?) on the screen, directly below or just after the point in the input (Figure 4.5) that caused the error. See Figure 4-5.

When the question mark appears on the screen, you should key in a question mark in return to find out the error message. Table 4-3 lists the errors that may occur by code and their messages as displayed on the CRT. If you need additional help, call Customer Service for ES Products.

If you are operating the Satellite Emulator under host system control, only two errors are likely to occur, assuming the host system software has been fully debugged: Error 6, a checksum error and Error 34, a read-after-write error. The host system can be set up to return a question mark to the emulator and use the error code number to consult its own table for further action.

Figure 4-5.
Error Recognition

```
  DB.X 0 TO S2F
     ?
 >?
ERROR #1
EXPRESSION HAS NO MEANINGFUL RELATION TO THE REST OF THE COMMAND
 >
 >
 >DST
    ?
 >?
ERROR #5
UNDEFINED SYMBOL OR CHARACTER DETECTED
 >
 >
 >SET #20,55
           ?
 >?
ERROR #42
ILLEGAL SETUP SET VALUE
 >
```

Table 4-3.
Error Codes

CODE	MESSAGE DISPLAYED	COMMENTS
	EXPRESSION HAS NO MEANINGFUL RELATION TO REST OF THE COMMAND	Often caused by entering symbols out of context. DR and BRK are both legal operators but entered together as DR BRK would cause this error message.
5	UNDEFINED SYMBOL OR INVALID CHARACTER DETECTED	Generally 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 that of the received data. Refer to download for error handling during computer control.
7	BAD STATUS = ...RETURNED FROM EMULATOR CARD	Contact Customer Service for ES Products.
8	ARGUMENT IS NOT A SIMPLE INTEGER OR INTERNAL RANGE	Don't Cares are not allowed in this context.
9	NO MORE OVERLAY MEMORY AVAILABLE	You haven't cleared the map or you are trying to map in more memory than is allowed.*
	MULTIPLE-DEFINED EVENT GROUP	Only one group may be referenced in any event clause; caused by trying to mix event register groups in an event clause e.g., 2 WHEN AC1.2 THEN BRK would cause this error.
11	ILLEGAL ARGUMENT TYPE FOR EVENT SPECIFICATION	
13	ARGUMENTS MUST BE A SIMPLE INTEGER	
16	OPERATION INVALID FOR THESE ARGUMENT TYPES	Often 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	
18	TOO MANY ARGUMENTS IN LIST...(9 MAX)	
19	INVALID GROUP NUMBER...(NOT IN 1-4)	
23	OPERATION INVALID FOR THESE ARGUMENT TYPES	See error 16.

*Contact Applied Microsystems for optional Overlay Memory expansion.

CODE	MESSAGE DISPLAYED	COMMENTS
24	BASE ARGUMENT MUST BE A SIMPLE INTEGER	Argument should be #0 to #16.
26	RANGE TYPE ARGUMENT NOT ALLOWED AS DATA	
27	ADDRESS ARGUMENT MUST BE A SIMPLE INTEGER	
29	ILLEGAL DESTINATION - SOURCE TYPE MIX	Caused by trying to store don't care data into a range variable and other similar operations.
31	RANGE START AND END ARGUMENTS MUST BE SIMPLE INTEGERS	
32	RANGE END MUST BE GREATER THAN RANGE START	6 LEN 1 is not a valid range
33	RANGE START AND END ARGUMENTS MUST BE SIMPLE INTEGERS	
34	READ AFTER WRITE-VERIFY ERROR	Downloaded data is verified on a byte-by-byte basis. The error message contains the location and results of the comparison.
35	WARNING - DATA WILL BE LOST WHEN EMULATION IS BROKEN	Caused by attempting to store into 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 will show 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.
38	NO ROOM...BREAKPOINT CLAUSES TOO NUMEROUS OR COMPLEX	
39	INVALID GROUP NUMBER...(NOT IN 1-4)	
40	ILLEGAL SELECT VALUE	First argument after SET operator is invalid.
41	INCORRECT NUMBER OF ARGUMENTS IN LIST	
42	ILLEGAL SETUP SET VALUE	The argument nearest to the "?" is illegal.
43	"WHEN" CLAUSE REDUCED TO NULL FUNCTION	Caused by such constructs as "WHEN AC1 AND NOT AC1."

CODE	MESSAGE DISPLAYED	COMMENTS
44	INTERNAL ERROR...NULL SHIFTER FILE	Contact Customer Service for ES Products.
45	MAP CANNOT BE ACCESSED DURING EMULATION	The map hardware is constantly used by the emulating processor during emulation.
46	ARGUMENT MUST BE AN INTERNAL RANGE	
47	16-BIT RANGE END LESS THAN START	
48	ILLEGAL MODE SELECT VALUE	
49	INVALID GROUP NUMBER...(NOT IN 1-4)	
50	INVALID GROUP NUMBER...(NOT IN 1-4)	
51	SAVE/LOAD INVALID ARGUMENT VALUE	
52	DISPLAY BLOCK NEEDS AN IRA ARGUMENT	
53	EEPROM WRITE VERIFY ERROR	Data in the EEPROM is verified during the SAV operation. (The store operation is retried many times before the error is generated.) EEPROMs have a finite write cycle life. The EEPROM in your emulator is warranted for one year. Contact Customer Service for ES Products.
54	ATTEMPT TO SAVE/LOAD DURING EMULATION	
55	EEPROM DATA INVALID DUE TO INTERRUPTED SAVE	Previous SAV was interrupted by a reset or power off.
56	TRACE DATA IS INVALID DURING EMULATION	
57	INVALID GROUP NUMBER (NOT 1-4)	
58	IMPROPER NUMBER OR ARGUMENTS	
59	ARGUMENT MUST BE AN INTERNAL RANGE	
60	ARGUMENT MUST BE A SIMPLE INTEGER	
61	IMPROPER NUMBER OF ARGUMENTS	
62	CANNOT STORE THIS VARIABLE DURING EMULATION	
63	ILLEGAL ARGUMENT TYPE	
64	ARGUMENT TOO LARGE	Caused by entering range or integer values with the DRT command that include numbers greater than #2045.

	ILLEGAL RANGE	
66	STATUS CONSTANTS CANNOT BE ALTERED	
67	TOO MANY "WHEN" CLAUSES	
68	COMMAND INVALID DURING EMULATION	
70	CANNOT INITIALIZE VECTORS DURING EMULATION	Typed LDV, RNV, RBV during emulation.
71	UNKNOWN EMULATOR ERROR	Call Customer Service for ES Products.
72	INCOMPATIBLE EEPROM DATA	Previous data save was not from 680XX emulator system.
73	OVE MUST BE SET TO [SUP, USR] AND [PGM, DTA]	
74	COMMAND INVALID DURING EMULATION	
75	INVALID RECORD TYPE	Download routine received invalid record type code.

4.9 BUS ERRORS

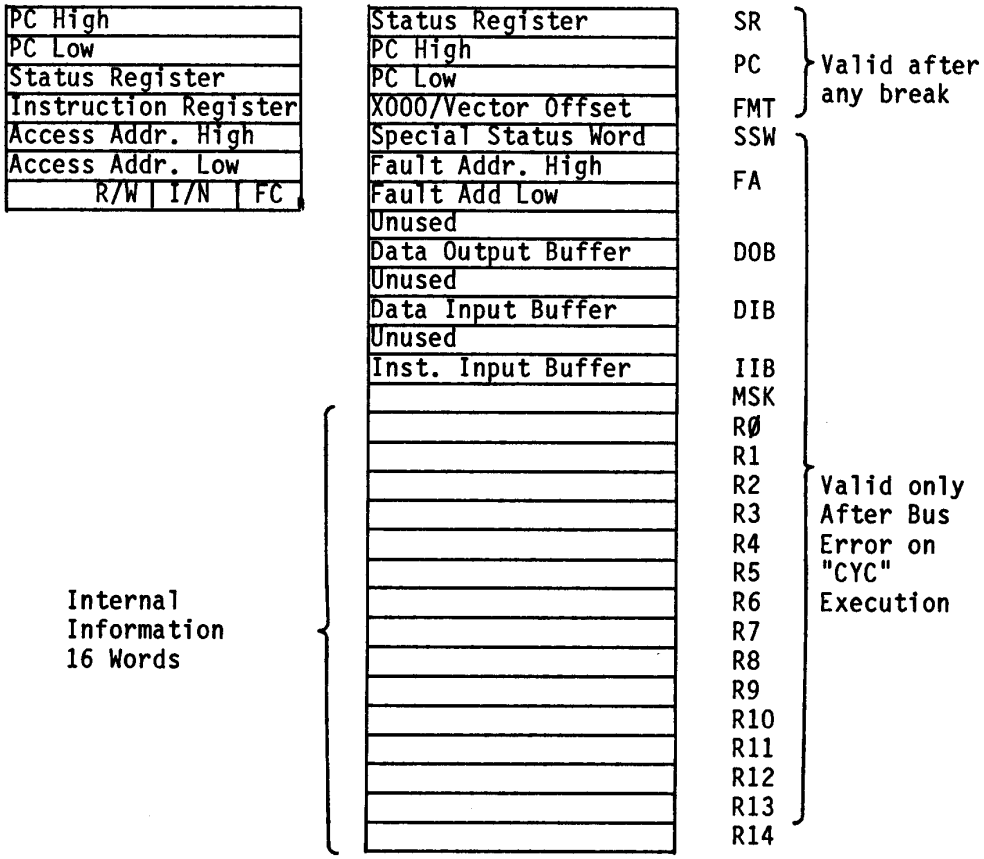
The emulator can send a bus error to the MPU for two different reasons:

- When executing the "CYC" command a bus error is forced in order that the long stack of information 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 "long" 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 an of these registers may result in unpredictable operation of the 68010 MPU.

Table 4-4.
Bus Errors



4.10 THE MEMORY DISASSEMBLER

The memory disassembler allows you to dump the contents of memory and have it displayed or printed in an easy-to-read format similar to a program listing.

NOTE:

You should be familiar with 68000 assembly language programming before reading this section. The information presented here is an overview, which will provide the necessary instructions when used in conjunction with Motorola documentation. You should have these handy:

- 16-BIT MICROPROCESSOR, User's Manual MC68000UM[AD3]
- MC68000 16-Bit Microprocessor Programming Card, MC68000(AC1)

4.10.1 Display Disassembled Memory DIS

This operator will cause memory to be disassembled and output to the controlling port (computer or terminal). The key sequence is shown in example 4.10-1. If no argument is specified, one page of disassembly is displayed, beginning at the last address when this operation was previously invoked.

Example 4.10-1 Use of DIS

- To disassemble one page of memory beginning at the last address when this operation was previously invoked:

  >DIS<return>

- To disassemble one page of memory beginning at the specified address:

  >DIS <single value><return>

- To disassemble a range of memory:

  >DIS <range><return>

- To continue disassembly one line at a time:

  ><space> (at the end of each line)

- To continue disassembly one page at a time:

  ><return> (at the end of each page)

4.11 THE LINE ASSEMBLER

The 68000 Line Assembler allows you to enter and assemble Motorola 68K mnemonic instructions into target memory. In addition to instructions, there are "Assembler Directives" to aid you in selecting memory addresses, using symbols, inserting numbers and text strings into memory, etc. The Line Assembler gives you a powerful software tool to facilitate in software patching, hardware/software debug, developing small programs, writing hardware/software test routines, etc.

4.11.1 Standard Mnemonics

All standard Motorola 68000 mnemonics are supported. These are listed in the Motorola 16-bit Microprocessor User's Manual.

NOTE:
Lines shown in bold print with a <return> are user entries; lines shown in regular print are the assembled response.

4.11.2 Assemble Line to Memory ASM

This operator will cause the line assembler to be invoked. The key sequence is shown in the following example. If no argument is specified, line assembly will begin at the last address, when this operation was previously invoked. To exit line assembly and return to the command level, enter either END or X with the address prompt displayed (as shown here).

Example 4.11-1 Use of ASM

- To start line assembly beginning at the last address when this operation was previously invoked:

```
     >ASM<return>
**** 680XX LINE ASSEMBLER ****

000000 >
```

- To start line assembly beginning at the specified address:

```
     >ASM $4C6A3<return>
**** 680XX LINE ASSEMBLER ****

     04C6A3>
```

- To terminate line assembly:

```
     012345 >X<return>
**** END OF LINE ASSEMBLY ****

     >
```

4.11.3 Assembler Directives

The following assembler directives are supported:

DIRECTIVE	DESCRIPTION
ORG	Set program origin
DC	Define bytes, words, longwords, and text strings
END	End line assembly
X	End line assembly
EQU	Define symbol value (only valid if symbolic debug hardware is installed)
SET	Define/Redefine symbol value (only valid with local labels §L0 to L9† unless symbolic debug hardware is installed)
L0,L1...L9	Print value of local symbol
'symbol	Print value of symbol (only valid with installed symbolic debug hardware)
<return>	Disassemble one instruction at current address
*	Current line assembly address

The key sequences for these assembler directives are shown in the following example.

Example 4.11-2 Use of Assembler Directives

- To set program assembly origin:

```
35F300 >ORG $4000<return>¥F0
004000>
```

- To define constant byte, word, longword, or text string:

```
002400 > DC.W $8034,256,1024<return>
002400 8034 0100 0400
002406>
```

- To exit line assembly:

```
0058FD >X<return>
**** END OF LINE ASSEMBLY ***

>
```

- To define symbol (if symbolic debug hardware is installed):

```
01FE00 >'Unit EQU $89400<return>
01FE00>
```

- To define/redefine local symbol or symbol (if symbolic debug hardware is installed):

```
018802 >L3 SET $032000<return>
018802>
```

- To print local symbol:

```
018802 >L3<return>
018802 >L3 SET $0003200
018802 >
```

- To print symbol (if symbolic debug hardware installed):

```
018890 >'Unit <return>
018890 >'Unit SET $0089400
018890 >
```

- To disassemble one instruction at current address:

```
0F5D0A ><return>
0F5D0A 33F4D8FE00450000  MOVE.W $FE(A4.A5.L),$450000
0F5D12 >
```

4.11.4 Usage Notes

+ _

- Plus or minus (+ or -) are the only arithmetic operators allowed in expressions

/ "

- Slashes (/) or double quotes (") are used to delimit ASCII strings. If you enclose the string in slashes, you may not use slashes within the string, but any number of double quotes may be used. If you enclose the string in double quotes, you may not use double quotes within the string, but any number of slashes may be used.

'<bs>

- Upper-case strings are the default. (If you have symbolic debug installed, the use of '<backspace> will allow entry of lower-case letters until you enter a <space>.)

$

- The two number bases are hexadecimal and decimal. Decimal is the default. Numbers beginning with $ are hexadecimal. All other numbers are decimal.

PCR

- When referencing memory, you must use a leading zero with the Program Counter Relative, as shown in these examples:

PCR with index and displacement
0(PC,D3)

Address register indirect with index and displacement
0(A4,D3)

#

- Remember that the pound sign has two different meanings:

--At the ES command level, # denotes a decimal number.
--Within the line assembler, # denotes an immediate addressing mode.

4.11.5 More Examples

The following examples represent ways in which the line assembler can be used.

mple 4.11-3
ing Addresses

```
>ASM 100<return>
   ****  680XX LINE ASSEMBLER   ****

000100 >MOVEQ #8,D3<return>
000100 7608                          MOVEQ #8,D3
000102 >MOVE (A2)+,(A6)+<return>
000102 3CDA                          MOVE (A2)+,(A6)+
000104 >DBNE D3,$102<return>
000104 56CBFFFC                      DBNE D3,$102
000108 >X<return>
****  END OF LINE ASSEMBLY  ****
```

Example 4.11-4
Using Local Symbols

```
>ASM 100<return>
****  680XX  LINE ASSEMBLER  ****
000100 >MOVEQ #8,D3<return>
000100 7608                          MOVEQ #8,D3
000102 >L5 MOVE (A2)+,(A6)+<return>
000102 3CDA                          L5 MOVE (A2)+,(A6)+
000104 >DBNE D3,L5<return>
000104 56CBFFFC                      DBNE D3,L5
000108 >X<return>
****  END OF LINE ASSEMBLY  ****
```

mple 4.11-5
Using Assembler
Directives

```
>ASM<return>
****  680XX LINE ASSEMBLER  ****
000108 >ORG *+$1000<return>
001108 >L2 SET $4<return>  == $00000004
001108 >L3 SET L5-*+2<return> == $FFFFEFFC
001108 >L1 DC.B "TEST",0<return>
001108 54 45 53 54  00
00110D >ORG $2000<return>
002000 >LEA L1(PC),A0<return>
002000 41FAF106                      LEA L1(PC),A0
002004 >MOVE.B (A0),-(A4)<return>
002004 1910                          MOVE.B (A0),-(A4)
002006 >ADDA #1,A0<return>
002006 D0FC0001                      ADDA #1,A0
00200A >BRA $2004<return>
00200A 60F8                          BRA $2004
00200C >X<return>
****  END OF LINE ASSEMBLY ****
```

Example 4.11-6
Error Message in
Response to ?

```
>ASM $4000<return>
****  680XX LINE ASSEMBLER  ****

004000 >'label BSR $2006<return>
          ?
>?
ERROR #9
ARGUMENT OUT OF RANGE
```

Example 4.11-7 Using Symbols With Symbolic Debug Hardware Installed

```
>ASM <return>
**** 68OXX Line Assembler  ****

004000 >BSR.L $2006<return>
004000 6100E004                    BSR.L $2006
004004 >'label SET $00004000<return>
004004 >'page_table SET $F01890<return>
004004 >'pt_blink SET $4<return>
004004 >MOVEA.L #'page_table ,A5<return>
004004 27C00F01890                 MOVEA.L #'page_table ,A5
00400A >LEA 'pt_blink (A5),A3<return>
00400A 47ED0004                    LEA 'pt_blink (A5),A3
00400E >X
****  END OF LINE ASSEMBLY  ****
```

SECTION 5
PROGRAMMING THE EVENT MONITOR SYSTEM

5.1 INTRODUCTION

The Event Monitor System is an expanded and enhanced breakpoint system. It is used to detect specific events occurring in the target system and to perform actions when these events are detected. Action is taken according to a set of statements. These statements combine detection comparators and action items. When an event is detected, any of the following actions may occur:

- all-cycle trace
- single-cycle trace
- window-mode trace
- external triggering
- pass counting
- breakpoints (ranging from simple to highly complex)

In addition, the Logic State Analyzer option gives you access to sixteen external logic signals that can be defined and considered in the Event Monitor System.

To set up the Event Monitor System, you must define event detectors that will trigger an action list. The event detectors and the action list are combined into WHEN/THEN statements, which become active when running the target system. WHEN/THEN statements take the following form:

WHE[N] <event> THE[N] <action>

Event detectors may be combined using AND, OR, and NOT. These are like logical ANDs, ORs, and NOTs, except that they are not on a bit level. A more complex example of a WHEN/THEN statement might look like this:

WHE[N] <event> AND <event> OR <event> THE[N] <action>, <action>, <action>

There are four event groups, each group consisting of eight comparators. The system can operate in only one group at a time. Each WHEN/THEN statement must be defined for a specific group. If no group is defined, the statement will default to group 1. WHEN/THEN statements are used to link event groups together for sequential operation.

The table on the next page summarizes the operators used with the Event Monitor System. (These operators are also displayed online on page 2 of the Help Menu.)

Table 5-1.
Event Monitor System Operators

	OPERATOR	NAME	BITS WIDE
SETTING AND CLEARING	CES	clear event system	
	DES	display event system	
EVENT COMPARATORS (singly or in combinations comprise event detectors)	AC1	address comparator 1	24*
	AC2	address comparator 2	24*
	DC1	data comparator 1	16**
	DC2	data comparator 2	16**
	S1	status comparator 1	16**
	S2	status comparator 2	16**
	LSA	Logic State Analyzer comparator	16**
	CL	count limit comparator	16
	*single address or address range **includes Don't Cares		
ACTIONS (What the Satellite Emulator does in response to the event detectors; several actions may be combined in a single statement)	CNT	count event	
	FSI	Force Special Interrupt	
	BRK	break emulation during RBK or RBV	
	TGR	trigger signal high for one bus cycle	
	TRC	trace event	
	RCT	reset count limit	
	GRO	switch event group	
	TOT	toggle tracing	
	TOC	toggle counting	
STATEMENT OPERATORS (used to combine event comparators and actions into statements)	IRA	internal range	
	XRA	external range	
	TO	to	
	LEN	length	
	WHEN	when	
	THEN	then	
	AND	and	
	OR	or	
	NOT	not	
	DC	Don't Care	
	SIA	Special Interrupt Address	

Remember that the Event Monitor System must be set up prior to its use. Comparator values can be stored in the EEPROM between emulation sessions. (Two users may store their event system setups.)

NOTE

When the Event Monitor System is used in conjunction with emulation, timing is not affected--the emulator still operates in real time.

5.2 DISPLAYING AND CLEARING THE EVENT MONITOR SYSTEM

DES
CES

Two operators are included for clearing the contents of the Event Monitor System and displaying its contents.

- To clear all the WHEN/THEN statements:
  >CES<return>

- To clear the WHEN/THEN statements for a single group:
  >CES <group number><return>

- To display all of the WHEN/THEN statements:
  >DES<return>

- To display the comparators as well as the WHEN/THEN statements for a given event group:
  >DES <group number><return>

5.3 EVENT COMPARATORS

There are eight event comparators for each of the four event system groups:

- two address comparators
- two data comparators
- two status comparators
- one count limit comparator
- one Logic State Analyzer comparator

Three types of data may be assigned to the comparators: integers, ranges, and don't cares. Values contained in any other registers in the system may be assigned to the comparators, as long as the data types are compatible, for example:

S1 = MMS

5.3.1 Address Comparators

AC1
AC2

The address comparators match addresses occurring within the emulation process against the 24-bit address bus. If a match is detected, the associated action occurs.

Address comparators can be a single address, an internal range, or an external range.

The examples shown here illustrate the format for assigning address comparators and ranges. When a single address is assigned to an address comparator, such as AC1 = $4766<return>, each time the address $4766 appears on the address bus, the AC1 comparator will detect this "event" and will produce a true output (the action associated with the AC1 event detector).

- To set an address comparator to a single address:
  >AC1.3 = $06FF<return>
  or
  >AC2.1 = $3488<return>
  or
  >AC1 = PC + $200<return>

  The assignment statement may include other operations, such as adding an offset to one of the CPU registers (in this case the program counter). This would cause the specified event to occur upon an access $200 bytes ahead of the current program counter.

IRA

- Ranges are set up with the IRA, XRA, TO, and LEN operators.

TO
LEN

To set an address comparator to an internal range (all addresses from n to m, including addresses n and m):

address comparator = IRA <address n> TO <address m><return>
>AC2 = IRA $3000 TO $3FFF<return>
or
address comparator = <address> LEN <length><return>
>AC2 = $3000 LEN $1000<return>
or
address comparator = <address n> TO <address m><return>
>AC2 = $3000 TO $47FF<return>

Note that when no prefix is applied (IRA or XRA) the range is assumed to be internal--IRA is implied.

XRA

- To set an address comparator to an external range (all addresses not between n and m -- addresses lower than and including n, or addresses higher than and including m):

address comparator = XRA <address n> TO <address m><return>
>AC1 = XRA $2000 TO $32FB <return>
or
>AC1 = XRA $2000 LEN $32FA <return>

(!)

- The inverse operator (!) can also be used:
>AC2 = !AC1

The above would define AC2 as the inverse of AC1. If AC1 is internal, AC2 would become its complementary external range and vice versa.

Both internal and external ranges include endpoints as part of the valid range. The LEN Operator provides an alternative to specifying ranges. When a range is specified with a LEN expression, the first value specified is the beginning address of the range and the last value is the block size (the length specified with LEN, minus one). Ranges can also be defined from other ranges with the inverse operator (!) shown in the first example.

NOTE
Addresses can also be assigned with the indirection operator (@). See Section 3.6 for an example.

5.3.2 Count Limit

CL

Each event has a count limit comparator, and the system has one hardware counter. When entering RUN mode, the value from CL.1 is automatically loaded into the hardware counter, and may be used in event system WHEN/THEN statements, as shown here:

```
S1 = RD + OVL
CL = #200
WHE[N] S1 THE[N] CNT
WHE[N] CL THE[N] BRK
```

In order to load the value from another CL register into the hardware counter, an RCT (reset count) action must be specified in

conjunction with the switch to a new group. This new count limit value may then be used in WHEN/THEN statements, as shown here:

```
AC1 = $7800
CL.2 = #10
AC1.2 = $7840
WHE[N] AC1 THE[N] RCT, GRO 2
2 WHE[N] AC1 THE[N] CNT
2 WHE[N] CL THE[N] BRK
```

5.3.3 Data Comparators

The data comparators are set like the address and LSA comparators. Data comparators may be assigned integer values and may contain Don't Care bits (see Section 5.3.5 for a detailed explanation of Don't Cares). Other registers, such as general purpose registers GD0-7 may be assigned to these comparators.

DC1
DC2

- To assign an integer
  DC1 = $F033

- To assign a Don't Care value
  DC2.3 = $FF00 DC $FF
  DC1 = GD0 (general purpose data register)

5.3.4 Status Comparators

S1
S2

The Satellite Emulator records a 16-bit status and control word in every Trace Memory cell. The bits in this word are a combination of 680XX-generated signals and signals internal to the emulator.

The emulator has a set of "constant" registers that the Event Monitor System can use as event comparators. When the status word matches the status defined by S1 and/or S2, the comparator output is true.

The following table lists the status constants. Example 5-1 shows how to set S1 and S2.

Table 5-2. Status Mnemonics

MNEMONIC	DESCRIPTION
BER	bus error
VM	valid memory address
VP	valid peripheral address
IP0-IP7	interrupt levels 0-7
SP	supervisor program
SD	supervisor data
UP	user program
UD	user data
CPU	CPU space
SC0-SC7	numeric names for all 8 space codes*
TAR	target system access
OVL	Overlay memory access
RD	read access
WR	write access
BYT	byte mode
WRD	word mode

* SCØ, SC3, SC4 used only by 6810.

Example 5-1.
Setting Status Comparators

- The general format is:

1	2		3		4		5		6		7
S1 =	TAR	+	RD	+	SC0	+	IP0	+	BYT	+	BER
S2	OVL		WR		UD/SC1		IP1		WRD		VM
					UP/SC2		IP2				VP
					SC3		IP3				
					SC4		IP4				
					SD/SC5		IP5				
					SP/SC6		IP6				
					CPU/SC7		IP7				

Note the seven-column format. A status comparator may be set with a maximum of one constant from each of columns 2-6; i.e., S1 and S2 can both have one from each of columns 2-6.

In addition, as many constants as desired can be included from column 7.

Remember that these are maximums. It is not necessary to use all the possible constants.

- Some sample status comparators are:

```
>S1 = TAR + RD + SD + IP3 + BER
>S2 = TAR + WR + CPU
>S2 = OVL + UD
```

The addition sign is used as a connective between the constant mnemonics, acting as a Boolean "AND."

Example 5-2
Examining the Contents of the Status Comparator

To examine the contents of the status comparator, type S1 or S2. Note, however, that when the status comparator name is keyed in, the system responds with a value, rather than the mnemonic code used to enter that value into the system. The table below is used to translate the system response back into the mnemonic codes entered originally. This table can also be used to help enter status comparator values directly.

Activated Bit Values			
	0=IP7	0=SC0	0=OVL + WR + WRD
	2=IP6	1=UD	1=WRD
	4=IP5	2=UP	2=RD
	6=IP4	3=SC3	3=RD + BYT
	8=IP3	4=SC4	4=TAR
	A=IP2	5=SD	5=TAR +BYT
	C=IP1	6=SP	6=TAR + RD
	E=IP0	7=CPU	7=TAR + RD + BYT

When you type S1 or S2, the system responds with a value with this general format:

$ 0000$X_4X_3X_2X_1$ DC 0000$X_4X_3X_2X_1$

The hexadecimal values X_1, X_2, X_3, X_4 represent the bit patterns of the status comparator register. Those to the left of the DC operator correspond to the activated bits (0s); those to the right are the Don't Cares, or mask values (1s).

Examination of the mask values reveals which bits have been activated, or enabled. A mask of FFFF shows that all the bits are masked, while a mask of FF8F indicates that all bits except 4, 5, and 6 have been masked. That is, 4, 5, and 6 are the only bits that have been enabled.

The activated bit values, to the left of the DC operator, correspond to the mnemonic entered into the comparator. These mnemonic codes can be read directly off the table once the enabled bit pattern has been determined. For example, suppose the system responds with:

a.

$ 00000060 DC 0000FF8F

The mask value shows that bits 4, 5, and 6 have been enabled. The 6 in the X_2 column to the left of the DC operator can be matched with the 6 in the same column of the table, indicating that the mnemonic entered was SP.

b.

$ 00000060 DC 0000FF8B

Mask values: all except 2, 4, 5, and 6 are masked

Mnemonic values: the 6 in X_2 corresponds to SP. The 0 in column X_1 corresponds to some combination of OVL, WR, and WRD; since only bit 2 is activated, though, the mnemonic entered must have been OVL.

Original entry: **S1 = SP + OVL**

c.

$ 00000054 DC 0000BF8B

Mask values: all bits except 2, 4, 5, 6, and 14 are masked.

Mnemonic values: since bit 14 is activated and a 0 shows in X4, VM must have been entered; the 5 in X_2 corresponds to SD, and the 4 in X_1 matches with TAR.

Original entry: **S1 = VM + SD + TAR**

d.

$ 00000607 DC 0000F1FD

Mask values: all except 1, 9, 10, and 11 are masked.

Mnemonic values: the 6 in X_3 corresponds to IP4 in the table; the 7 in X_1 indicates a combination of TAR, RD, and BYT, but only bit 1 is activated, so RD was entered.

Original entry: **S1 = IP4 + RD**

e.

$ 00000703 DC 0000F0FC

Mask values: bits 0, 1, 8, 9, 10 and 11 are enabled.

Mnemonic values: any odd value in X_3 indicates a combination of BER and one of the interrupt levels (IP0-IP7), simply because an odd value must include a 1 in bit 8. So the odd number corresponds to BER and the mnemonic for the next lowest even number, which is 6 in this case, so IP4 was entered. The 3 in X1, coupled with the mask value of C indicates that both RD and BYT were entered.

Original Entry: **S1 = IP4 + BER + RD + BYT**

This table also applies to the Memory Mode access status register, with certain restrictions. The MMS register allows the user to access only SD, SP, UD, UP, and CPU space. So, when the contents of the MMS register are displayed, only the X_2 column needs to be read. The format of the display is the same as that for the Status Comparator register.

The MMS register is not used in conjunction with the Event Monitor System.

5.3.5 Don't Cares
DC
X

The DC or X operators specify Don't Care bits. Bits specified to the left are significant while those to the right are ignored (Don't Cares). Where overlap occurs between significant and Don't Care bit positions, the bits are treated as Don't Cares.

Don't Cares are used with the event detectors when it is desirable to restrict monitoring to a subset of the sixteen data, status, and LSA lines; for example, you may wish to monitor only the low-order eight bits and ignore the high-order bits. This is done by specifying the high-order bits as Don't Cares.

Address comparators and count limit comparators may not contain Don't Cares.

- Don't Cares can be assigned in data, status, or LSA comparators. An example of setting a data comparator and including a Don't Care is:

>DC1 = $0055 DC $FF00<return>

The value of the Don't Care expression is assigned to DC1. The first value in the statement ($0055) is the match value. The comparator will be looking for this value on the data bus. The second value ($FF00) is the Don't Care mask. The comparator will mask all bit positions containing ones.

- Another method of entering Don't Cares and defining comparators uses Xs to mark the Don't Cares:

  >DC1 = $4XX2<return>

  The result of this assignment is $4FF2 as significant and $0FF0 as Don't Cares.

- A sample LSA comparator would be:

  >LSA = #65532 DC %10

  Note that the Don't Care value can be specified in different bases. The emulator looks at #65532 and translates it, then at %10 and translates it before dealing with the value as a whole.

5.4 EVENT MONITOR SYSTEM ACTIONS

The event detectors cause the Satellite Emulator to perform an action when they are detected during emulation. The trace function defaults to the ON state--tracing all bus cycles--unless TRC or TOT is specified.

BRK
TRC
CNT
TOT
TOC
RCT
TGR
GRO

The most commonly used detectors are BREAK, TRACE, and COUNT.

- BRK (Break) causes emulation to halt.
- TRC (Trace) traces the event; the Trace Memory is ON unless TRC or TOT is specified.
- CNT (Count) decrements the pass counter on the occurrence of a specified event. RCT resets this counter (see below).
- TOT (Toggle Trace) allows windowing. By identifying a starting event and ending event you can toggle the trace from ON to OFF or OFF to ON.
- TOC (Toggle Count) allows you to window the pass counter. By identifying a starting and ending event, you can toggle the counter from ON to OFF or from OFF to ON.
- RCT (Reset Counter) resets the pass counter to the specified count. To load the counter, see Section 5.3.2, Count Limit.
- TGR (trigger) causes the trigger output on the LSA Pod Assembly and BNC connector to be high for the next cycle. (LSA is discussed in detail in the next section. For BNC trigger information, see Section 5.6.2.)

- GRO (Group) causes the system to switch to another event group. (Event groups are discussed in more detail in Section 5.5)

The order in which actions are specified in a WHEN/THEN statement is not critical except in two cases:

- If CNT and RCT are both specified for the same event, the resulting action is RCT.

- If both CNT and TOC are specified for one event, the second action specified will be performed.

To perform both TRC and TOT or CNT and TOT functions, each function should be in a separate group.

Example 5-3.
Types of Breakpoints

Breakpoints range from very simple to highly complex. A simple breakpoint would be to break emulation when a particular address in the target program is accessed. For example, you could instruct the emulator to wait for the CPU to access a particular instruction in a program and to break emulation when the access occurs, as shown here. This type of breakpoint is useful for running the program until it reaches the code module or subroutine that you want to debug.

- To halt emulation when address = $3000

```
>WHE[N] AC1 THE[N] BRK
>AC1 = $3000
```

After setting this and using RBK (run with breakpoints), the program will execute until an access of any kind occurs at address $3000.

Each of the following examples adds a new feature or level of complexity to the WHEN/THEN statements shown here. Remember that the Event Monitor System can be used for many possible combinations of events and actions to suit your own needs. These examples illustrate only a small percentage of the system's capabilities. (Additional examples are in other sections of this chapter.)

- To halt at a code module with multiple entry points

```
>WHE[N] AC1 THE[N] BRK
>AC1 = $3000 TO $32FC
```

The same WHEN/THEN statement is used, but AC1 is now defined as a range

- To save only the bus cycles you want to view

```
>WHE[N] AC1 THE[N] TRC
>AC1 $3000 TO $32FC
```

In this case, you do not have to specify a breakpoint; only the bus cycles occurring in the range AC1 will be traced.

- To use the pass counter

```
>WHE[N] AC1 THE[N] TRC, CNT
>WHE[N] CL THE[N] BRK
>AC1 = $3000 TO $32FC
>CL = $000A
```

In this example, each bus cycle occurring in the address range $3000 to $32FC will be stored in trace memory and cause the pass counter to be decremented. When ten cycles ($0A_{16}$) have occurred, emulation will be broken.

- To stop program execution when a specific data pattern is written to memory at a certain address

```
>WHE[N] AC1 AND DC1 AND S1 THE[N] BRK
>AC1 = $3000              (address comparator)
>DC1 = $55AA              (data comparator)
>S1 = WR                  (status comparator set to write)
```

When conditions AC1, DC1 and S1 are met simultaneously, the emulator will break.

- To stop program execution when one of two data patterns appears at either of two addresses during a write cycle

```
>WHE[N] AC1 OR AC2 AND DC1 OR DC2 AND S1 THE[N] BRK
```

- To set two conditions for a breakpoint

```
>WHE[N] AC1 AND DC1 AND S1 THE[N] BRK
>WHE[N] AC2 AND DC2 AND S1 THE[N] BRK
```

5.4.1 Force Special Interrupt FSI SIA

The Force Special Interrupt feature allows your program to jump to any address (for instance, a particular subroutine) when a specified event is detected. This address is set by assigning a value to Special Interrupt Address (SIA).

The user program is interrupted by the Event Monitor System when the FSI event is detected, and program execution will begin at the SIA. The routine must terminate with a "return from exception" (RTE) instruction to properly return to the interrupted routine. The message FSI ACTIVE will be printed when an FSI occurs.

The FSI feature is helpful for inserting a quick patch in ROM code or to terminate a process requiring a careful termination routine. Only one SIA address can be set at a time.

The keystroke sequence for setting and clearing the Force Special Interrupt is shown in the next example. The address argument is the address of the interrupt service routine.

- To set the Force Special Interrupt address:
  >SIA = <address><return>

- To force a special interrupt

  (A sample use for this would be to insert a code module that you did not include in a linked program that is already compiled and loaded for debugging.)

```
>WHEN AC1 THEN FSI
>WHEN AC2 THEN FSI
>SIA = $F2D0
>AC1 = $302C
>AC2 = $4010
```

In this example, the program will execute normally until address $302C or $4010 is reached. When one of these addresses occurs, emulation will be halted. Address $302C (or $4012) will be pushed onto the CPU stack as the return address. The program counter will be set to the value specified in SIA ($F2D0), and the CPU will begin executing the program at the new address. To return to the original program at the end of the patch, execute a Return from Execution" (RTE) instruction (this will vary from processor to processor) and the CPU will unstack the pushed program counter ($302C or $4010 in this example) and continue running from where it left off.

EVENT GROUPS
GRO

The Satellite Emulator is capable of having up to four groups of event detectors defined at one time, analogous to "event states." This is done by adding the suffix .n (n = 1 through 4) to the event comparators; for example, AC1 can be AC1.1, AC1.2, DC2 can be DC2.4, etc. These groups are defined when you are assigning the event comparators or can be defined in the WHEN/THEN Statement. This is illustrated in the following example.

Within one WHEN/THEN statement, only one group of events can be dealt with at one time--the system can only be in one state at a time. The group operator (GRO) is used to switch the Satellite Emulator to a different event group in response to the event detector. If no group number is assigned, the system will default to group 1.

- To use more than one event group

```
>WHE[N] AC1 AND S1 THE[N] CNT
>WHE[N] CL THE[N] RCT, GROUP 2
>2 WHE[N] AC1 AND S1 THE[N] TRC, CNT
>2 WHE[N] CL THE[N] BRK
>AC1.1 = $4010; AC1.2 = $4011
>CL.1 = 3; CL.2 = $14 (20 decimal)
>S1.1 = WR; S1.2 = RD
```

This example could be used to monitor the activity of an I/O port after the port had been initialized. When AC1 has been accessed by three write cycles, the counter will be reset and the event monitor will transfer to group 2.

Example 5-4.
Sample Valid WHEN/
THEN Statements

- Possible Event Groups:*

Group 1 =AC1.1, AC2.1, DC1.1, DC2.1, S1.1, S2.1, LSA.1, CL.1
Group 2 =AC1.2, AC2.2, DC1.2, DC2.2, S1.2, S2.2, LSA.2, CL.2
Group 3 =AC1.3, AC2.3, DC1.3, DC2.3, S1.3, S2.3, LSA.3, CL.3
Group 4 =AC1.4, AC2.4, DC1.4, DC2.4, S1.4, S2.4, LSA.4, CL.4

* Any valid combination of comparators can be used but all must be from the same event group. Event groups are signified by adding .n (1, 2, 3, or 4) to the comparator as above.

- Simple WHEN/THEN Statement (no event group specified, defaults to group 1):

WHE[N] **<event>** **THE[N] <action>**
any comparator or valid combination formed with AND, OR, or NOT; must be from same event group

- Event Group WHEN/THEN Statement:

X WHE[N] <event> THE[N] <action>
X = 1, 2, 3, or 4

- Event comparators are assumed to be AC1.2 and AC2.2

>2 WHE[N] AC1 OR AC2 THE[N] BRK, TGR<return>

- WHEN/THEN clause assumed to be from group 3.

>WHE[N] NOT AC1.3 AND NOT DC1.3 THE[N] BRK, RCT, TRC<return>

- System defaults to group 1 when no group is specified.

>WHE[N] DC1 and AC1 THE[N] FS1<return>

- Group 4 comparators.

>4 WHE[N] AC2 THE[N] CNT<return>

5.6 OPTIONAL LOGIC STATE ANALYZER LSA

The Logic State Analyzer option assembly consists of a pod, cable, and probe clips. It provides you with access to external logic signals that can be fed directly into the trace and break card of the emulator. This data is qualified and clocked with other trace data by the Event Monitor System. (Trace Data is displayed by using the DRT command.)

5.6.1 LSA Functions

The Logic State Analyzer is used for many applications, such as:

- debugging data and address lines on the other side of the CPU buffer
- debugging decode lines
- keeping track of memory management
- debugging I/O
- address and chip select decoding

The LSA comparator is assigned with an assignment statement, just as the address comparators are. It is 16-bits wide; Don't Care bits are permissible.

- To monitor a specific activity outside the microprocessor

  This example will turn on a trace when that activity occurs and turn off the trace when the activity is terminated. The two event groups are required to specify separate on and off points.

```
>WHEN LSA THEN TOT, GRO[UP]2
>2 WHEN LSA THEN BRK
>LSA.1 = $0000 DC $FFFE
>LSA.2 = $0001 DC $FFFE
```

  This example waits for the logic state analyzer, Bit 0 to go low and then uses the toggle trace command (TOT) to turn on trace memory, and GRO[UP]2 to switch groups. In group 2 all bus cycles are traced until LSA pod bit 0 goes high. Then emulation is broken.

5.6.2 Timing Strobe

The ES uses a bus request signal (shown in Figure 5-1) to generate a trigger which is sent to the LSA pod and to the BNC connector on the rear panel. The trigger is a low-going-high signal for approximately one bus cycle and is generated after an event.

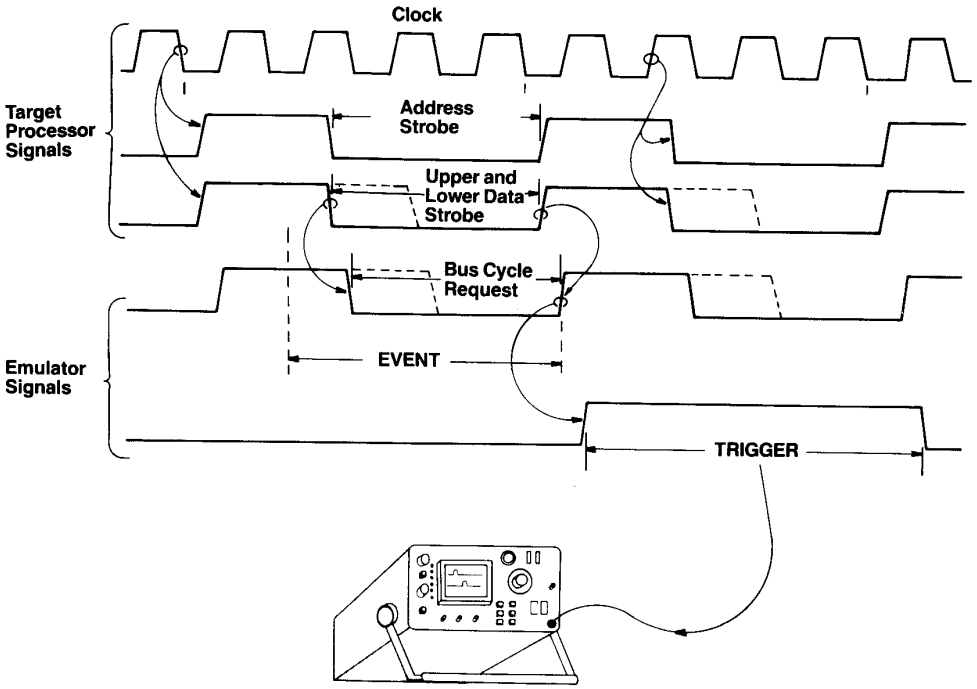
If you do not have the LSA pod, you can still take advantage of the trigger through the BNC connector for use with signature analysis equipment, a logic analyzer, or an oscilloscope:

- With an oscilloscope, the trigger could be used to flag a loop or I/O routine.
- With a signature analyzer, you can use the trigger to provide start and stop pulses, from the LSA pod or the BNC connector.

Another use for the trigger would be to connect two emulators, using the signal from the first to trigger a break in the second emulator. The Event Monitor System would be programmed as shown:

- Emulator 1: **WHE[N] <event> THE[N] TGR**
- Emulator 2: **WHE[N] LSA THE[N] BRK**

Figure 5-1
LSA Timing
iagram



5.7 STATEMENT CONTROL

5.7.1 Repeat Command

An asterisk (*) at the beginning of a command line repeats one or more commands. 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. A command having normal ESL syntax succeeds this argument.

For example:

```
>*5STP;DT
>* 5STP;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, it cannot be in hex, nor can it be a variable; and there must be a space following the repeat argument if the next character is a decimal digit.

Indefinite Repeat

When the repeat argument is not specified, it is assumed to be 4,294,967,295 ($2^{32}-1$). There are two ways to stop an indefinite repeat.

First, you can abort a repeat by executing a reset (usually a CNTL Z). However, note that this will also abort emulation if it is in progress, without saving the state of the CPU.

Second, there is a variable called "TST" that gets set to all 1's at the beginning of a repeat. Then it is tested for zero just before a line is re-executed. If TST becomes zero, the buffer is not executed and the repeat halts, returning control to the users terminal.

If you want to single step and disassemble until you reach a particular address, you could type, for example:

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

In this example, single stepping continues until the program counter equals OC324 Hex. If the PC does not reach OC234, you can still use CNTL Z to stop the repeat.

5.7.2 Loop Counter

When a repeat is initialized, just before execution begins, the repeat argument goes to an ESL variable called "LIM;" if "IDX" is greater than or equal to "LIM" the repeat is stopped. Since "LIM" and "IDX" are ESL variables, they may be used in commands or modified by the execution of the repeat.

Here are three examples:

```
>BASE DX=#10
>*3 IDX
#0
#1
```

#2

```
>MM.B $1000
$001000 $34          >*4 LIM-IDX-1
$001001 $C0
$001002 $BF
$001003 $00

$001004 $21          >MM MMP-4
$001000 $03          >*4
$001001 $02
$001002 $01
$001003 $00
$001004 $21          >
```

In the first example, "IDX" is printed showing that it is reset to zero and incremented thereafter. The second example shows how a block of memory can be initialized to a decrementing count ending in zero. In the last repeat example, the initialized block of memory is displayed.

If "IDX" is modified during a command repeat loop, it will still be incremented before being compared to "LIM." This may cause the loop to be exited one cycle earlier than expected.

5.7.3 Macros Defining Macros

You can define up to ten macros. They are referred to by decimal numbers Ø-9. The ten macros are linked in one buffer with #1 beginning first then #2...#9 with #Ø being last.

If the sum of the lengths of all 1Ø macros is greater than the buffer length, then the highest numbered macros will be truncated in order to fit them into the buffer size, starting with macro #Ø. This truncation happens silently, without any indication to the user.

Here is an example of some macro definitions:

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

The syntax is as follows: the first character on the line must be the underscore; the second character must be a decimal digit, a comma, or a period; the third character on the line must be an equals. If the syntax varies from that listed above, the line will be passed to the parser, which will throw it out as illegal syntax. If the syntax is correct, the remainder of the line after the equals and up to, but not including the return, will replace the previous definition of the macro. No syntax checking is done when a macro is defined, syntax errors will only be detected when the macro is executed.

In the above example, macro #3 contains two nested macros. The macros are not expanded when the macro is defined, but only when it is executed, so the definition of macro #3 may change depending on the content of macros #1 and #2.

If you define a macro, but only type a <return>, the macro is defined as null. A null macro is not displayed by the MAC command and when it is executed, no characters replace the macro call

argument.

Filling The Buffer

If macros #1 to #8 are defined, and in this process used up all of the space in the buffer, then an attempt to define macro #9 or #Ø would result in those macros remaining null. Also, if the length of any macro from #1 to #7 was increased after filing the buffer, then macro #8 would be truncated as a result and if the increase was more than the size of macro #8, then macro #8 would become null and macro #7 would be truncated, and so on. There are no warnings when truncation or nullification takes place, so if a number of long macros are defined, the "MAC" command should be executed to determine if the macros with the highest numbers are still intact.

Displaying Macros

The MAC command will display all of the macros that contain 1 or more characters. Nested macros are not expanded by MAC. The macros are displayed the way you typed them in and they are identified by the same three-character sequences that are used to define the macros.

This is an example of macro definition:

```
>_5='This 'is 'a 'macro
>_6=ABCDEFG
>_1=PC;RET;STP;DT
>_2=PC=$1ØØØ
>_3=MM.B $2ØØØ+GDØ
>_4=@(SSP+4)
>_6=
```

This is an example of macro display:

```
>MAC
_1=PC;RET;STP;DT
_2=PC=$1ØØØ
_3=MM.B $2ØØØ+GDØ
_4=@(SSP+4)
_5='This 'is 'a 'macro
```

Executing Macros

You can execute macros #1 and #2 by a single keystroke when not in memory mode. Whenever you type a comma as the first character on a line, macro #1 is executed; if you type a period as the first character on a line, macro #2 is executed. You an execute any of the ten macros by entering the underscore followed by a decimal number.

A macro may contain a portion of a command, or an entire command. It cannot contain part of a token, i.e., the "AØ" register cannot be specified by taking the last character of one macro ("A") and concatenating it with the first character of the next macro ("Ø"). If several macros each contain a single command, and it is desired to execute them serially as a string of commands, then the semicolon can be used to separate the macro calls.

For example:

```
>_1;_2;_3
```

The semicolon can also be used within the macro at the beginning

or end to separate commands.

Since a macro may contain a portion of a command, you could do something like the following example:

Macro definition:

```
> 4=GD1
> 5==$24
> 6==@4
```

Macro execution:

```
> 4 6          >GD1=@4
> 4 5          >GD1=$24
```

The right side shows how the macro is expanded when executed, the contents of the two macros are concatenated to form a complete command.

SECTION 6
INTERFACING AND COMMUNICATIONS

6.1 INTRODUCTION

This section gives information necessary for interfacing and communication between the Satellite Emulator and other units in an emulation system. Information includes:

- serial data requirements
- setting system control
- data transfer and verification

Specifications for the serial data formats are located in Appendix A.

6.2 SERIAL DATA REQUIREMENTS

The Satellite Emulator is compatible with RS232C standard pin conventions and signaling levels (given in section 2.3.2).

The standard software transmits and receives ASCII characters requiring seven-bit representation. One stop bit is recommended for most uses; however, some data terminals require two for proper operation.

The format of a serial word is shown in Figure 6-1. When no data is being transmitted, the Serial Data Out pin will be at the 12 volt level (marking). When the Satellite Emulator sends a character, there will always be a start bit, followed by 7 or 8 data bits, and 1 or 2 stop bits. The number of data bits and stop bits are specified by command, described in the operation section for the microprocessor you are using.

The Satellite Emulator sends and checks parity according to system set-up parameters.

Two additional signals used by the emulator are the Request to Send (pin 4) output and the Clear to Send (pin 5) input. The software uses these signals to coordinate data transfer. When the emulator is ready to begin receiving data, it changes the Request to Send line from low to high and awaits data transmission. When it has finished receiving data, it returns the Request to Send line to the low state. When the emulator is ready to send a character, the software tests the condition of the Clear to Send line. When used in conjunction with XON and XOFF, transmission of the character is provided only if Clear to Send is in the high state; the character is held if the signal is in the low state. Thus, a receiving unit may control the transfer of data by taking the Clear to Send line high when more data is desired and low when not ready for data. The ASCII control characters, XON and XOFF, are recognized by the emulator.

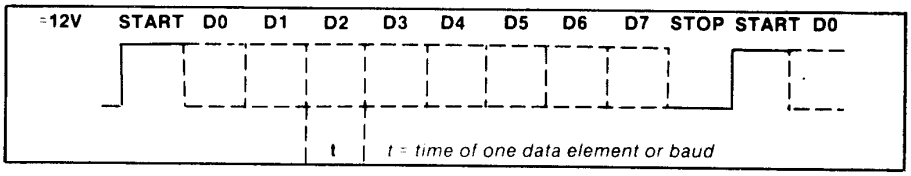


Figure 6-1
Format of a Serial Word

6.3 SETTING SYSTEM CONTROL

The Satellite Emulator can operate under CRT terminal or host computer control, or can become transparent, allowing the CRT terminal to communicate directly with a host computer.

6.3.1 Terminal Control TCT

This operator is entered from a host system interfaced through the computer port, only when the Satellite Emulator is in the host system control mode. Control is transferred to the CRT terminal and away from the host system. This overrides the setting of the interface parameter switch.

6.3.2 Computer Control CCT

This operator, analogous to the Terminal Control operator, is entered from a CRT terminal interfaced through the terminal port, only when the emulator is being controlled via a CRT terminal to a host system interfaced through the computer port. This overrides the setting of the interface parameter switch. CCT is used with TRA for automated tests.

There are four characteristics to remember about CCT:

- First, the emulator will echo most of the characteristics sent to it, so the computer can use this feature to check the data transmission.
- Second, when the host sends a RETURN, the emulator begins processing the command line. New lines generally begin with RETURN LINEFEED NULL NULL.
- Third, the host must be able to handle incoming data at high rates as the emulator will be sending at 960 characters/second (9600 baud); the host should be able to send XON/XOFF to the emulator.
- Fourth, UPL (upload) and DNL (download) expect data from the same port whether you are using TCT or CCT: if you are downloading the emulator **always** expects data to come from the host, and if you are uploading data is **always** sent to the host.

NOTE:

If you execute CCT in error, turn the emulator off, then on again.

6.3.3 Transparent Mode TRA

This operator instructs the emulator to become transparent, allowing the CRT terminal interfaced through the terminal port to communicate directly with a host system interfaced through the

computer port. TRA can be entered while in either Terminal or Computer Control modes.

Example 6-1. Terminal Control, Computer Control, Transparent Mode

The Terminal Control and Computer Control operators are used to switch control back and forth between a host system and CRT terminal. The Transparent operator allows you to bypass the Satellite Emulator and communicate directly between your CRT terminal and host system. The emulator acts only as an interface. In this mode the emulator can buffer up to 64 characters for each port and can operate the ports at independent baud rates.

Refer to the system configuration shown in Figure 6-2. The initial physical connection is made according to the procedures outlined in Section 2. Original control is set with the interface parameter switch.

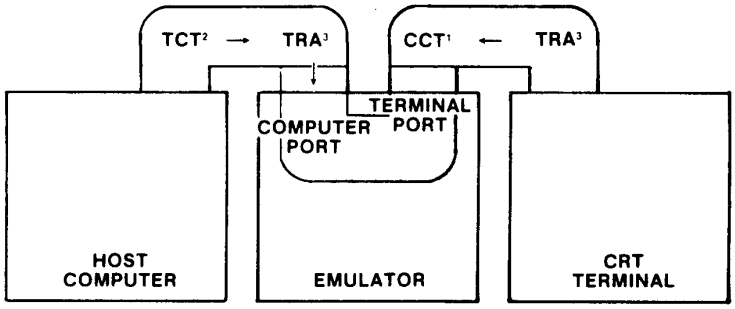
With the set-up complete, start with the CRT terminal controlling the Satellite Emulator. This is done by having the interface parameter switch in any position but those that are for computer control. (Positions 3 and 4 are for computer control.)

- If you now want to switch control to the host system, despite the fact that the switch is positioned for terminal control, you will enter:
  >CCT<return>

- To go back to terminal control, enter at the computer port:
  >TCT<return>

- If you want to communicate directly between the CRT terminal and the host system, enter from the controlling port:
  >TRA<return>

Figure 6-2.
System Control



1. *CCT switches control to computer.*
2. *TCT switches control to CRT terminal.*
3. *TRA (entered from controlling port) allows communication directly between CRT terminal and computer.*

6.4 DATA TRANSFER AND MANIPULATION

These commands are used for moving data in and out of the emulator and manipulating data within. Formats are described in Appendix A. The commands are:

- Upload - UPL
- Download - DNL
- Verify or compare - VFY

6.4.1 Upload and Download UPL DNL

Upload and download operations initiate routines to load the target system memory and/or Overlay Memory with data from a host system, and to dump data from the target system address space to a host system and/or Overlay Memory.

The Satellite Emulator will download in either a software handshake or no-handshake mode. While the no-handshake mode is faster and very simple, the handshake mode offers verification that data is received correctly and allows resending of data that is received bad.

In an Upload operation, data is transferred from the emulator to a host computer or other peripheral interfaced to the Satellite Emulator computer port. A Download, conversely, moves data from a host computer to the emulator.

The following steps are necessary to Upload data from the emulator to a host computer or other peripheral.

- Type TRA <return>. This puts the ES in the transparency mode for entering a command line to the host computer and prepares it to receive a file. Note that this TRA command is not necessary when using and uploading to a hard copy printer. Do not enter a <return> at the end of the line. Instead, type in the two-character transparency escape code. This code returns the emulator to its normal communication status with the user and causes the emulator to send the host computer a "command line terminator." The host computer (or other peripheral) should now be ready to accept data.

- Enter the Upload command for the desired range (see Example 6-2. Upload and Download). The emulator will now dump data to the computer port in the download format specified in the Set Menu. Refer to Table 3-5.

Typing the DNL command at the terminal causes two things to happen. First the emulator readies itself to receive data; then it goes into a "transparent" mode (like the TRA operator described previously, though accomplished here without the TRA command), allowing the CRT terminal to communicate directly with the host computer. This is necessary to allow you to enter the command line necessary to tell the host computer to send data to the emulator. Do not enter a <return> at the end of this line. Instead, type in the two-character transparency escape code. As data records are received, they are displayed on the CRT terminal. The command line terminator, transparency escape code, and the serial data format are user-defined with the Set-Up command, described in Section 3.

If the DNL operator is issued by the host computer (in computer control mode), the process is somewhat different. The emulator will not go into transparent mode and data records will not be displayed on the terminal. However, after successful reception and storage of every data record, the emulator will respond to the host with an ASCII ACK (6) character. Thus, to monitor the download process, the host should send one record, then wait for a response. If the response is the ACK, the host should continue with the next data record.

If the response is not the ACK, the emulator will have detected an error or End of File condition. In the case of the EOF, the emulator will return the normal prompt because the data transfer is complete.

If the Satellite Emulator has detected an error, it will respond with a <return>, line feed, several spaces, a ?, and a BEL. Then it will revert to the normal prompt on a new line. The host can then find the cause of the error by sending a ? to the Satellite Emulator.

There are only two errors that can occur during a download. There may be a checksum error in the data record. In this case, the DNL process is aborted before any data from that record is stored to memory. The second type of error is a read-after-write verify error. Every byte in a data record is verified after it is stored. If an error occurs, the DNL process is aborted but some of the data in the record has obviously been stored to memory.

- Upload:
  >UPL<range argument><return>

- Download:
  >DNL<return>
  ><transparent><commands to computer><escape code>
  This terminates the Transparent mode and the Download occurs. Note that the escape code is created with the Set-Up command described in Section 2.

6.4.2 Verify VFY

The Verify operator (VFY) compares data received at the computer port with memory. Any differences are displayed. The operator interaction required is similar to the Download command. The VFY command does not display incoming data records. See the following example.

- The format for Verify operator is:
  >VFY<return>

- Any differences will be displayed as:
  address = XX NOT YY

In the above example, the address is the address where the misverify occurred. XX denotes the actual data present at that location. YY denotes what should be at that location.

SECTION 7
DIAGNOSTIC FUNCTIONS

7.1 INTRODUCTION

The Special Functions are a group of utility routines and diagnostic tests invoked with the SF operator. These routines are used for debugging hardware or accommodating unusual hardware conditions.

NOTE:

If the default base in your machine is hexadecimal, you can omit entry of the $.

7.2 RAM DIAGNOSTICS

These routines are "canned" RAM tests that can be run on the target or RAM Overlay system. The tests are executed in word mode.

7.2.1 SF $0, <RANGE>

This routine involves three steps. First, the RAM test consists of writing a zero to the test cell and then reading the cell to see if a zero exists. Next, a one is used for the test pattern followed by $3, $7, $F, $1F, $3F,..., $FFFF, $FFFE, $FFFC,..., $C000, $8000. Finally if a failure is detected, the problem address, correct data and faulty data are displayed. This routine can be aborted by resetting the emulator but will finish after a single pass.

7.2.2 SF $1, <RANGE>

Executes a complete RAM test over the words within the specified range. The test was derived from a study by Nair, Thatte and Abraham entitled "Efficient Algorithms for Testing Semiconductor Random-Access Memories: (IEEE Transactions on Computers, vol. c-27, no. 6 June 1978). The test corresponds to their algorithm "A" and is more efficient than standard "GALPAT" type tests. If an error is detected by this test, the associated address, good data, bad data, and test sequence number are displayed. The sequence number corresponds exactly to the sequence numbers suggested in the article, but if you do not have the article, the above information should be sufficient. This is a single-pass test that can be aborted by resetting the emulator.

7.2.3 SF $2, <RANGE>

Continuously executes the test described for "SF $0" above. While executing the test, a pass count is maintained and displayed on the screen. The count is updated by rewriting the display line without using a "line feed." Thus, intermittent errors will not be pushed off the screen by the pass count. You must reset the emulator to terminate this test.

7.2.4 SF $3, <RANGE>

Executes the RAM test described in "SF $1" above, continuously over the words in <RANGE>. A pass counter is displayed as in "SF $2." You must reset the emulator to terminate this test.

7.3 SCOPE LOOPS

Scope loops are diagnostic routines built into emulator firmware. The uses for these special functions range from locating stuck address, data, status or control lines to generating signatures using common signature analysis equipment.

The routines for Scope Loops are executed at maximum speed. This short cycle time allows the hardware engineer to easily view the timing of pertinent signals in the target system without using a storage type oscilloscope. All of these routines must be terminated by resetting the emulator. The scope loops can be executed in byte or word data lengths. In addition to byte and word data, SF $16 may use long word data. The data length will be the global default (see SZ) or can be specifically set by using the dot operator with the command i.e. "SF.W $10, $4149F3."

If the length specified is invalid for the routine, the longest possible length will be used. As with the RAM tests, these scope loops access the memory space defined by the last setting of MMS (memory mode status). The following paragraphs describe each of the different scope loops.

7.3.1 SF #10, <ADDR> Executes "reads" into the target system. (Peeks)

7.3.2 SF #11, <ADDR>, <DATA> Executes "writes" into the target system. (Pokes)

7.3.3 SF #12, <ADDR>, <PAT-1>, <PAT-2> Writes alternating patterns to the target system.

7.3.4 SF #13, <ADDR>, <PAT> Writes the pattern to the target system but the pattern is rotated one bit left after each "write."

7.3.5 SF #14, <ADDR>, <DATA> Writes the supplied data to the target system and then reads it from the target system at the same address. Data read from the target system is ignored by the ES.

7.3.6 SF #15 <RANGE> Executes a read from the target system for every address in <RANGE>.

7.3.7 SF #16 <ADDR> Writes an incrementing count value into the target system.

7.3.8 SF #17 Generates RESET pulses into the target system.

, CLOCK AND CRC
CLK
CRC

The CLK and CRC operators are useful during diagnostic testing.

- CLK reads the target system clock and returns the value in KHz, accurate to 1 to 2 KHz.
  >CLK<return>

- CRC computes a cyclic redundancy check over an address range. It will be useful for checking if a block of memory has changed.
  >CRC <address range><return>

***NOTE**

For Diagnostic purposes CRC will function for byte, word or long word.

7.5 Bus
BUS

BUS displays the status of several status lines. For example:

- >BUS

	FAULT STATUS		
HLT	IPL	RST	VCC
0	0	0	0

In this example, "0" indicates a no-fault condition; a fault condition would be indicated by "1."

Table 7-1.
Special Functions

TYPE TEST	BYTE, WORD OR LONG WORD	KEY SEQUENCE	DESCRIPTION
RAM Diagnostics	word	SF $0,<range><return>	Simple RAM test, single pass
	word	SF $1,<range><return>	Complete RAM test, single pass
	word	SF $2,<range><return>	Simple RAM test, looping
	word	SF $3,<range><return>	Complete RAM test, looping
Scope Loops	byte or word	SF $10,<address><return>	READ
	byte or word	SF $11,<address><data> <return>	WRITE
	byte or word	SF $12,<address>, <pattern 1>, <pattern 2><return>	WRITE alternate patterns
	byte or word	SF $13,<address>, <pattern><return>	WRITE pattern then rotate
	byte or word	SF $14,<address>,<data> <return>	WRITE data then READ
	byte or word	SF $15,<range><return>	READ data over entire range
	byte, word, or long word	SF $16,<address><return>	WRITE incrementing count
		SF $17<return>	Generate RESET pulses

7.6 COM AND DIACOM

COM allows you to communicate directly with a program running in the target system. COM allows the simulation of communication between the target system program and a serial interface (usually the ES control terminal, sometimes another computer attached to the ES.

The routine is invoked with a simple 32-bit argument, the address of a 2-byte "port" in target memory.

- The first byte is for characters coming from the target program. The MSB of this byte is used to indicate "new character" to the ES. If the bit is set, this routine will read the character, display it on the controlling port, and clear the target memory location (as a handshake).

- The second byte is the write byte. If a character arrives from the controlling port, the routine will place it in the target memory with the MSB set. The routine will terminate only when the terminal transparent mode escape sequence is detected. The routine does not check to see if the last character written to the target system was accepted.

As examining target memory requires that emulation be halted for about 180 microseconds, COM will wait 1/2 second between target system reads. However, if a character is placed in the output port byte by the target system program, COM will collect the character, reset the MSB and re-examine the port as soon as the character has been put into the output UART buffer. COM will also immediately examine the output port whenever it places a character in the input port.

DIA

DIA allows you to display, on your controlling device, a string of characters that are stored in target memory. This routine is invoked with a simple 32-bit argument.

- The 8 MSB's of the argument contain the expected stop characters.
- The lower 24 bits contain the address of the first character of the message.

DIA begins with a RETURN on a line feed. The routine then reads one byte at a time from your target system, starting with the address you specified and working towards high memory. The characters are displayed on the controlling port (usually the ES controlling CRT).

The DIA routine is completed when the character read matches the stop character.

7 EXECUTING CUSTOM DIAGNOSTICS

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 GDO-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 Long-Branch must be present or the Diagnostic will not be executed. The Long-Branch will vector to 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" 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.

Table 7-1
Custom Diagnostics
Access Codes

Number	Description
SF $40	Execute user-written diagnostic that begins with a long branch at 7000
SF $41	Execute user-written diagnostic that begins with a long branch at 7004
SF $42	Execute user-written diagnostic that begins with a long branch at 7008
SF $43	Execute user-written diagnostic that begins with a long branch at 700C
SF $44	Execute user-written diagnostic that begins with a long branch at 7010
SF $45	Execute user-written diagnostic that begins with a long branch at 7014
SF $46	Execute user-written diagnostic that begins with a long branch at 7018
SF $47	Execute user-written diagnostic that begins with a long branch at 701C
SF $48	Execute user-written diagnostic that begins with a long branch at 7020
SF $49	Execute user-written diagnostic that begins with a long branch at 7024

7.7.1 Peeking and Poking to the Target System (68010)

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. 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 680XX's 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 the overlay exclusively (OVO) or the target exclusively (TGO). The other two codes access the target and also the overlay, 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 the 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).

Table 7-2
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 when it is mapped.

7.7.2 Peeking and Poking to the Target System (68000/68008)

Unlike the 68010, the 68000 and 68008 processors do not support the "MOVES" instruction. Consequently, the 68000/68008 emulators access the target differently.

When the 68000/68008 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 the Overlay, if it is 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.

7	6	5	4	3	2	1	0
n/c	n/c	n/c	TGO	OVO	X2	X1	X0

Table 7-3
68000/68008 Function Code Substitution Register

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.

7.7.3 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:

Table 7-4
Memory Allocation for the GR0-7 & GD0-7 Registers

$3000 - $3003	Beginning Range - GR0
$3004 - $3007	Ending Range - GR0
$3008 - $3008	Beginning Range - GR1
$300C - $300F	Ending Range - GR1
$3010 - $3013	Beginning Range - GR2
$3014 - $3017	Ending Range - GR2

```
$3018 - $3018                    Beginning Range - GR3
$301C - $301F                       Ending Range - GR3

$3020 - $3023                    Beginning Range - GR4
$3024 - $3027                       Ending Range - GR4

$3028 - $302B                    Beginning Range - GR5
$302C - $302F                       Ending Range - GR5

$3030 - $3033                    Beginning Range - GR6
$3034 - $3037                       Ending Range - GR6

$3038 - $303B                    Beginning Range - GR7
$303C - $303F                       Ending Range - GR7

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

$3040 - $3043                                 Data - GD0
$3044 - $3047                  Don't Care Data - GD0

$3048 - $304B                                 Data - GD1
$304C - $304F                  Don't Care Data - GD1

$3050 - $3053                                 Data - GD2
$3054 - $3057                  Don't Care Data - GD2

$3058 - $35B                                  Data - GD3
$305C - $305F                  Don't Care Data - GD3

$3060 - $3063                                 Data - GD4
$3064 - $3067                  Don't Care Data - GD4

$3068 - $306B                                 Data - GD5
$306C - $306F                  Don't Care Data - GD5

$3070 - $3073                                 Data - GD6
$3074 - $3077                  Don't Care Data - GD6

$3078 - $307B                                 Data - GD7
$307C - $307F                  Don't Care Data - GD7
```

One more note: the GR registers are 32 bits wide, but ranges are only valid for the first 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 this:

```
>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 GD0.

SECTION 8
MAINTENANCE AND TROUBLESHOOTING

8.1 MAINTENANCE

Maintenance of the ES-Series Satellite Emulator has been minimized by the extensive use of solid-state components throughout the instrument. There are only two areas where you need concern yourself with maintenance.

8.1.1 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 erractically or intermittently if the cables are flexed during the "RUN" mode. (You can also run a memory diagnostic and flex the cable during execution. If the diagnostic fails, the cable is faulty). By swapping the cables in question with a known good set of cables, you can easily isolate the faulty cable. The parts list at the end of this section contains cable part numbers if you need to order replacements.

8.1.2 Probe Tip Assembly

The Probe Tip Assembly is the small DIP header assembly that plugs into the target system CPU socket. The most obvious area to inspect is the 64-pin adapter, as the pins can be broken during insertion or extraction. If one of the pins should be inadvertently broken, you should replace the complete 40-pin adapter.

NOTE:

The 64-pin adapter can be protected by installing a CPU socket (male-female) onto the 64-pin adapter. If a pin is then broken on the CPU socket, it is easier to replace because of its common usage.

You should also inspect the probe tip assembly to see if any of the resistors have been broken. Due to close physical tolerance surrounding the resistors, we recommend that the probe tip assembly be returned to the factory for repair.

8.2 TROUBLESHOOTING

Your emulator is equipped with diagnostic test routines. The diagnostic programs are described in Section 7; if you need to perform any specific test, you should refer to the description in Section 7. Before starting troubleshooting procedures, be sure that interconnect cables are installed properly in a compatible target system, with power applied to both the target system and the emulator.

The most common problems encountered are listed in Table 8-1. We recommend that you contact Customer Service for ES Emulators 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

You will be asked for this information when you call.

NOTE:

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

Table 8-1.
Troubleshooting

SYMPTOM	POSSIBLE CAUSES	SECTION
Target system runs erratically	1. Faulty interconnect cables	8.1
	2. Intermittent contact on Probe Tip Assembly PC Card	*
	3. Broken pin on 64-pin adapter	2.3.3
	4. "Hold-tites" on Probe Tip Assembly missing (for connection to 64-pin adapter)	*
	5. Broken resistor on Probe Tip Assembly	8.1
	6. Emulator and target system not compatible	1.1
	7. LDV not executed before RUN (vector not loaded).	4.3.4
Emulator will not communicate over RS-232 line	1. Baud rate set incorrectly	2.4.1
	2. Target system requires "null" modem cable (pin 2 and pin 3 of RS-232 connector) reversed.	2.3.3
Target system will not run	1. Cables plugged in wrong	3.3
	2. Faulty interconnect cables	8.1
	3. Broken pin on 64-pin adapter	8.1 2.3
Unable to perform download	1. Transparent mode escape sequence not compatible with host	3.5 #15/23
	2. Host computer and computer port of ES need to be set at 4800 baud (RS-232 link may need to be slowed down)	3.5 #20
	3. Wrong format selected	3.5 #26

*Call Customer Service for ES Emulators
**Check Target System

8.3 PARTS LIST

The following parts are available for you to order:

64-pin Adapter	210-11412-00
Short Cable Set	600-11141-00
Long Cable Set	600-11142-00

APPENDIX A
SERIAL DATA FORMATS

The following sections detail the five serial data formats compatible with the Satellite Emulator. Each is illustrated in the accompanying figures.

A.1 MOS TECHNOLOGY FORMAT

A.2 MOTOROLA EXORCISOR FORMAT

A.3 INTEL INTELLEC 8/MDS FORMAT

A.4 SIGNETICS ABSOLUTE OBJECT FORMAT

A.5 TEKTRONIX HEXADECIMAL FORMAT

A.6 EXTENDED TEKHEX

A.1 MOS TECHNOLOGY FORMAT

The data in each record is sandwiched between a seven-character prefix and a four-character suffix. The number of data bytes in each record must be indicated by the byte count in the prefix. The input file can be divided into records of varios lengths.

Figure A-1 simulates a series of valid data records.

- Each data record begins with a semicolon (;). The Satellite Emulator ignores all characters received prior to the first semicolon. All other characters in a valid record must be valid hex digits (0 through 9 and A through F).
- A two-digit byte count follows the start character; the byte count, expressed in hexadecimal digits, must equal the number of data bytes in the record.
- The next four digits make up the address of the first data type in the record.
- Data bytes follow, each represented by two hexadecimal digits.
- The suffix is a four-character checksum, representing the sixteen-bit binary sum of all hexadecimal bytes in the record, including the address and byte count. Carry from the most significant bit is dropped.

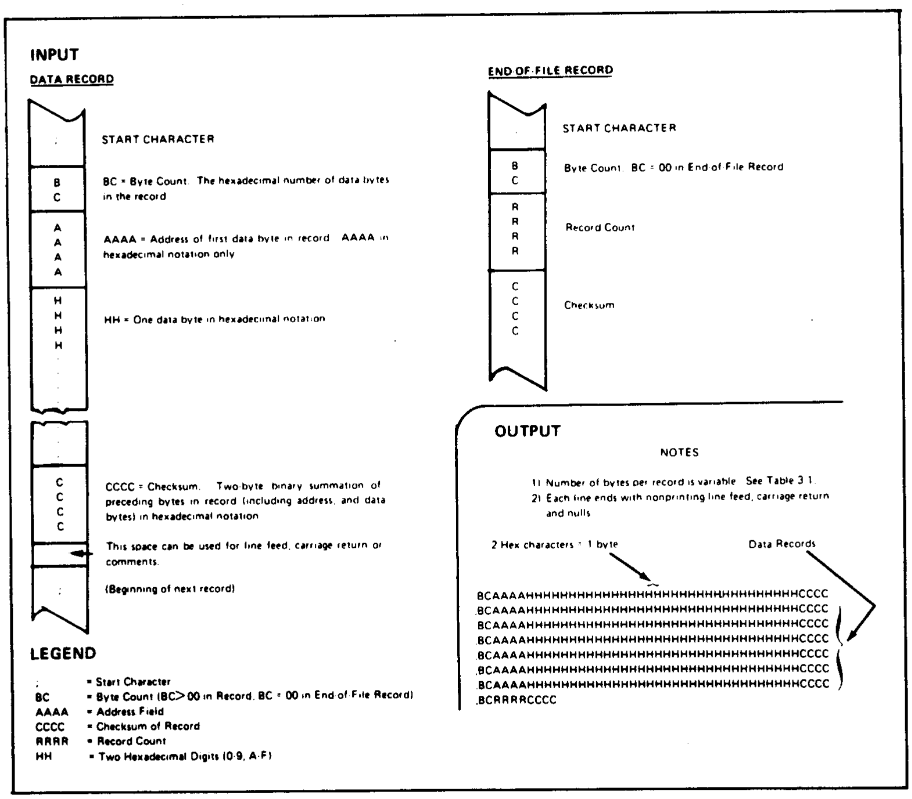


Figure A-1 Specifications for MOS Technology Data Files

A.2 MOTOROLA EXORCISER FORMAT

Motorola data files may begin with a sign-on record, initiated by the code S0. Valid data records start with an eight-character prefix and end with a two-character suffix.

Figure A-2 demonstrates a series of valid Motorola data records. S-record output format follows on pages A3a-A3d.

- Each data record begins with the start characters, "S1"; the emulator will ignore all earlier characters.
- The third and fourth characters represent the byte count, expressing the number of data, address, and checksum bytes in the record.
- The address of the first data byte in the record is expressed by the last four characters of the prefix.
- Data bytes follow, each represented by two hexadecimal characters. The number of data bytes occurring must be three less than the byte count.
- The suffix is a two-character checksum.

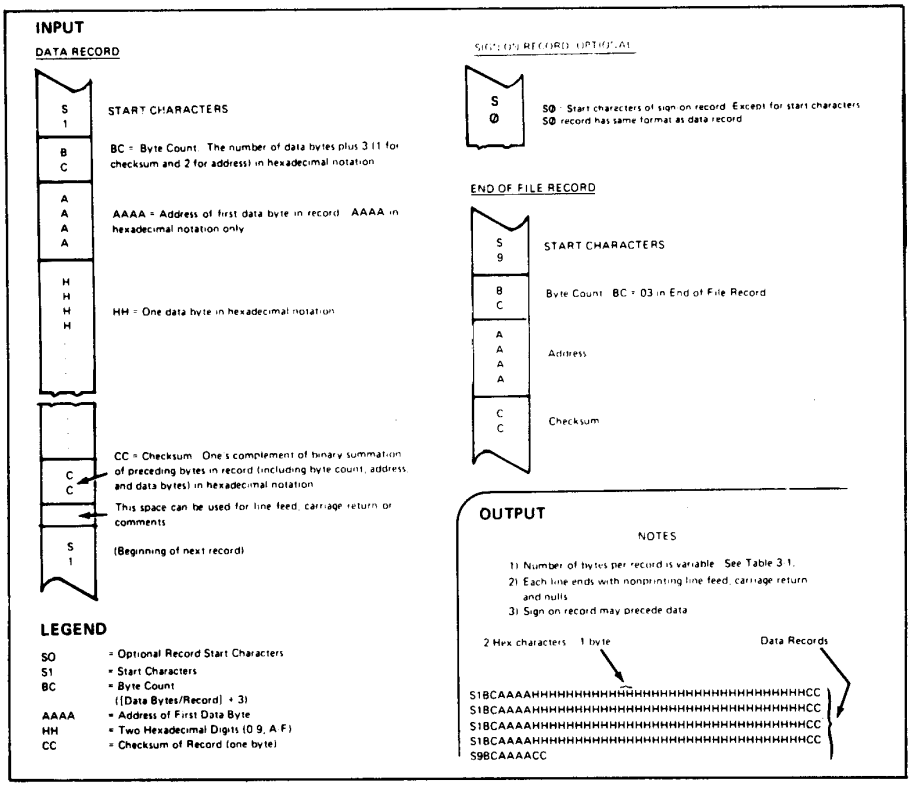


Figure A-2
Specifications for Motorola EXORciser 16-Bit Data Files

NOTE:
S2 and S8 records are also accepted. They have 24 bit addresses (six address characters) rather than the 18 shown.

A.3 INTEL INTELLEC 8/MDS FORMAT

Intel data records begin with a nine-character prefix and end with a two-character suffix. The byte count must equal the number of data bytes in the record.

Figure A-3 simulates a series of valid data records.

- Each record begins with a colon (:), followed by a two-character byte count.
- The four digits following the byte count give the address of the first data byte.
- Each data byte is represented by two hex digits; the number of data bytes in each record must equal the byte count.
- The suffix is a two-digit checksum, the two's complement of the binary summation of the previous bytes in the record.

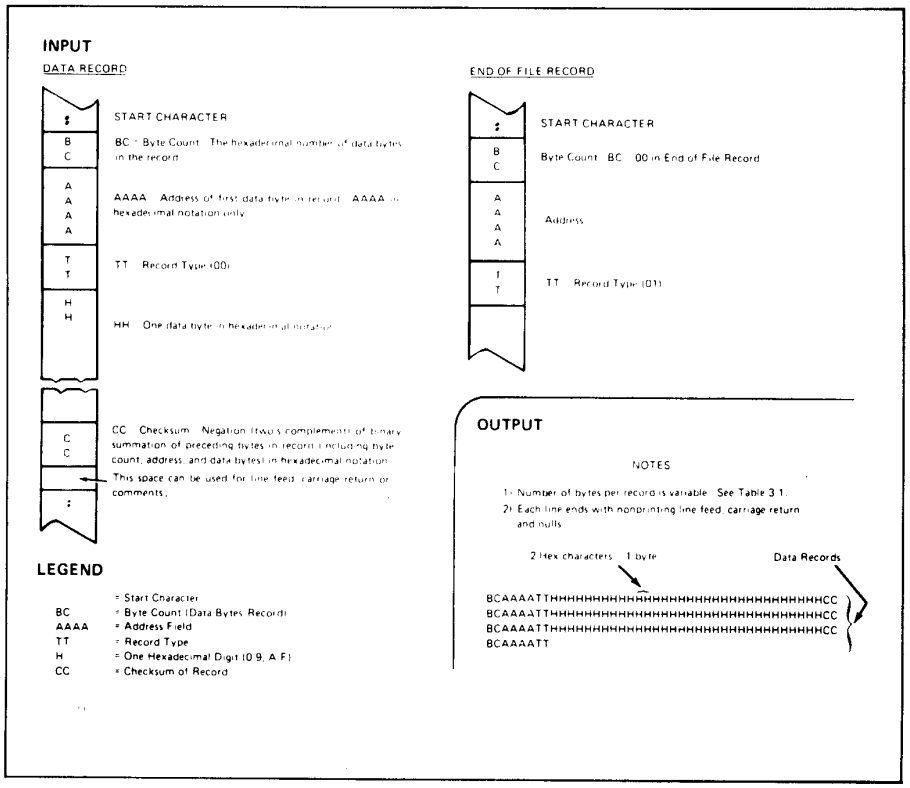


Figure A-3
Specifications
for Intel Intellec
8/MDS Data Files

A.4 SIGNETICS ABSOLUTE OBJECT FORMAT

Figure A-4 shows the specifications of Signetics format files. The data in each record is sandwiched between a nine-character prefix and a two-character suffix.

- The start character is a colon (:).
- This is followed by the address of the first data byte, the byte count, and a two-digit address check.

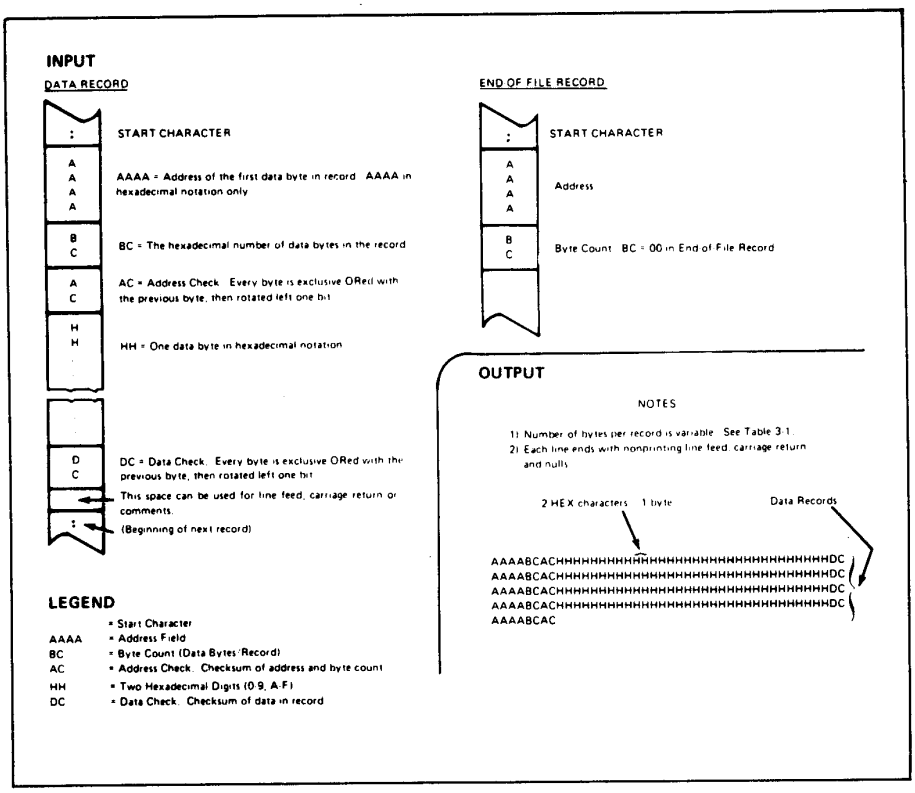


Figure A-4
Specifications for Signetics Absolute Object Data Files

A.5 TEKTRONIX HEXADECIMAL FORMAT

Figure A-5 illustrates a valid Tektronix data file. The data in each record is sandwiched between the start character and a two-character checksum.

- The start character is a slash (/).
- The next four characters of the prefix express the address of the first data byte.
- The address is followed by a byte count, representing the number of data bytes in the record, and by a checksum of the address and byte count.
- Data bytes follow, represented by pairs of hexadecimal characters and succeeded by a checksum of the data bytes.
- The End-Of-File record consists only of control characters, used to signal the end of transmission, and a byte count and checksum for verification.

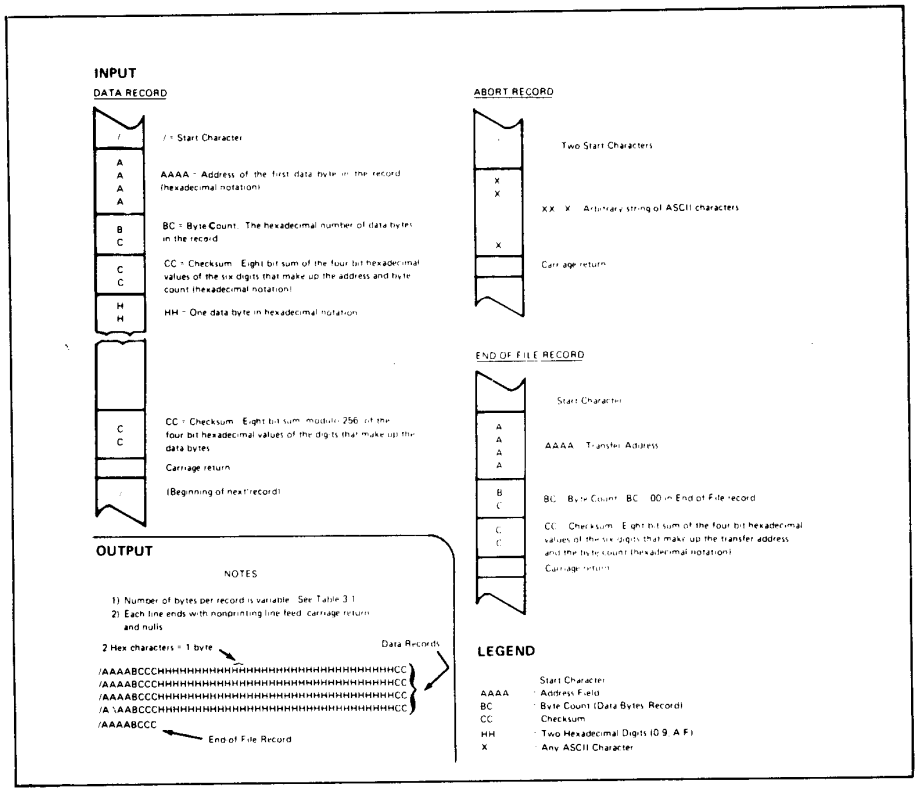


Figure A-5
Specification
for Tektronix
Hexadecimal
Data Files

A.6 EXTENDED TEKHEX

Extended Tekhex uses three types of message blocks:

- A data block contains object code.
- A symbol block contains information about a program section and the symbols associated with it. This information is needed only for symbolic debug.
- A 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, as defined earlier in this section.

Each block begins with a six-character header field and ends with an end-of-line character sequence (on the 8550, a RETURN). A block can be up to 255 characters long, not counting the end-of-line. A header field has the format shown in Table A-1.

Table A-1
Extended Tekhex Header Field

Item	Number of ASCII Characters	Description
%	1	A percent sign specifies that the block is in Extended Tekhex format.
Block Length	2	The number of characters in the block; a two-digit hex number. this count does not include the leading % or the end-of-line.
Block Type	1	6 = data block 3 = symbol block 8 = termination block
Checksum	2	a two-digit hex number representing the sum, mod 256, of the values of all the characters in the block, except the leading %, the checksum digits, and the end-of-line. Table A-2 gives the values for all characters that may appear in Extended Tekhex message blocks.

Table A-2
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

A.6.1 Variable-Length Fields

In Extended Tekhex, certain fields may vary in length from 2 to 17 characters. This practice enables you to compress 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.

A.6.2 Data and Termination Blocks

If you do not intend to transfer program symbols with your object code, you can do without symbol blocks. Your load module can consist of one or more data blocks, followed by a termination block. Table A-3 gives the format of a data block, and Table A-4 gives the format of a termination block.

Table A-3
Extended Tekhex Data Block Format

FIELD	NUMBER 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.

Table A-4
Extended Tekhex Terminal Block Format

FIELD	NUMBER OF ASCII CHARACTERS	DESCRIPTION
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.

A.6.3 Symbol Blocks

A symbol used in symbolic debug has the following attributes:

1. The symbol itself: 1 to 16 letters, digits, dollar signs, periods, or underscores. The first character of the symbol can be a letter or (if the symbol is a section name) a percent sign. 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). Symbolic debug does not currently use the code/data distinction, so 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. The concept of global symbols is discussed in the Assembler Core Manuals for both A Series and B Series assemblers. 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.

Table A-5 gives the format of a symbol block. Tables A-6 and A-7 give the formats for section definition fields and symbol definition fields, which are parts of a symbol block.

Table A-5
Extended Tekhex
Block Format

FIELD	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. Table A-6 gives the format for this field.
Symbol Definition(s)	5 to 35	Zero or more symbol definition fields, each as described in Table A-7.

Table A-6
Extended Tekhex
Symbol Block
Section Definition
Field

ITEM	NUMBER OF ASCII CHARACTERS	DESCRIPTION
0	1	A zero signals a section definition field.
Base Address	2 to 17	The starting address of the 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).

Table A-7
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.

NOTE

Symbol records are currently ignored by the emulator.

Figure A-8 shows how this information might be encoded in Extended Tekhex symbol blocks. (All this information could be encoded in a single 96-character block.) It is divided into two blocks for purposes of illustration.)

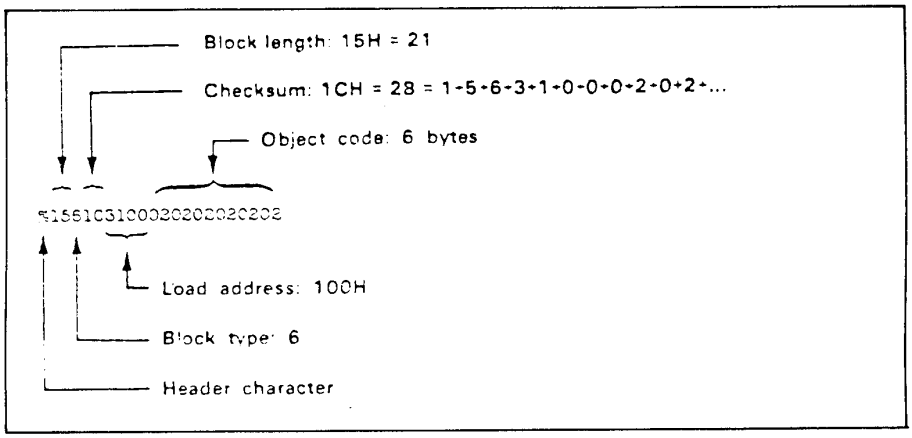
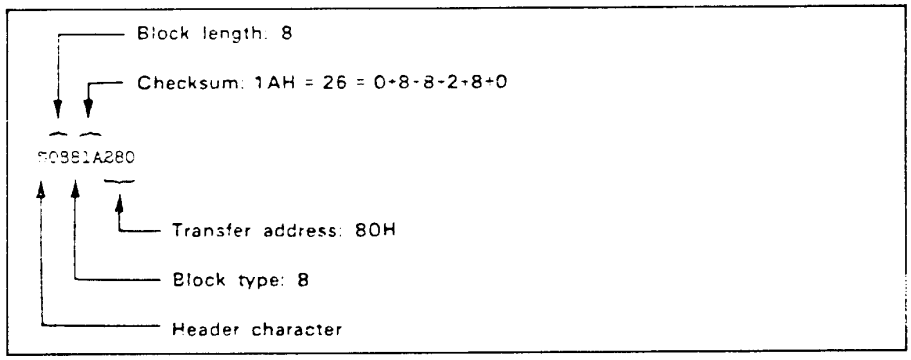


Figure A-6.
Tekhex Data Block



ure A-7
ekhex Termination Block

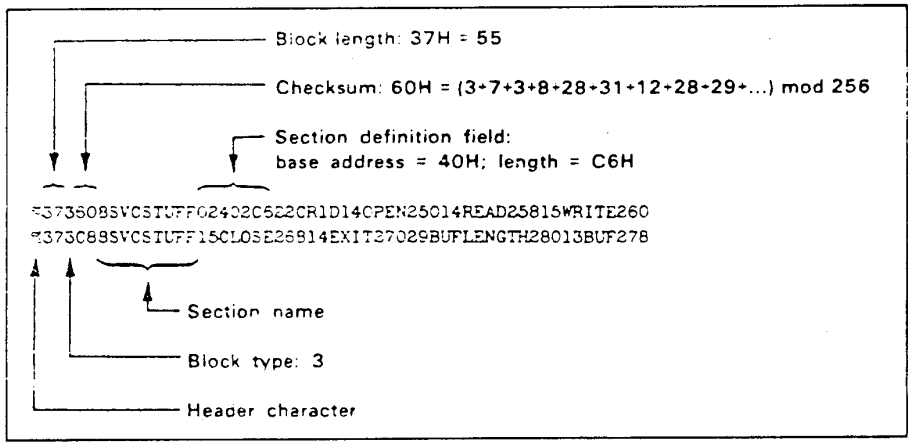


Figure A-8
Tekhex Symbol Block

APPENDIX B
REFERENCE MATERIAL

B.1 Glossary

argument. An independent variable; the number or numbers that identifies the location of a desired value.

baud. The shortest code element computed to a unit of signaling speed. The speed in baud equals the number of code elements per second.

breakpoint. A point in a program where an external source can intervene by giving a specific instruction to interrupt the normal sequence of operations. The normal sequence can be resumed after the interruptions used for debugging or visual checks on a terminal are terminated.

default. An option or value that is assumed provided another one has not been specified.

disassembly (disassembler). A program that converts binary instructions into their symbolic mnemonic representation.

don't care. A term applied to an input or output value that is irrelevant to the specific operation or consideration.

duplex. Communications in a two-way independent transmission moving in both directions.

echo. Part of a transmitted signal recognized and received as interference because of the magnitude and delay of the signal reflected back.

EEPROM. Electrically Eraseable Programmable Read Only Memory.

error code. A marking that indicates error by a code.

host system. The system that controls; for example, the development system, minicomputer, or automatic test equipment (ATE) system.

indirection. The term means indirect addressing; particularly with respect to the mechanism that performs it.

logic state analyzer (LSA). Monitors a system or component board and shows the monitored information to be reviewed.

mainframe. A reference to large computers to distinguish them from microcomputers, microprocessors, and minicomputers. With respect to the ES1800 Satellite Emulator, the mainframe houses the emulator board, RAM Overlay Board, the controller board, the trace and break board, the memory control board, and the power supply.

memory map. A table or drawing representing the memory locations for devices, programs, or functions.

modulo. The result of a mathematical operation of a specified number that has been divided leaving a remainder. The remainder equals the modulo.

operator. The element in an operation that defines what action is to be performed on the operand.

parameter. A quantity which may be given variable values.

parity. A method of self-checking the accuracy of binary number transmission.

run. A term describing the execution of emulation.

run with breakpoints. The execution of a program with temporary halts to permit the operator to make some checks.

statement. A generalized instruction or syntactically complete string of characters.

step. Single step operation.

stop bit. One or two 1-bits used as a character delimiter in start-stop transmission.

target system. With respect to emulation, the target system is the computer (your hardware) that is emulated.

Trace Memory. Functions as a history of target system program execution.

XOFF. Transmitter off.

XON. Transmitter on.

Table B-1.
Number Bases Cross Reference

BINARY				OCTAL	HEXADECIMAL	DECIMAL	STANDARD ABBREVIATION
			0000	0	0	0	
			0001	1	1	1	
			0010	2	2	2	
			0011	3	3	3	
			0100	4	4	4	
			0101	5	5	5	
			0110	6	6	6	
			0111	7	7	7	
			1000	10	8	8	
			1001	11	9	9	
			1010	12	A	10	
			1011	13	B	11	
			1100	14	C	12	
			1101	15	D	13	
			1110	16	E	14	
			1111	17	F	15	
		0001	0000	20	10	16	
		0010	0000	40	20	32	
		0100	0000	100	40	64	
		1000	0000	200	80	128	
	0001	0000	0000	400	100	256	
	0010	0000	0000	1000	200	512	
	0100	0000	0000	2000	400	1,024	1K
	1000	0000	0000	4000	800	2,048	2K
	1100	0000	0000	6000	C00	3,072	3K
0001	0000	0000	0000	10000	1000	4,096	4K
0001	0100	0000	0000	12000	1400	5,120	5K
0001	1000	0000	0000	14000	1800	6,144	6K
0001	1100	0000	0000	16000	1C00	7,168	7K
0010	0000	0000	0000	20000	2000	8,192	8K
0010	0100	0000	0000	22000	2400	9,216	9K
0010	1000	0000	0000	24000	2800	10,240	10K
0100	0000	0000	0000	40000	4000	16,384	16K
1000	0000	0000	0000	100000	8000	32,768	32K
0001 0000	0000	0000	0000	200000	10000	65,536	64K

7 6 5	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	
BITS 1 2 3 4	CONTROL		& SYMBOLS		NUMBERS UPPER CASE		LOWER CASE	
0 0 0 0	0 NUL 0 0	20 DLE 10 16	40 SP 20 32	60 0 30 48	100 @ 40 64	120 P 50 80	140 ` 60 96	160 p 70 112
0 0 0 1	1 SOH 1 1	21 DC1 11 17	41 ! 21 33	61 1 31 49	101 A 41 65	121 Q 51 81	141 a 61 97	161 q 71 113
0 0 1 0	2 STX 2 2	22 DC2 12 18	42 " 22 34	62 2 32 50	102 B 42 66	122 R 52 82	142 b 62 98	162 r 72 114
0 0 1 1	3 ETX 3 3	23 DC3 13 19	43 # 23 35	63 3 33 51	103 C 43 67	123 S 53 83	143 c 63 99	163 s 73 115
0 1 0 0	4 EOT 4 4	24 DC4 14 20	44 $ 24 36	64 4 34 52	104 D 44 68	124 T 54 84	144 d 64 100	164 t 74 116
0 1 0 1	5 ENQ 5 5	25 NAK 15 21	45 % 25 37	65 5 35 53	105 E 45 69	125 U 55 85	145 e 65 101	165 u 75 117
0 1 1 0	6 ACK 6 6	26 SYN 16 22	46 & 26 38	66 6 36 54	106 F 46 70	126 V 56 86	146 f 66 102	166 v 76 118
0 1 1 1	7 BEL 7 7	27 ETB 17 23	47 ' 27 39	67 7 37 55	107 G 47 71	127 W 57 87	147 g 67 103	167 w 77 119
1 0 0 0	10 BS 8 8	30 CAN 18 24	50 (28 40	70 8 38 56	110 H 48 72	130 X 58 88	150 h 68 104	170 x 78 120
1 0 0 1	11 HT 9 9	31 EM 19 25	51) 29 41	71 9 39 57	111 I 49 73	131 Y 59 89	151 i 69 105	171 y 79 121
1 0 1 0	12 LF A 10	32 SUB 1A 26	52 * 2A 42	72 : 3A 58	112 J 4A 74	132 Z 5A 90	152 j 6A 106	172 z 7A 122
1 0 1 1	13 VT B 11	33 ESC 1B 27	53 + 2B 43	73 ; 3B 59	113 K 4B 75	133 [5B 91	153 k 6B 107	173 { 7B 123
1 1 0 0	14 FF C 12	34 FS 1C 28	54 , 2C 44	74 < 3C 60	114 L 4C 76	134 \ 5C 92	154 l 6C 108	174 7C 124
1 1 0 1	15 CR D 13	35 GS 1D 29	55 - 2D 45	75 = 3D 61	115 M 4D 77	135] 5D 93	155 m 6D 109	175 } 7D 125
1 1 1 0	16 SO E 14	36 RS 1E 30	56 . 2E 46	76 > 3E 62	116 N 4E 78	136 ^ 5E 94	156 n 6E 110	176 7E 126
1 1 1 1	17 SI F 15	37 US 1F 31	57 / 2F 47	77 ? 3F 63	117 O 4F 79	137 — 5F 95	157 o 6F 111	177 Rubout 7F 127
KEY	Addressed Commands	Universal Commands	Listen Addresses		Talk Addresses		Secondary Addresses or Commands	

octal
25 PPU
NAK
15 21
GPIB code
ASCII character
decimal

Table B-3.
ASCII and IEEE Code Chart

Table B-3.
ASCII Control Characters

ACK	acknowledge
BEL	bell
BS	backspace
CAN	cancel
CR	carriage return
DC1	playback on, CNTL Q, X-ON
DC2	record on, CNTL R, PUNCH-ON, SOM
DC3	playback off, CNTL S, X-OFF
DC4	record off, CNTL T, PUNCH-OFF, EOM
DEL	delete, rubout
DLE	data link escape
EM	end of medium
ENQ	enquiry
EOT	end of transmission
ESC	escape
ETB	end of transmission block
ETX	end of text
FF	form feed
FS	file separator
GS	group separator
HT	horizontal tabulation
LF	line feed
NAK	negative acknowledge
NUL	null
RS	record separator
SI	shift in
SO	shift out
SOH	start of heading
STX	start of text
SUB	substitute
SYN	synchronous idle
US	unit separator
VT	vertical tab

APPENDIX C
SYMBOLIC DEBUG

The symbolic debug option allows easier debugging, using a wider range of capabilities. These 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

The only standard symbolic format currently accepted is extended Tekhex. If you are using another symbolic format, please see the usage note at the end of this appendix.

C.1 COMMANDS

- **Implicit symbol definition and symbol value change**

  > '<SYMBOL> = <VALUE>

If SYMBOL is undefined, it is placed into the symbol table and assigned the value VALUE. If SYMBOL was previously defined, it will be reassigned the value VALUE.

--<VALUE> is a 32-bit integer value. Don't cares are not allowed in symbolic definitions.

--<SYMBOL> is any combination of the ASCII characters with decimal values in the range 33-126. This range includes all of the printable ASCII characters. Symbols are delineated by a single starting quote (') and the first blank space or RETURN. Symbols can be up to 64 characters long, although only the first 16 characters are displayed with symbolic disassembly.

- **Symbolic reference**

  > '<SYMBOL> ;GRO = '<SYMBOL> ;'<SYMBOL> + $41900;...

  The reference to 'SYMBOL will be exactly like referencing any of the common registers in the ES, with the exception that symbols not at the end of the command line must be terminated with a space.

- **Displaying symbols**

  >SYM [VALUE]

  This displays the symbol(s) that have been assigned the value VALUE. If no argument is entered, all symbols and their values will be displayed.

- **Section definition**

  >'<SYMBOL> = <RANGE>

  Any symbol that is assigned a range value will, by definition, be a section. <RANGE> is a standard ES 24-bit range value.

  NOTE:

  Overlapping sections and sections with the same name as a symbol are illegal.

- **Display of section values**

  >SEC [<VALUE>]

  The section containing the value will be displayed along with its assigned values. If no argument is entered, all section names and values will be displayed.

- **Deletion of a symbol or section**

  >DEL '<SYMBOL>

  This will remove the symbol or section definition

- **Clearing symbolic memory**

  >PUR

  This command permanently removes all symbol data from ES memory.

- **Upload and download of symbolic information**

  >UPS

  This command uploads all symbols and sections in extended Tekhex format.

  --Sections are defined in separate records.

  --Symbols are defined as belonging to the section "m".

  Extended Tekhex restricts the number and range of characters that can be used for a symbol name. The ES will truncate symbols to 16 characters and will substitute **%** for characters not allowed by Tekhex.

  >DNL

  This command will accept symbolic definition records as well as data records if the ES download format variable is set to 5 (extended Tekhex).

The use of symbols in disassembly allows the ES to display trace data in a more useful format. Disassembly with defined symbols will display the symbol name everywhere there is an address reference that matches the symbol's value. Section names will be shown whenever the program addresses fall within a defined section. Also, when in a defined section, the program addresses will be displayed as offset values from the beginning of that section.

This example outlines these points. The first disassembly is without any defined symbols. The second disassembly shows the effect of the three symbolic definitions. Note how the program address display mode changes as the addresses move out of the section.

```
   >DT 0 LEN #10
SEQ#  ADDR  OPCODE   MNEMONIC   OPERAND FIELDS BUS CYCLE DATA
-----------------------------------------------------------------
0009 000166 31C23000  MOVE.W    D2,$3000  003000<8787
0008 00016A D081      ADD.L     D1,D0
0007 00016C 64000004  BCC.L     $000172
0006 000172 D885      ADD.L     D5,D4
0005 000174 64F0      BCC.S     $000166
0004 000166 31C23000  MOVE.W    D2,$3000  003000<8787
0003 00016A D081      ADD.L     D1,D0
0002 00016C 64000004  BCC.L     $000172
0001 000172 D885      ADD.L     D5,D4
0000 000174 64F0      BCC.S     $000166
 >
 >'Loop = 166
 >'Demon.module = 'Loop TO 16C
 >'I/O_port_0 = 3000
 >DT 0 LEN #10
SEQ#  ADDR  OPCODE   MNEMONIC   OPERAND FIELDS BUS CYCLE 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
>
```

C.2 USAGE NOTE FOR USERS WITH SYMBOLIC FORMATS OTHER THAN EXTENDED TEKHEX

Of the three methods of entering symbolic data, downloading from the host using DNL is preferable since it is not only fast, but includes error checking in the transmission of your data. However, if you are working with any symbolic format other than Extended Tekhex, you will not be able to use this method. Two alternates are available: both require that you convert the symbolic format that you are using before you enter the symbolic data.

- For very small programs, you can enter symbolic data manually from a symbol map as follows:

  ><symbol> =<value><return>

- For other applications, you would want to put the Satellite Emulator under computer control, using CCT. This method is just as fast as downloading; however, no error checking is performed. You must write a program that converts your symbolic data as shown above; the program can then transmit the strings to the emulator.

There are four characteristics to remember about CCT.

First, the emulator will echo most of the characters sent to it, so the computer can use this feature to check the data transmission.

Second, when the host sends a RETURN, the emulator begins processing the command line. New lines generally begin with RETURN LINEFEED NULL NULL.

Third, the host must be able to handle incoming data at high rates as the emulator will be sending at 9600 baud; the host should be able to send XON/XOFF to the emulator.

Fourth, UPL (upload) and DNL (download) expect data from the same port whether you are using TCT or CCT: if you are downloading the emulator always expects data to come from the host, and if you are uploading data is always sent to the host.

APPENDIX D
S-RECORD OUTPUT FORMAT

D.1 S-RECORD OUTPUT FORMAT

The S-record format for output modules was devised for the purpose of encoding programs or data files in a printable format for transportation between computer systems. The transportation process can thus be visually monitored and the S-records can be more easily edited.

D.1.1 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 shown below:

type	record length	address	code data	checksum

Where 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 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 wih a CR/LF/NULL. Additionally, an S-record may have an initial field to accomodate 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.

D.1.2 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.

S0 The header record for each block of S-records. The code/data field may contain any descriptive information identifying the following block of S-records. Under VERSAdos, the resident linker's IDENT command can be used to designate module name, version number, revision number, and the description information which will make up the header record. The address field is normally zeroes.

S1 A record containing code/data and the 2-byte address at which the code/data is to reside.

S2 A record contaning 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.

.2 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 a 16-bit microprocessor-based system. Programs are also available for uploading an S-record file to or from an EXORmacs system.

Example

Shown below is a typical S-record-format module, as printed or displayed:

```
S0060000484421B
S1130000285F2212226A00044290008237C2A
S1130010000200080008262900185381234100l813
S113002041E900084E42234300182342000824A952
S107003000144Ed492
S9030000FC
```

The module consists of one SØ record, four S1 records, and an S9 record.

The S0 records 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, zeroes 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
00 + Four-character 2-byte address field; hexadecimal address 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 program 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)

(The balance of this code is continued in the code/data fields of the remaining S1 records, and stored in memory location ØØ1Ø, 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 Ø7 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.

Ø3 Hexadecimal Ø3, indicating that three character pairs (3 bytes) follow.

ØØ
ØØ The address field, zeroes.

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. For example, the first S1 record above is sent as:

type				length				address								code/data												...	checksum			
S		1		1		3		0		0		0		0		2		8		5		F		...	2		A				
5	3	3	1	3	1	3	3	3	0	3	0	3	0	3	0	3	2	3	8	3	5	4	6	...	3	2	4	1				
0101	0011	0011	0001	0011	0001	0011	0011	0011	0000	0011	0000	0011	0000	0011	0000	0011	0010	0011	1000	0011	0101	0100	0110	...	0011	0010	0100	0001				

INDEX TO TOPICS

A

B

C