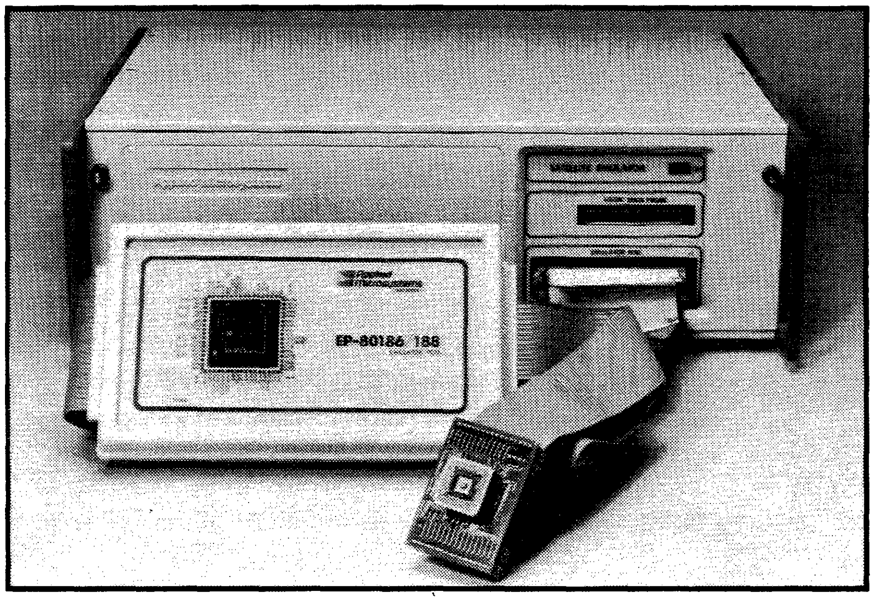


ES 1800 SATELLITE EMULATOR OPERATOR'S MANUAL

FOR 80186/88 MICROPROCESSORS

Applied Microsystems CORPORATION

5020 148th Avenue N.E.
P.O. Box 97002
Redmond, WA 98073-9702
(206) 882-2000 1-800-426-3925

920-11437-01

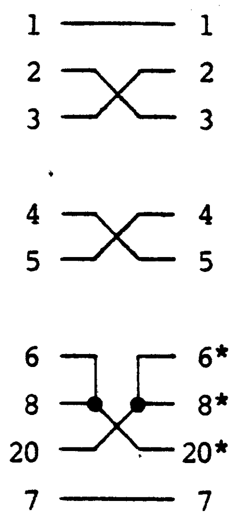
December 1985

SET-UP CHECKLIST
80186/88

Please read this checklist completely before using your new Applied Micro-systems' Satellite Emulator.

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

2. **If using communications without a modem, you may need a null modem cable.** Example:



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

3. At a minimum, **you should review sections applicable to the steps listed here, plus "Getting Started With The 80186 or 80188."**

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

PLACE CHECKLIST INSIDE FRONT COVER FOR FUTURE REFERENCE

Applied Microsystems CORPORATION
5020 148th Avenue N.E.
Redmond, WA 98052
or
Box C-1002
Redmond, WA 98073-1002
Toll Free Service: 1-800-426-3925

GETTING STARTED WITH THE 8Ø186 or 8Ø188

The 8Ø186 or 8Ø188 microprocessors include on-chip peripheral circuits which offer many benefits to the user. The addition of these on-chip peripheral circuits adds a degree of complexity to development workstations not previously encountered with either the 8Ø86 or 8Ø88 microprocessors. As you might expect, this adds greatly to the complexity of emulation, particularly in the area of setting up an emulator to run with the target system.

This document should help identify obstacles to the installation and operation of the emulator/target workstation. To help ensure a successful and speedy start up, please follow these step-by-step instructions.

STEP 1: SET UP THE EMULATOR/TARGET WORKSTATION

Connect the emulator to your target. Refer to section 2 of the Operator's Manual.

Apply power to your emulator then to your target system.

STEP 2: SET UP THE PCB RELOCATION REGISTER

Note: If you do not relocate the Peripheral Control Block from $FFØØ in I/O space, then go to Step 3.

ENTER: REL = register-value <Return>*

***Refer to an Intel 8Ø186 User's Guide for the proper way to set up the register.**

STEP 3: SET UP THE ON-CHIP CHIP SELECT PERIPHERAL

Note: If you do not use on-chip chip selects, then go to Step 4.

ENTER: ON RCS <Return>

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

PAUSE-TO-RUN transitions will write the emulator chip select PCB values into the target PCB.
RUN-TO-PAUSE transitions will read the emulator chip select PCB values from the target PCB.

ENTER: Values needed for UMCS, LMCS, MPCS, MMCS and PACS registers, as: UMCS = register_value <Return>.*

***Refer to an Intel 8Ø186 User's Guide for the proper way to set up the register.**

STEP 4: SET UP THE ON-CHIP DMA PERIPHERAL

Note: If on-chip DMA circuitry is not used, then go on to Step 5.

ENTER: Values needed for USRCO, SRCO, UDSTO, DSTO, XCO and CWO registers, as: USRCO = register_value <Return>.*

***Refer to the intel 80186 User´s Guide for the proper set up.**

Note: If you do not need DMA active while paused, then go on to Step 5.

ENTER: ON DME <Return>

STEP 5: SET UP THE ON-CHIP TIMER PERIPHERAL

Note: If on-chip timer circuitry is not used, then go on to Step 6.

ENTER: Values needed for TCØ, TC1, TC2, MAØ, MA1, MA2, MBØ, MB1, MB2, MCWØ, MCW1 and MCW2 registers, as:
TCØ = register_value <Return>.*

***Refer to the Intel 80186 User´s Guide for the proper set up.**

Note: If you need a timer circuit active while paused, then turn on the appropriate emulator software switch, as:
ON TEØ <Return> for timer zero, and/or
ON TE1 <Return> for timer one, and/or
ON TE2 <Return> for timer three.

STEP 6: SET UP THE ON-CHIP INTERRUPT CONTROL PERIPHERAL

Note: If on-chip Interrupt Control circuitry is not used, then proceed to Step 7.

ENTER: Values needed for INTØ, INT1, INT3, EOI, POL, POS, MSK, PLM, ISV, IRQ, IST, TCR, DMAØ, DMA1, and DMA2 registers, as:
INTØ = register_value <Return>*

***Refer to the Intel 80186 User´s Guide for the proper set up.**

STEP 7: SET UP OVERLAY AND A MINIMAL PROGRAM

Note: This step assumes you have neither target memory nor a valid program located at the startup location (*FFFFØ). If you have target memory and a valid program then go on to Step 8.

ENTER: MAP $FF8ØØ;DM <Return>
This will map in overlay from $FF8ØØ to $FFFFF and display the memory map.

ENTER: ON RDY <Return>
This will ensure that Reads and Writes to overlay use the emulator's internal ready signal.

ENTER: ASM <Return>
This will invoke the single-line assembler to enter a sequence of NOP instructions.

ENTER: CSEG = ØFFFF <Return>
This will set the assembler to an absolute address of $FFFFØ.

ENTER: NOP <Return> (repeat several times)
This throw-away program is used to initialize the on-chip peripheral circuitry.

ENTER: X <Return>
This will exit the assembler.

STEP 8: ACTIVATE THE ON-CHIP PERIPHERALS

Note: The following tasks should have been accomplished before reaching this point:
1) The state of all on-chip peripherals should have been set up via the PCB registers.
2) The emulator's ON or OFF software switches have been properly set up.
3) A program resides at the start up location ($FFFFØ).

ENTER: AC1 = stopping_point <Return>
This will define the stopping point of the program which should follow the initialize section.

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

AC1 = $FFFF2 (if manually initializing and using the NOP program in Step 7),

or,

AC1 = stopping_point (if using your own PCB initialization program).

ENTER: WHEN AC1 THEN BRK
This will allow a breakpoint when AC1 is recognized during emulation.

ENTER: RST;RBV <Return>

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

RBV sets CS:IP registers to the absolute address of $FFFF0, activates the event monitor system, and initiates a real-time run.

This concludes the initial startup procedure. At this point, the PCB has been initialized and the emulator set up for further use. You should be able to perform normal emulator functions.

Because this configuration of the ES1800 Satellite Emulator is intended for use in developing, debugging, and testing Intel 80186/88 microprocessor-based systems, it is presumed that the user is familiar with the terminology of the 80186/88 microprocessors.

SECTION 4. OPERATION

SECTION 5. PROGRAMMING THE EVENT MONITOR SYSTEM

QUICK INDEX TO OPERATORS

OPERATOR	NAME	PAGE	SECTION NUMBER
ABS	absolute value	3-12	3.4.2
AC1, AC2	address comparators 1 and 2	5-4	5.3.1
ALT	alternate data access	5-8	5.3.4
AND	logical event AND	5-13	5.6
AX, AL, AH	accumulator (low and high)	4-2	4.2
BAS	display base value	3-4	3.3.3
BKX	break on instruction execution (not prefetch)		
BMO	block move	4-12	4.5.3
BP	base pointer	4-2	4.2
BRK	break	5-10	5.4
BUS	display status of lines	7-4	7.5
BX, BL, BH	base register (low and high)	4-2	4.2
BYM	byte mode	4-6	4.4.4
BYT	byte status	5-11	5.5.5
CCT	Computer Control	6-3	6.3.2
CD	overlay enable for code access	4-11	4.5.3
CES	clear Event Monitor System When/Then statements	5-3	5.2
CK	select internal clock	3-13	3.5
CLK	measure target system clock	7-3	7.4
CLM	clear memory map	4-10	4.5.2
CNT	count event	5-10	5.4
CPY	copy switch	3-15	3.5
CRC	calculate cyclic redundancy check in target system	7-3	7.4
CS	code segment	4-2	4.2
COD	code status	5-8	5.3.4
COM	communicate with program running in system	7-5	7.6
CTL	count limit	5-9	5.5.2
CWØ, CW1	dma control words Ø and 1	4-2	4.2
CX, CL, CH	count register (low and high)	4-2	4.2
DAT	data access status	4.8	4.4.6
DB	display memory block	4-8	4.4.7
DC	Don't Care	5-9	5.3.5
DC1,DC2	data comparators 1 and 2	5-6	5.5.3
DEL	delete symbol	6-3	6.1
DES	display Event Monitor System When/Then statements	5-3	5.2
DFB	default base value	3-4	3.3.2
DI	destination index	4-2	4.2
DIS	display disassembled memory	4-21	4.9.1
DM	display memory map	4-9	4.5.2
DMA	DMA cycle status	5-8	5.3.4
DMAØ, 1	dma interrupt control regisers (Ø.1)	4-3	4.2
DMAØ (iRMX mode)	level 2 control register (DMA Ø)	4-3	4.2
DMA1 (iRMX mode)	level 3 control register (DMA 1)	4-3	4.2
DME	enable DMA timer during pause	3-14	3.5

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 CONCEPTS

The Applied Microsystems ES 1800 Satellite Emulator (Figure 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 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 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.

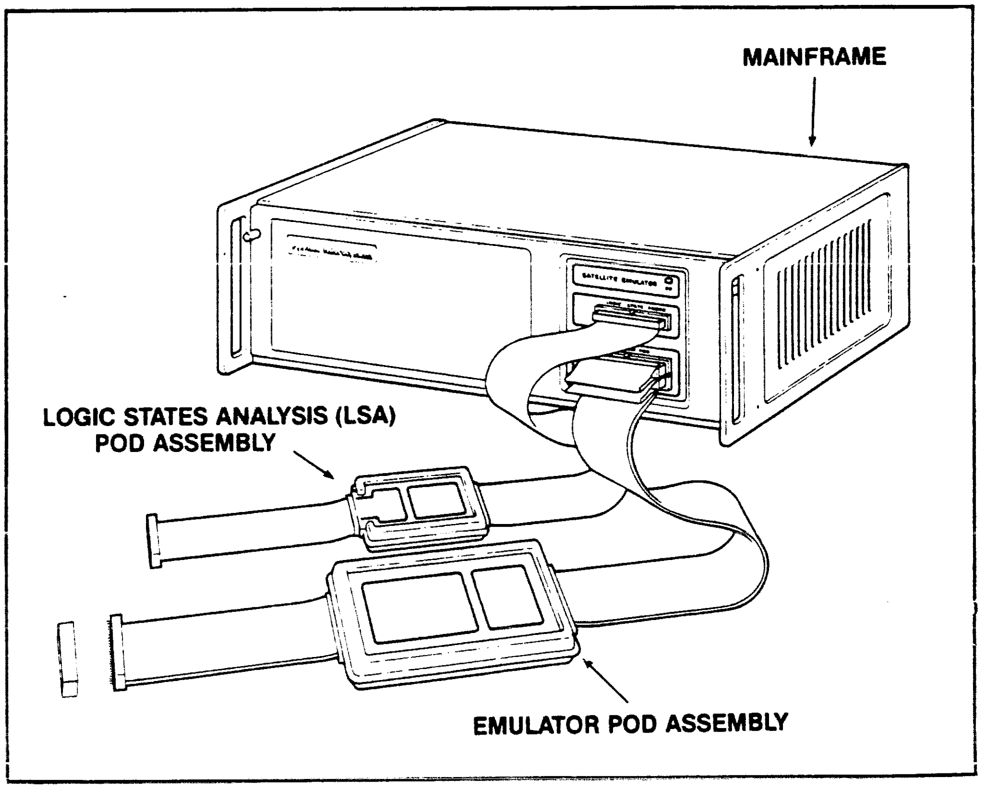


Figure 1-1.
The Satellite Emulator

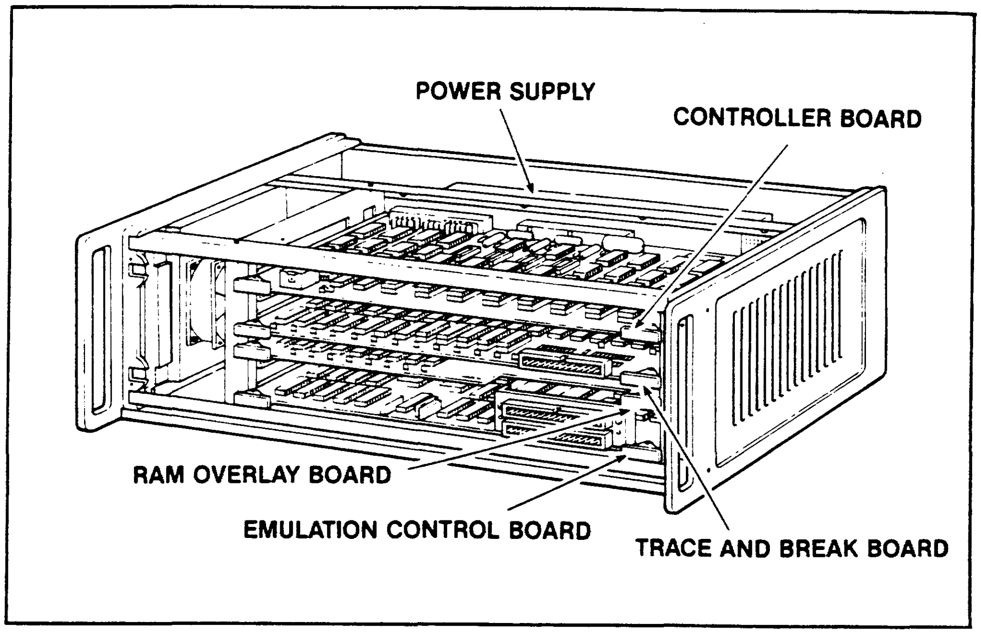


Figure 1-2. Mainframe Components

MEMORY CONTROLLER BOARD (MCB)-NOT SHOWN. It is normally between the controller board and the trace and break board. (See Figure 2-4).

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 special probe tip connects the 11-inch ribbon cable to the target system. The probe tip has been designed to be compatible with sockets for the 80186/80188.

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, computer, 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.

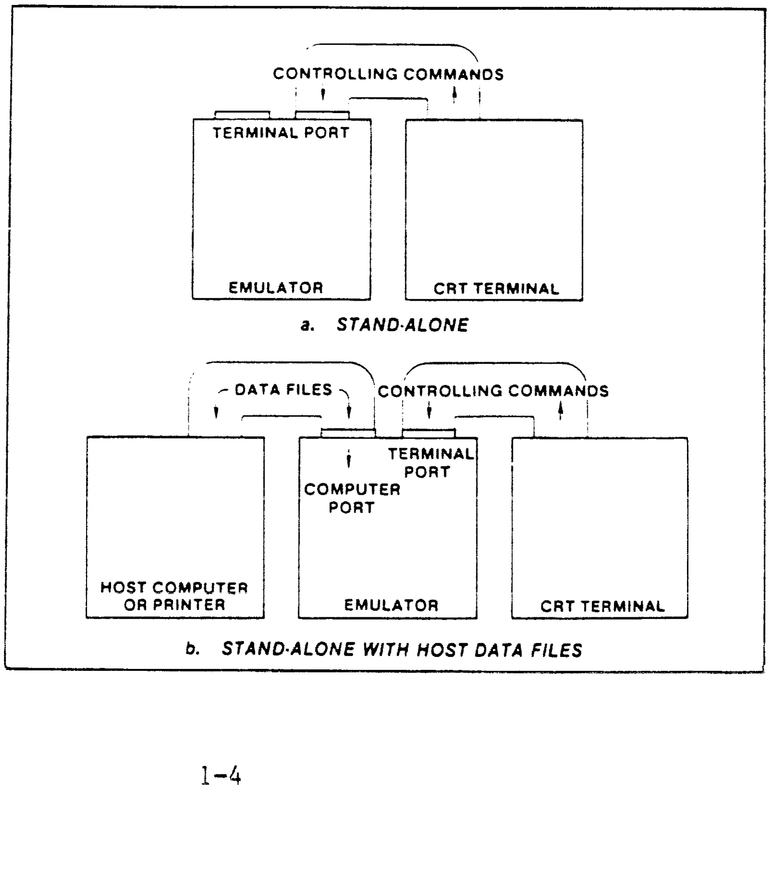


Figure 1-3. System Configurations

1.1.5 System Features

Table 1-1 summarizes the system features of the emulator. Features can be combined in various ways to form an emulation system that fulfills your exact needs. Section 3 gives a detailed description of how features 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, macros, switch settings, Event Monitor System parameters, and the memory map. Parameters can be accessed and changed at any time during an emulation session while the target system is stopped. 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 during the target system program. Possible 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 (pod assembly option)	Provides external logic signal recording and event detection capability (16 inputs to a 16 x 2046-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.
Internal Clock	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.
Macros	Up to ten macros can be defined and called out. See Section 4.11.
Software Switches	Allow you to set system parameters. See Section 3.5.

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, and 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 80186/88 APPLICATIONS

Your Satellite Emulator is configured for 80186/80188 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:		
68010	ES-68010B	ES-68010P
68000	ES-68000B	ES-68000P
68008		ES-68008P
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: $0^{\circ}C$ to $40^{\circ}C$ ($32^{\circ}F$ to $104^{\circ}F$)
Storage Temperature: $-40^{\circ}C$ to $70^{\circ}C$ ($-40^{\circ}F$ to $158^{\circ}F$)
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.).

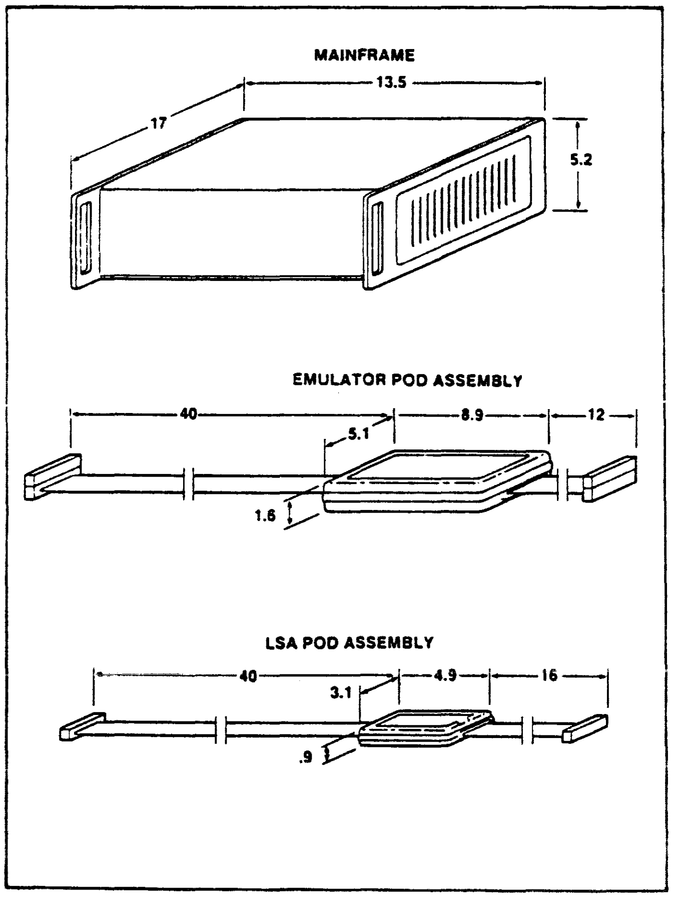


Figure 1-4. Dimensions

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 80186 or 80188 microprocessors
- the emulation control board
- the mainframe power cord
- the 80186 and 80188 Operator Manual
- Optional equipment you may have ordered:
  - Overlay Memory
  - Logic State Analysis pod assembly
  - Symbolic Debug
  - a carrying case

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 with 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.

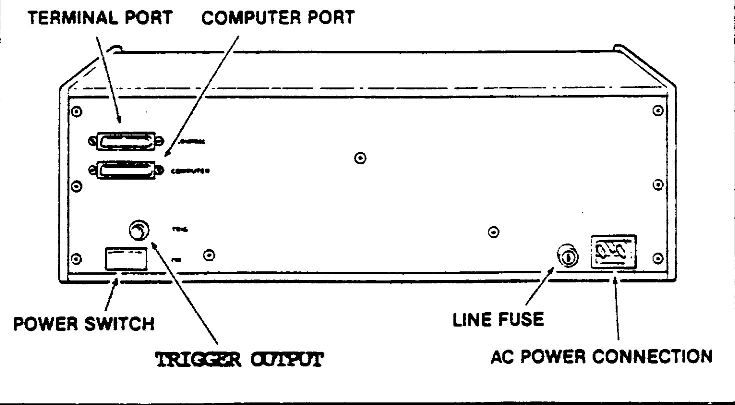


Figure 2-1.
Rear Panel

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.

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.

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 assigned pins 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.

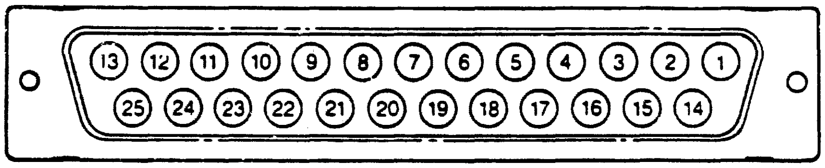


Figure 2-2. Serial Port Connector Pinout

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 6.2 of this manual.
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	

***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.

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.

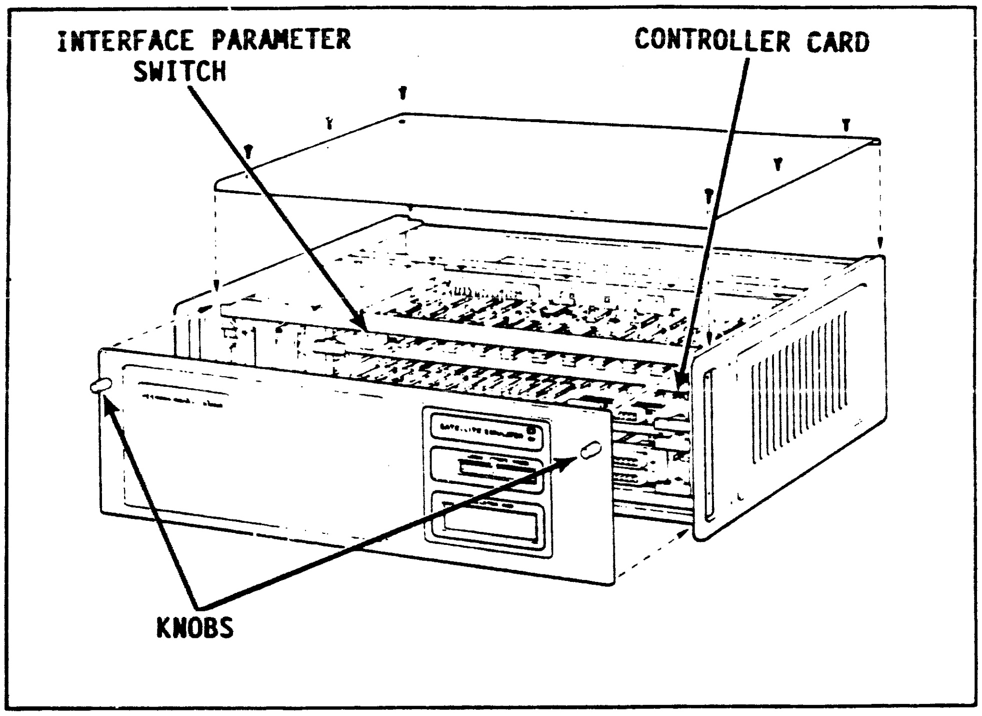


Figure 2-3. Top and Front Panel Removal

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 Applied Microsystems 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 the terminal 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.
Motorola:			
68000	ES-68000B	ES-68000P	Left
68008	ES-68008B	ES-68008P	Centered
68010	ES-68010B	ES-68010P	Centered
Zilog:			
ES-Z8000B	ES-Z8000P		Right
Z8001	"	ES-Z8001P	Right
Z8002	"	ES-Z8002P	Right
Z8003	"	ES-Z8003P	Right
Intel:			
8086	ES-8086B	ES-8086P-86	Centered
8088	"	ES-8088P-88	Centered
80186	"	ES-80186P-186	Centered
80188	"	ES-80188P-188	Centered

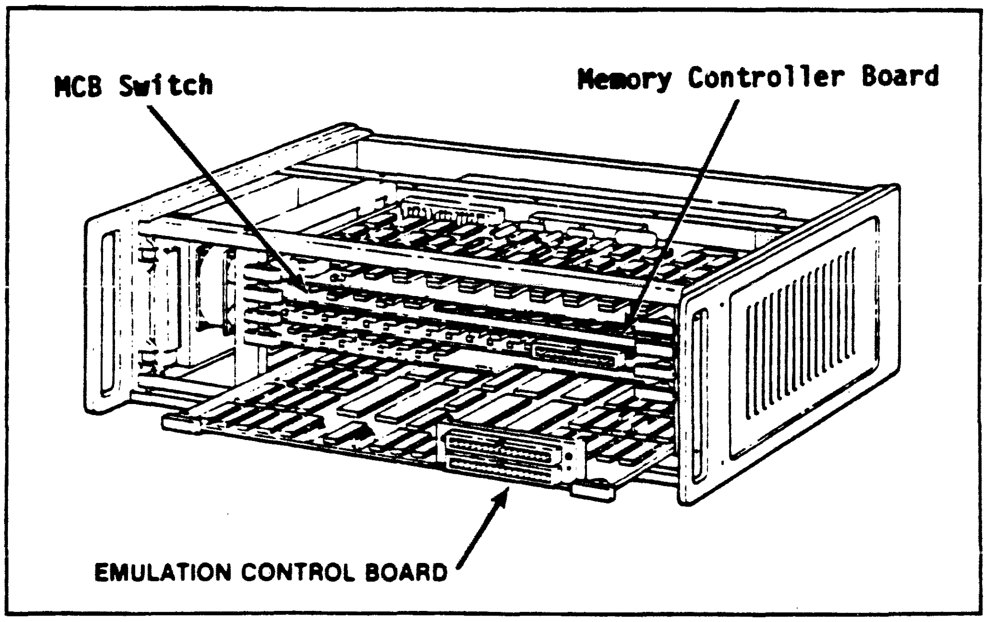


Figure 2-4. Installing the Emulator Control Board

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

CAUTION:

NOTE CORRECT PIN 1 ORIENTATION

5. The next section gives power-up procedures.

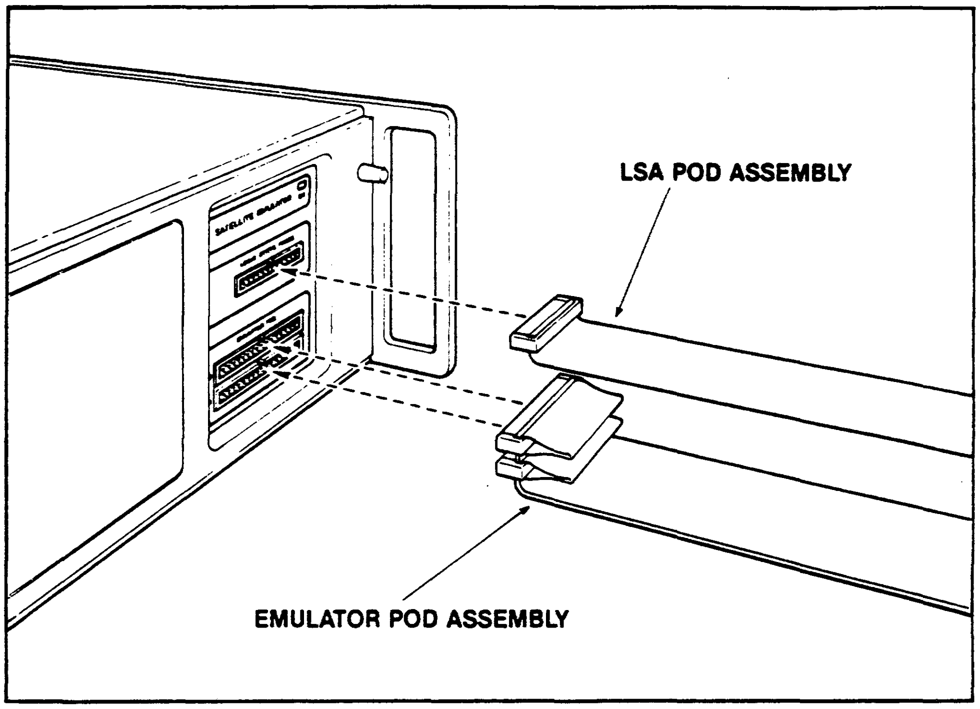


Figure 2-5.
Connecting the
Pod Assemblies
To the Mainframe

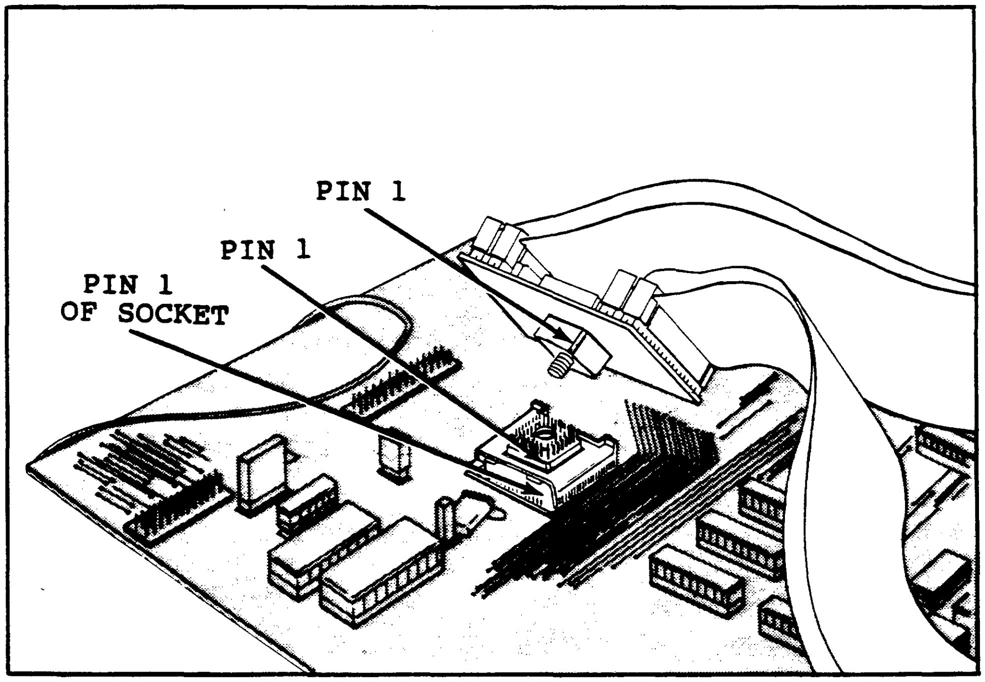


Figure 2-6.
Installing the
Probe Tip

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.

The first time you use your emulator, it must be powered up with the interface parameter switch in position Ø or positions 5 through B. To ensure the proper switch position, first check the baud rate on the terminal, then, if necessary, select the appropriate parameter switch setting so that the baud rate on the emulator correlates with the baud setting on the terminal. We recommend the highest rate possible for best response. You can later make and store final adjustments for the terminal/emulator interface by using the SET & SAVE commands.

NOTE

> If the interface parameter switch is in any of positions 1 through 4, the parameters and register values stored in the EEPROM are loaded on power-up. See Section 3.5 and Section 2.3.4.

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.

   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 3.5 and Section 2.3.4.

3. The emulator transmits the following message to the terminal:

   ```
   COPYRIGHT 1984
   APPLIED MICROSYSTEMS CORPORATION
   SATELLITE EMULATOR 80186 V2.4
   USER = 0   SW = 1
   ```

4. The processor type is that of the microprocessor installed in the pod. The version number reflects the released revision of the software in the emulator. The USER number and switch position, SW, are determined by the Interface Parameter Switch.

5. The emulator performs its self test which may take a couple of seconds. During this time the emulator determines the amount of overlay memory and transmits to the terminal:

   # nK AVAILABLE OVERLAY n=32, 64, 128, 256 or 512 depending upon amount of overlay memory installed in the system.

When there is no clock in the target system there will be a different message:

```
# nK AVAILABLE OVERLAY
NO CLOCK TYPE "Y" TO SELECT INTERNAL CLOCK
```

If you do elect to use the internal clock type a Y and the emulator will respond with a >.

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 may lose characters 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.

Before reviewing the pre-emulation checklist and character set, type "CLK" <return> just to verify that your target system clock is ok.

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 operating, you can call up the Help Menu by entering the question mark character, "?". 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 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

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

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

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

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

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

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

FIRST PAGE OF HELP MENU

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

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

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

STEP 1 - ASSIGN EVENT DETECTORS         STEP 2 - CREATE EVENT SPECIFICATIONS
AC1 = $1234;S1 = BYT + RM               WHEN AC1 AND S1 THEN GROUP 2
AC1.2 = $4576+14*6;DC2.2 = $56ØØ DC $FF 2 WHEN AC1 AND NOT DC2 THEN CNT
CTL.2 = 24;AC2.2 = $FØØØ LEN $4ØØ       WHEN CTL.2 OR AC2.2 THEN BRK
```

SECOND PAGE OF HELP MENU

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 RAM Overlay Memory, the on/off switches and macros.

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>

Figure 2-8. Display Format

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

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

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

TERMINAL:  #1Ø BAUD RATE = #14; [2=11Ø,5=3ØØ,1Ø=24ØØ,14=96ØØ]
           #11 STOP BITS = 1; [1,2]
           #12 PARITY = Ø; [Ø=NONE,1=EVEN,2=ODD]
           #13 CRT LENGTH = #24
           #14 TRANSPARENT MODE ESCAPE SEQUENCE = $1B,$1B

COMPUTER:  #2Ø BAUD RATE = #14; [7=12ØØ,12=48ØØ,15=192ØØ]
           #21 STOP BITS = 1
           #22 PARITY = Ø
           #23 TRANSPARENT MODE ESCAPE SEQUENCE = $1B,$1B
           #24 COMMAND TERMINATOR SEQUENCE = $ØD,$ØØ,$ØØ
           #25 UPLOAD RECORD LENGTH = #32; [1 to 127]
           #26 DATA FORMAT = 2; [Ø=INT,1=MOS,2=MOT,3=SIG,4=TEK,5=XTEK]
           #27 ACKNOWLEDGE CHAR = $Ø6
```

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 these 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

LD

SAV

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

- 0 = system set-up (defined via SET)
- 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.
- 5 = Macros

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, you must initialize the EEPROM to the proper data by executing 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 in the EEPROM. For example:

- To Load or Save all the groups, key in the Load and Save command.

```
example:  >LD
          >SAV
```

- 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.

```
example:  >LD1
          >SAV3
```

- 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.

```
example:  >LD2; LD4
          >SAV1; SAV3
```

Run With Vectors RNV

When you enter RNV, the code segment and instruction pointer are loaded to their starting values, all other registers are initialized to their default powerup values (F defined) and emulation is started <return>.

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.

- When you input a command to the emulator that it does not understand, it will respond with a "?".

Sample sequence for first-time emulation:

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

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.

- RAM 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 refer to specific sections when you need to use them. New users do not need to read all of the information in this chapter 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 language, 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 recognized 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 text. Some conventions have been adopted for ease of explanation.

- When an angle bracket <> encloses an expression, it is a 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.

: — A colon may be used as a separator for an 80186 family pointer type. For example CS:IP will convert the segment and offset to the absolute 20-bit address they represent (it is also a map type separator).

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.

3.3 NUMBERS AND BASE VALUES

There are three basic types of values used in the emulator: normal, Don´t Cares and ranges. The following paragraphs describe each of the values in detail.

o 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 such 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.

```
DC1 = $FFØØ DC $FF
```

- 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. Also, XRA and IRA can be prefixed to the arguments to define external or internal ranges, respectively. The default is IRA.

```
AC1 = Ø to $7FF
```

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 base 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.

3.3.3 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 modify the valve of 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

The following sections describe the properties of the group and the commands within each.

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

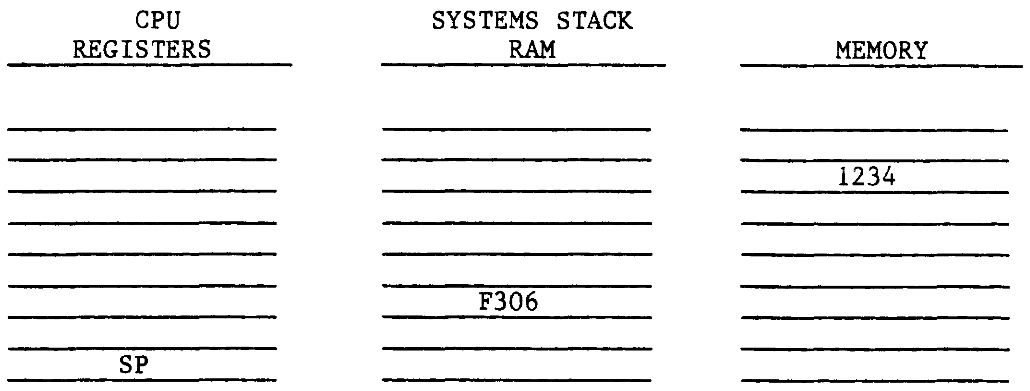
There is no set number of levels of parentheses that the Satellite Emulator can work with. The only limitation is that statements can be no more than 79 characters long. Whatever level of complexity you can handle within this limitation will be handled easily by the emulator.

@ **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. See the example below.

In this example the dual indirection is used to access a table of data that is pointed to by the system stack pointer.

```
>@@ SP
>1234
```



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 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.

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	DC	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	Normal	* / 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.

Multiplication and Addition

Here's an easy example:

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

GD4	General Purpose data register 4
=	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 = 0000100000000000

& **BITWISE AND.** Bitwise And operator (&) functions as a logical AND. The & operator infers the ANDing of the bits that form the two arguments.

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

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

Bitwise And
Bitwise Or

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

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

3.4.3 Single-Argument

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.

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 becmes 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

!

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 negative value would become positive, a positive value would remain unchanged.

3.5 PARAMETER SET-UP AND EEPROM STORAGE

As mentioned in the overview in Section 2.3.4, you may set system parameters with the SET command. These are listed in Table 3-5, which follows.

You can also set an additional 10 parameters using software switches. Nine 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.

```
ES SWITCH SETTINGS MENU

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

VALUE      NAME          DESCRIPTION

OFF        BKX           BREAK ON INSTRUCTION EXECUTION (NOT PREFETCH)
ON         CK            SELECT INTERNAL CLOCK
OFF        CPY           COPY DATA TO TERMINAL & COMPUTER PORTS
ON         FSX           FSI ON INSTRUCTION EXECUTION (NOT PREFETCH)
OFF        RDY           SELECT INTERNAL READY WHEN ACCESSING OVERLAY
--         DME           ENABLE DMA DURING PAUSE
--         TEØ           ENABLE TIMER Ø DURING PAUSE
--         TE1           ENABLE TIMER 1 DURING PAUSE
--         TE2           ENABLE TIMER 2 DURING PAUSE
--         STI           ENABLE STEP THROUGH INTERRUPTS
--         RCS           READ CHIP SELECTS ON RUN TO PAUSE
```

Figure 3-1.
ES Switch Settings

Software Switches

ON
OFF

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

- To display switches
  >ON
  or
  >OFF

- To turn on "break on instruction execution"
  >ON BKX

- To turn off all switches
  >OFF -1

- To turn on more than one switch
  >ON BKX + FSX + CPY + etc.

Break on Instruction Execution BKX

If BKX is ON, an event system break will occur on the instruction execution rather than the instruction pre-fetch.

CLOCK
CK

If clock is ON, the emulator uses an internal clock.

If clock is OFF, the target system clock is used.

FSI on Instruction Execution FSX

If FSX is ON, an FSI will occur on instruction execution rather than the instruction pre-fetch. Note that if FSX is off, the emulator may perform several FSIs as the instruction is prefetched several times.

Copy Switch **CPY**	The CPY switch allows the user to enter a copy mode which refers to the two ports on the back of the emulator. When the copy switch 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. When CPY is off, data is sent only to the controlling port.
Internal Ready Switch **RDY**	If the ready switch is on, the emulator will use an internally generated ready when accessing overlay memory. If the target system does not provide a ready signal for addresses that are mapped into overlay memory, this switch must be set on. The switch should be off when overlaying Dynamic RAM for targets using dynamic memory controllers that hold ready inactive during refresh.
Enable DMA **DME**	When DME is ON, DMA will be enabled during pause. When it is OFF, DMA will be disabled during pause.
Enable Pause Timer 0 **TEØ**	When TEØ is ON, Timer 0 will be enabled during pauses. When it is OFF, Timer Ø will be disabled during pause.
Enable Pause Timer 1 **TE1**	When TE1 is ON, Timer 1 will be enabled during pause. When it is OFF, Timer 1 will be disabled during pause.
Enable Pause Timer 2 **TE2**	When TE2 is ON, Timer 2 will be enabled during pause. When it is OFF, Timer 2 will be disabled during pause.
Step Through Interrupts **STI**	When STI is ON, the user may step through interrupts. If it is OFF, interrupts will not be serviced while stepping.
Chip Select **RCS**	When RCS is ON, the emulator will read the Chip-Select Control Registers upon entering PAUSE. This should be turned on only after the Chip-Select Control Registers have been set because reading enables them. RCS applies to the group of Chip-Select Control Registers except UMCS. By writing to an individual Chip-Select Control Register during PAUSE, you can enable reading it. For example, LMCS = $8000. Note: Pause means that the emulator is not in one of the RUN modes.

Table 3-5.
SET Select Numbers

KEY SEQUENCE	DESCRIPTION/RESULT	RESET CHARACTER REQUIRED
SET #1,#0<return>	Select User 0	No
SET #1,#1<return>	Select User 1	No
SET #2,$n	n is the desired reset character	No
SET #3,$n,$m<return>	Set values for X ON (n) and X OFF (m)	No
SET #10,#0<return>	Set CRT terminal baud rate to 50	Yes
SET #10,#1<return>	75 baud (CRT terminal)	Yes
SET #10,#2<return>	110 baud (CRT terminal)	Yes
SET #10,#3<return>	134.5 baud (CRT terminal)	Yes
SET #10,#4<return>	150 baud (CRT terminal)	Yes
SET #10,#5<return>	300 baud (CRT terminal)	Yes
SET #10,#6<return>	600 baud (CRT terminal)	Yes
SET #10,#7<return>	1,200 baud (CRT terminal)	Yes
SET #10,#8<return>	1,800 baud (CRT terminal)	Yes
SET #10,#9<return>	2,000 baud (CRT terminal)	Yes
SET #10,#10<return>	2,400 baud (CRT terminal)	Yes
SET #10,#11<return>	3,600 baud (CRT terminal)	Yes
SET #10,#12<return>	4,800 baud (CRT terminal)	Yes
SET #10,#13<return>	7,200 baud (CRT terminal)	Yes
SET #10,#14<return>	9,600 baud (CRT terminal)	Yes
SET #10,#15<return>	19,200 baud (CRT terminal)	Yes
SET #11,#1<return>	CRT terminal data frame has 1 stop bit	Yes
SET #11,#2<return>	2 stop bits (CRT terminal)	Yes
SET #12,#0<return>	CRT terminal parity (send and receive) none	Yes
SET #12,#1<return>	Parity even (CRT terminal)	Yes
SET #12,#2<return>	Parity odd (CRT terminal)	Yes
SET #13,#n<return>	Set CRT terminal lines per page; n=5 to 255	No
SET #14,$n<return>	Specify a 7-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 see 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. See SET #12,n above	Yes
SET #23,$n,$m<return>	Set computer transparent mode escape characters. See SET #15,n,m	No

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

Example 3-4 shows how to Load or Save system 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.

SECTION 4
OPERATION

4.1 INTRODUCTION

This section describes the procedures for operating the Satellite Emulator and error codes that may occur. 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 80186 and the emulator, and to display these values.

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
CTL	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 enable register	Memory Mode
MMP	memory space pointer	Memory Mode
MMS	Memory Mode access status register	Memory Mode
IOP	IO space pointer	IO Mode
GDØ-7	general purpose data register	Miscellaneous
GRØ-7	general purpose range register	Miscellaneous
DFB	default base register	Miscellaneous
AX, AL, AH	accumulator (low and high)	CPU Registers
BP	base pointer	CPU Registers
BX, BL, BH	base register (low and high)	CPU Registers
CS	code segment	CPU Registers
CX, CL, CH	count register (low and high)	CPU Registers
DS	data segment	CPU Registers
DI	destination index	CPU Registers
DX, DL, DH	data register (low and high)	CPU Registers
ES	extra segment	CPU Registers
FLX, FLL, FLH	flags register (low and high)	CPU Registers
IP	instruction pointer	CPU Registers
SI	source index	CPU Registers
SP	stack pointer	CPU Registers
SS	stack segment	CPU Registers
CW0	dma 0 control word	PCB Registers
CW1	dma 1 control word	PCB Registers

DST0	dma 0 lower 16 bits of destination address	PCB Registers
DST1	dma 1 lower 16 bits of destination address	PCB Registers
EOI	end of interrupt register	PCB Registers
LMCS	lower memory chip select register	PCB Registers
MA0	timer 0 max count "A" register	PCB Registers
MA1	timer 1 max count "A" register	PCB Registers
MA2	timer 2 max count "A" register	PCB Registers
MB0	timer 0 max count "B" register	PCB Registers
MB1	timer 1 max count "B" register	PCB Registers
MCW0	timer 0 mode control word register	PCB Registers
MCW1	timer 1 mode control word register	PCB Registers
MCW2	timer 2 mode control word register	PCB Registers
MMCS	mid-range memory chip select (base address) register	PCB Registers
MPCS	mid-range memory chip select (block size) register	PCB Registers
PACS	peripherial chip select register	PCB Registers
REL	relocation register	PCB Registers
SRC0	dma 0 lower 16 bits of source address	PCB Registers
SRC1	dma 1 lower 16 bits of source address	PCB Registers
TC0	timer 0 count register	PCB Registers
TC1	timer 1 count register	PCB Registers
TC2	timer 2 count register	PCB Registers
UDST0	dma 0 upper 4 bits of destination address	PCB Registers
UDST1	dma 1 upper 4 bits of destination address	PCB Registers
UMCS	upper memory chip select register	PCB Registers
USRC0	dma 0 upper 4 bits of source address	PCB Registers
USRC1	dma 1 upper 4 bits of source address	PCB Registers
XC0	dma 0 transfer count	PCB Registers
XC1	dma 1 transfer count	PCB Registers

The following interrupt controller registers are used only in iRMX mode

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

The following interrupt controller registers are used in non-iRMX mode

DMA0	dma 0 interrupt control register	PCB Registers
DMA1	dma 1 interrupt control register	PCB Registers
INT0	interrupt 0 control register	PCB Registers
INT1	interrupt 1 control register	PCB Registers
INT2	interrupt 2 control register	PCB Registers
INT3	interrupt 3 control register	PCB Registers
IRQ	interrupt request register	PCB Registers
IST	interrupt status register	PCB Registers
ISV	in service register	PCB Registers
MSK	mask register	PCB Registers

PLM	priority mask register	PCB Registers
POL	poll register	PCB Registers
POS	poll status register	PCB Registers
TCR	timer interrupt control register	PCB Registers

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>

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

Figure 4-1. Display Registers Format

4.2.1 Loading A Register

Example:

```
>BX<return>
$00000000

>BX=5000<return>

BX<return>
$00005000
```

4.2.1.1 Peripheral Control Block (PCB) Registers

Because of the dynamic nature of some PCB registers, these are handled slightly differently than regular CPU registers (such as AX, BX, etc.). (Exceptions to the following procedure are also described below.)

4.2.1.1.1 General PCB Handling

When the emulator exits the RUN mode, all memory and I/O space is searched for the PCB.

- When the PCB is located, it is moved to locations $FF00-$FFFF in I/O space.

- All register values are then copied to a table in internal RAM and uploaded to the ES controller. These register values are the ones displayed in response to the PCB command.

  The values in this table are modified by commands such as:

  MCWO-$1234

  or

  IST=$5678

The commands do not modify the current contents of the physical PCB until the next PAUSE-TO-RUN transition.

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

4.2.1.1.2 Exceptions

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

Timers

The ON/OFF switches TEØ, TE1 and TE2 are used to selectively enable/disable the integrated timers during PAUSE mode.

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

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

DMA Controllers

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

- If the switch is set to ON DME:

  -The IST register is copied to the RAM table
  -The DHLT bit is then cleared, causing DMA cycles to resume. All DMA cycles are steered to the target system.
  -Upon PAUSE to RUN transition, the RAM table value of the IST register will be reloaded to the physical PCB. If you want DMA activity to continue when reentering RUN mode, the "RUN" command must be preceeded by: "IST = IST & $7FFF" to clear the DHLT bit of the RAM table´s IST register value.
  -No DMA register values are reloaded to the physical PCB with this setting.

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

Chip Select Registers

The ON/OFF RCS switch controls the emulator´s reading of the LMCS, MMCS, MPCS, and PACS registers upon RUN to PAUSE transition.

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

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

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

Interrupt Controller Registers

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

Because these two registers are Read Only, they are not reloaded to the physical PCB upon PAUSE to RUN transition.

4.2.2 General Registers

GD0-7 and GR0-7 are miscellaneous registers used to save keystrokes 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

Example:

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

>MEMORY MAP:
>MAP $000000 to $000FFF : TGT
>MAP $001000 to $002FFF : RW
>MAP $003000 to $FFFFFF : TGT
```

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 (pod only)

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.

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. Use STP, /, /, /, ... instead of STP, STP, STP, ... it saves typing.

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 promp R> will be present during RBK.

These examples show how to start emulation:

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 a Run With Breakpoints:
  R>STP<return>
- To Single Step through instructions (emulation not currently in progress):
  >STP<return>

4.3.4 Vector Loading and Running With Vectors

The 80186 microprocessor, when reset, loads the IP (instruction pointer) to $0000 and the CS (code segment) to FFFF. The emulator can automatically load these same values before going to a run. Three commands implement these functions; LDV initializes the registers, RNV initializes the registers and begins run, RBV initializes the registers and starts run with breakpoints enabled. See the following examples.

LDV

- To initialize registers:
  >LDV<return>

RNV
RBV

- To initialize registers and run:
  >RNV<return>
  or
  >LDV;RUN<return>

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

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

  After an LDV, the registers contain the following data:

CS = $FFFF	IP = $ØØØØ	FLX = $FØØ2	DS = $ØØØØ
ES = $ØØØØ	SS = $ØØØØ	REL = $2ØFF	UMCs = $FFFB
CWØ = $ØØØØ	CW1 = $ØØØØ	MCWØ = $ØØØØ	MCW1 = $ØØØØ
MCW2 = $ØØØØ	INTØ = $ØØØF	INT1 = $ØØØF	INT2 = $ØØØF
INT3 = $ØØØF	TCR = $ØØØF	DMAØ = $ØØØF	DMA1 = $ØØØF
MSK = $ØØFF	PLM = $ØØØ7	ISV = $ØØØØ	IRQ = $ØØØØ

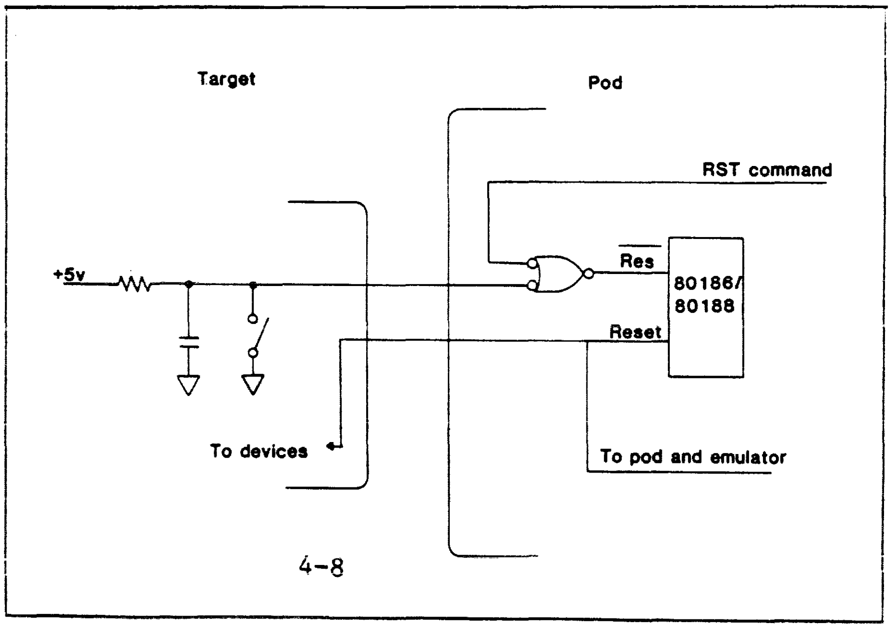
4.3.5 Reset RST

Reset is used to reset the microprocessor pod and emulator board and also the target. (On all other EM/ES emulators RST does not reset the target.) The 80186/88 has two reset pins on it (see diagram):

- /Res (pin 24) is an input to the microprocessor and can be driven from both the target and the RST command

- Reset (pin 57) is an output from the microprocessor and is sent back to the target

When an RST command comes from the controller board, it resets the microprocessor and the target hardware as well. One exception to this is when the /res pin goes to devices other than the microprocessor, these devices will not get reset from the RST command.



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;BX<return>

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

NOTE

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.

4.4 MEMORY MODE IO MODE

Memory Mode allows you to examine or change the contents of the target system memory. IO mode allows you to examine or write data to IO ports. 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, word and byte modes, and how to examine and change the target system memory.

4.4.1 Entering and Exiting Memory Mode M or MM X MIO

M or MM 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.

MIO is used to enter I/O mode. If no address is specified, the address will default to the value of IOP. Upon entering this mode, the ports are not read. The address is displayed preceeding the prompt. To read the port, execute a <return> as the only character on the line. The port will be read and the address and data displayed. The address will not be incremented unless a "." is entered. Refer to the following example.

- 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>

- To change address while in memory mode:
  $00000000 $FFFF >M <address><return>

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

  - byte mode $000000 $FF>
  - word mode $000000 $FFFF>

- To enter I/O space at a specific address:
  >MIO<address><return>

- To enter I/O space at the default address:
  >MIO<return>

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

4.4.2 Memory Mode and Pointers MMP IOP

The Memory and I/O mode pointers, when invoked, will display the last address invoked in Memory or I/O Mode since power-up. The key sequence is shown in the following example.

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

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

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

- To display the last I/O address examined:
  >IOP <return>

- To change the I/O pointer:
  >IOP = <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:
  <address><data> > NXT <return>
  or
  <address><data> >.
  or
  <address><data> ><return>

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

IO Examples:

- Scroll to next address:
  <address> >

- Display Data:
  <address> ><return>
  <address><data> >

- Scroll to lower address:
  <address> >,

4.4.4 Word and Byte Modes BYM WDM

If you wish to scroll through the memory spaces one byte (8 bits) at a time, invoke BYM. WDM is used to scroll in the word mode (16 bits). The system will default to a byte mode.

BYM and WDM should be considered global defaults that affect all operations, not just Memory Mode.

- To scroll in the byte mode:
  > BYM

- To return to the word mode:
  > WDM

4.4.5 Examining and Changing Values

Now that you can access Memory and I/O Modes, work with the pointers and scroll higher and lower in either the byte or 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 new address (and its value if in memory mode). 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 when in memory mode. In I/O mode a string of values will be stored at the same address. The maximum length for a string of values is 9.

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

- To change a series of values at consecutive locations:
  >M<address> <return>
  The emulator will respond with:
  <address> <current data>>
  Then enter:
  <address><data> >$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 Displaying a Block of Memory and Finding a Memory Pattern DB

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

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

- To display a block of memory:
  >DB <address range> <return>

- To display a page of memory starting from a specified address:
  >DB<address><return>

- To display a page of memory starting from the ending address of the previous DB command:
  >DB<return>

- To easily scroll through memory use the slash "/" following a **DB<return> command.**

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 <range>, <data> <return>

- To find a bit pattern using Don't Cares (either form):
  FIN 1000 TO 2FFF, 60XX
  or
  FIN 1000 LEN 1000, 6000 DC OFF

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

```
 >BYM
 >DB Ø LEN 3Ø
ØØØØØØ  8Ø 48 45 4C 4C 4F 8Ø 8Ø - 2F ØF F1 F9 5E 2F F6 FØ .HELLO../...^/..
ØØØØ1Ø  ØF Ø3 FØ 4Ø ØF ØC FØ 4Ø - Ø7 Ø6 FØ 9Ø ØF ØC D8 ØØ ...@...@........
ØØØØ2Ø  FF ØF FF F9 FF 1F FF 7F - FF 3F FF BD FF 1F FF FF .........?......
 >
 >WDM
 >DB Ø LEN 3Ø
ØØØØØØ    488Ø  4C45  4F4C  8Ø8Ø - ØF2F  F9F1  2F5E  FØF6
ØØØØ1Ø    Ø3ØF  4ØFØ  ØCØF  4ØFØ - Ø6Ø7  9ØFØ  ØCØF  ØØD8
ØØØØ2Ø    ØFFF  F9FF  1FFF  7FFF - 3FFF  BDFF  1FFF  FFFF
 >
 >
```

Figure 4-2.
Display Memory
Block Format

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 need to define the memory map first.

The overlay memory is available to the target for DMA only when using the ON-chip DMA controller. If an external DMA scheme is used, it cannot access the overlay. The overlay memory is accessible to the target system whenever it is mapped. An exception to this rule is DMA. During External DMA or other non-CPU activity times, the Overlay Memory is not available to the target. (Internal DMA within the 80186/188 still has use of the Overlay.)

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 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.

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 MAP Operator is used in conjunction with the memory type operators (Read Only, Read/Write, etc.).

- 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>

```
 >MAP Ø:RO
 >DM
MEMORY MAP:
MAP $ØØØØØØ TO $ØØØ7FF : RO
MAP $ØØØ8ØØ TO $FFFFFF : TGT
```

Figure 4-3. Display Memory Map Format

4.5.3 Overlay Memory Operators

OVE
DTA
CD

Overlay Memory **ENABLE.** The OVE operator allows you to load values that determine which memory status space the Overlay Memory responds to. The possibilities are Code Space (CD) and Data Space (DTA). The current value is shown when the memory map is displayed. Factory default is CD + DTA.

- The general format for OVE is:
  >OVE = CD + DTA <return>

Load Overlay Memory LOV

LOV. LOV loads Overlay 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.

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 X TO Y, Z
      or
>FIL 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 or word, depending on the global default.

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 <address range>, <argument><return>
```

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

Block Move Verify Block Move BMO VBM

The Block Move operator moves a block of data from one location within the target system memory or Overlay Memory to another via a source/ destination format. The space you move data into should be designated as writeable.

The format for a block move is:

```
>BMO<source range>
<destination start address>,<return>
>GR1=ØLEN2ØØ
>BMO GR1, $ØØØ
```

- You can verify byte data between the target and Overlay Memory:

```
>VBM 2000 TO 3FFF,2010
```

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 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 by using an internally generated clock.

The procedure consists of mapping memory space and loading the Overlay Memory with your program. You can now use the features of the emulator to execute program code and test modules.

4.7 ERROR HANDLING AND CODES

When an error occurs during operation of the Satellite Emulator with a CRT terminal, the system will print a question mark (?) on the screen, directly below or just after the point in the input character stream that caused the error. See Figure 4-6.

Figure 4-6. Error Recognition

```
 >HELLO
       ?
 >?
ERROR #5
UNDEFINED SYMBOL OR CHARACTER DETECTED
 >
 >
 >SET   GDØ=Ø LEN 6Ø
               ?
 >?
ERROR #29
ILLEGAL DESTINATION - SOURCE TYPE MIX
 >
 >
 >
 >
 >MAP Ø TO ØFFFFFF
                   ?
 >?
ERROR #9
NO (MORE) OVERLAY RAM AVAILABLE
```

Table 4-2. Error Codes

CODE	MESSAGE DISPLAYED	COMMENTS
1	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.
4	PARSE ERROR	Call Customer Service
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 Applied Microsystems Technical Services Department.

8	ARGUMENT IS NOT A SIMPLE INTEGER OR INTERNAL RANGE	Don´t Cares are not allowed in this context.
9	NO (MORE) OVERLAY RAM AVAILABLE	You have not cleared the map or you are trying to map in more memory than is allowed.*
10	MULTIPLY-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 AC1.3 THEN BRK would cause this error.
11	ILLEGAL ARGUMENT TYPE FOR EVENT SPECIFICATION	
13	ARGUMENT MUST BE A SIMPLE INTEGER	
14-16	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)	
20-23	OPERATION INVALID FOR THESE ARGUMENT TYPES	See error 14-16.
24	BASE ARGUMENT MUST BE A SIMPLE INTEGER	Argument should be #0 to #16.
25	´DON´T CARES´ NOT ALLOWED IN I/O DATA	
26	RANGE TYPE ARGUMENT NOT ALLOWED AS DATA	
27	ADDRESS ARGUMENT MUST BE A SIMPLE INTEGER	
28	ILLEGAL MODE SWITCH ATTEMPT.	Can´t enter I/O mode from memory mode ad vice versa.
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	
33	RANGE START AND END ARGUMENTS MUST BE SIMPLE INTEGERS	6 LEN 1 is not a valid range.

*Contact Applied Microsystems for optional Overlay Memory expansion.

CODE	MESSAGE DISPLAYED	COMMENTS
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.
36-38	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."
44	INTERNAL ERROR...NULL SHIFTER FILE	Contact Applied Microsystems.
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	

CODE	MESSAGE DISPLAYED	COMMENTS
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 Applied Microsystems for service.
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.
65	ILLEGAL RANGE	
66	STATUS CONSTANTS CANNOT BE ALTERED	
67	TOO MANY "WHEN" CLAUSES	
68	INVALID DATA FORMAT FOR SYMBOLS	Symbols must be up/down-loaded using Extended Telchex format. The format specified in the setup menu is not Extended Telchex.
70	CANNOT INITIALIZE VECTORS DURING EMULATION	Typed LDV, RNV, RBV during emulation.
71	UNKNOWN EMULATOR ERROR	Call Applied Microsystems.
72	INCOMPATIBLE EEPROM DATA	Previous data save was not from 8086 or 80186 emulator system.
74	COMMAND INVALID DURING EMULATION	

CODE	MESSAGE DISPLAYED	COMMENTS
75	INVALID RECORD TYPE	Download routine received invalid record type code.
76	NO SYMBOLIC DEBUG	
77	UNDEFINED SYMBOL OR CHARACTER DETECTED	
78-80	TOO MANY SYMBOLS	
81	SYMBOL OR SECTION PREVIOUSLY DEFINED	The ES will load all symbols preceeding the duplicate symbol into its symbolic RAM. When you then type the command SYM, these will all be displayed in the order that they were downloaded. The symbol on the end of the list can be found in your download file, and the symbol immediately following it will be the one that caused the error.
82	SYMBOL NAME IN USE	
83	TYPE CONFLICT WITH DEFINED SYMBOL	
87	SECTION TABLE FULL	
88	INVALID ARGUMENT SIZE	
89	INVALID ADDRESSING MODE	
90	ARGUMENT OUT OF RANGE	
94	ARGUMENT NOT SYMBOLIC	
--	NO BUS CYCLES	No ALE´S were detected for at least 2.2 ms or longer. The controller checks then for other fault conditions: checking to see if the CPU is halted, waiting, reset is asserted or if the bus has been granted. If any of these other conditions exist then the appropriate message for that condition would be displayed. If no other fault condition is found then ´NO BUS CYCLES´ will be displayed.
--	SYSTEM RESET ERROR	When a reset has been asserted from the controller and the emulator does not report back that a reset has been asserted then this message will be displayed (also when reset has been deasserted and the emulator does not report back to the controller that the reset has been deasserted).

CODE	MESSAGE DISPLAYED	COMMENTS
--	POD CPU NOT INITIALIZED	Whenever a reset occurs (power up, ^Z,RST), the controller and the emulator will begin an initialization routine to establish communication. If this initialization routine fails to complete, this message will be displayed.
--	POD CPU NOT RESPONDING	Whenever a STP, MM XXXX, or MIO XXXX is executed the controller will check to see if an error occurred during the command. If no error occurred then it looks to see if the command completed. If the command did not complete then the controller looks to see if the CPU is still running. If the CPU is not running, an appropriate error message will be displayed. If the CPU is running, this message is displayed.
--	RESET ASSERTED	This indicates that a reset from a target system has been asserted for greater than 1/2 sec. When the reset is deasserted then the message will be removed. However, if the reset is less than 1/2 sec., then the message will not be displayed. This message is intended as an error reporting condition.
--	NO TARGET POWER	If the +5v supply in the target system is at approximately +4v or less then this message will be displayed. The message will be removed when the fault condition has been corrected.
--	PROCESSOR WAITING	This means that the CPU is waiting for a RDY (either ARDY, SRDY, or internal rdy) to come back. This message will be displayed only if the cpu has been waiting for greater than 2.2ms. When the condition has been corrected then the message will be removed.
--	HOLD ACKNOWLEDGE ASSERTED/BUS GRANTED	This message will be displayed when a hold acknowledge has been asserted for greater than 2.2ms. This message will be removed when the CPU is given back the bus.

CODE	MESSAGE DISPLAYED	COMMENTS
--	PROCESSOR HALTED	A HLT instruction has been executed and has been asserted for greater than 2.2ms. This message will be removed when the CPU is given back the bus.
--	NO CLOCK	The CPU clock must have a period of 1Øus or faster to prevent this message from being displayed. The message will be removed when this error condition has been corrected.

4.8 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 signals of the last series of cycles can be displayed and scrolled 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.

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 the event monitor system (only available during a Run mode). (See Section 5).

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. Figures 4-4 and 4-5 show printouts of the Trace Memory and of the Trace Memory with Disassembly. When printed, the disassembler writes over the line which invokes it. This may cause overstriking on your printout.

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.

4.8.1 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 most current 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.

```
>DRT #5Ø
LINE  ADDRESS    DATA  R/W        M/IO  BCYC          QUE  LSA   -   8 7 Ø
 #69  ØØ1ØØØ   > ØFB9   R   OVL    M    IF           F Ø  %11111111 %11
 #68  ØØ1ØØ2   > BEØØ   R   OVL    M    IF             2  %11111111 %11
 #67  ØØ1ØØ4   > 2ØØØ   R   OVL    M    IF             2  %11111111 %11
 #66  ØØ1ØØ6   > ØØBF   R   OVL    M    IF             1  %11111111 %11
 #65  ØØ1ØØ8   > A522   R   OVL    M    IF             2  %11111111 %11
 #64  ØØ1ØØA   > A4F3   R   OVL    M    IF             2  %11111111 %11
 #63  ØØ1ØØC   > 81Ø3   R   OVL    M    IF             3  %11111111 %11
 #62  ØØ2ØØØ   > FF5Ø   R   OVL    M    RM             4  %11111111 %11
 #61  ØØ22ØØ <  FF5Ø   W   OVL    M    WM             4  %11111111 %11
 #6Ø  ØØ1ØØE   > FFØØ   R   OVL    M    IF             3  %11111111 %11
 #59  ØØ1Ø1Ø   > Ø2B9   R   OVL    M    IF             5  %11111111 %11
 #58  ØØ2ØØ2   >   3E   R   OVL    M    RM             6  %11111111 %11
 #57  ØØ22Ø2 <     3E   W   OVL    M    WM             6  %11111111 %11
 #56  ØØ2ØØ3   > FF     R   OVL    M    RM             6  %11111111 %11
 #55  ØØ22Ø3 <  FF      W   OVL    M    WM             6  %11111111 %11
 #54  ØØ2ØØ4   >   ØØ   R   OVL    M    RM             6  %11111111 %11
 #53  ØØ22Ø4 <     ØØ   W   OVL    M    WM             6  %11111111 %11
 #52  ØØ2ØØ5   > ØØ     R   OVL    M    RM             6  %11111111 %11
 #51  ØØ22Ø5 <  ØØ      W   OVL    M    WM             6  %11111111 %11
 #5Ø  ØØ2ØØ6   >   FF   R   OVL    M    RM             6  %11111111 %11
```

Figure 4-4.
Trace Memory Format

4.8.2 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 line feed 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 Ø 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>

NOTE

When using Display Register (DR) and Disassemble Trace (DT) on the same line, enter DT first, then DR, as shown in this example:

RBV;WAI;DT;DR<cr>

If you enter DR first, the DT command writes over the last line of the register display.

4.8.3 Disassemble Previous and Following Trace DTB DTF

These two operators will scroll you through the disassembled Trace Memory a page at a time. This is shown in Figure 4-5.

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

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

```
 >DTE
SEQ#  ADDR   OPCODE MNEMONIC  OPERAND FIELDS   BUS CYCLE DATA
-----------------------------------------------------------------------------
0069 PGM_8086_80186_Test
0069 1000  B90F00       MOV      CX,000F
0068 1003  BE0020       MOV      SI,2000
0066 1006  BF0022       MOV      DI,2200
0065 1009  A5           MOVS     WORD PTR    2000>FFF0  2200<FFF0
0064 100A  F3           REPZ
0064 100B  A4           MOVS     BYTE PTR
                        2002>3E  2202<3E  2003>FF  2203<FF  2004>00  2204<0
                        2005>00  2205<00  2006>FF  2206<FF  2007>FF  2207<F
                        2008>00  2208<00  2009>00  2209<00  200A>FF  220A<F
                        200B>FF  220B<FF  200C>00  220C<00  200D>00  220D<0
                        200E>FF  220E<FF  200F>F5  220F<F5  2010>00  2210<C
0063 100C  038100FF     ADD      AX,WORD PTR [BX-100][DI]  2111>FF00
0059 1010  B90200       MOV      CX,0002
0026 1013  F2           REPNZ
0025 1014  A7           CMPS     WORD PTR    2011>FF10  2211>FF10
0025 1015  C116002405   RCL      WORD PTR Data_Word,05  2400>A002  2400<004A
0017 101A  C8400004     ENTER    0040,04
                        17FE<0000  FFFE>FFFF  17FC<FFFF  FFFC>FFFF
                        17FA<FFFF  FFFA>FFFF  17F8<FFFF  17F6<17FE
0015 101E  E0E0         LOOPNE   SHORT PGM_8086_80186_Test
```

Figure 4-5. Disassemble Trace Format

When the question mark appears on the screen, you should key in a question mark in return to find out the error message. Table 4-2 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.

4.9 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 8086/80186 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 Intel documentation. You should have the iAPX 86, 88, 186 and 188 User´s Manual Programmer´s Reference.

4.9.1 Display Disassembled Memory DIS

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

Example Using 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)

4.10 THE LINE ASSEMBLER

The 8086/80186 Line Assembler allows you to enter and assemble Intel 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.

4.10.1 Standard Mnemonics

All standard Intel 8086 and 80186 mnemonics are supported. These are listed in the ASM86 Language Reference Manual.

4.10.2 Assembler Directives

The following assembler directives are supported:

DIRECTIVE	DESCRIPTION
CSEG	Sets 64K byte code segment window (corresponds to CS register).
ORG	Sets 64K byte offset into the code segment window.
END	Exits Line Assembler to the command level.
DB	Defines byte data.
DW	Defines word data.
PRE	Toggles preview display mode.
EQU	Sets value for symbol (only valid with installed symbolic debug hardware) or local symbol (LØ-L9).
LØ,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 offset address.
THIS NEAR	Current line assembly offset address.
THIS FAR	Current line assembly segment and offset address.

The key sequences for these assembler directives are shown in example 5-6.

NOTE:

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

Example Use of Assembler Directives

- To set Code Segment window (64K-byte assembly window):

```
1Ø12 >CSEG OD4ØØH<return>
1Ø12 >
```

- To set line assembly origin within code segment window:

```
1Ø12 >ORG 38ACH<return>
38AC >
```

- To exit line assembly:
```
58FD >X<return>
**** END OF LINE ASSEMBLY ****
>
```

- To define constant byte data:
```
58FD >DB 1,2,3,4, "TEST",Ø<return>
58FD Ø1 Ø2 Ø3 Ø4 54 45 53 54 ØØ
59Ø6 >
```

- To define constant word data:
  (Note: odd length text strings are padded with nulls)
```
58FD >DW 1,2,3,4,"TEST",Ø<return>
58FD Ø1ØØ Ø2ØØ Ø3ØØ Ø4ØØ 4554 5453 ØØØØ
59ØB >
```

- To toggle to preview mode:
```
659Ø >PRE <return>
659Ø C647Ø234 MOV BYTE PTR [BX+2H],34H
```

- To toggle out of previous mode:
```
659Ø C647Ø234  MOV BYTE PTR [BX+2H],34H >PRE<return>
659Ø >
```

- To define/redefine local symbol or symbolic (if symbolic debug hardware is installed)"
```
659Ø >L3 EQU 7A44H<return>
659Ø >
```

  or if symbolic debug hardware is installed:

```
659Ø > ´Unit EQU ØFDEOH<return>
659Ø >
```

- To print local symbol:
```
756A >L3<return>
756A >L3 EQU 7A44H
756A >
```

- To print symbol (if symbolic debug hardware is installed):
```
756A >´Unit <return>
756A >´Unit EQU FDEOH
756A >
```

- To disassemble one instruction at current code segment and line assembly offset address:
```
5DØA ><return>
5DØA 33Ø6AD78     XOR AX,WORD PTR 78ADH
5DOE >
```

4.10.3 Usage Notes

+ - * / Plus, minus, asterisk, or slash are the only arithmetic operators allowed in expressions. Note that only 16 bit arithmetic is performed.

( ) Parenthesis are allowed to group expressions.

" %	Double quotes (") or percent signs (%) are used to delimit ASCII strings. If you enclose the string in percent signs, you may not use percent signs 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 percent signs may be used.
"´<bs>	Upper-case strings are the default. The use of "´<backspace> will allow entry of lower-case letters within a text string until you enter a <space>.
H O Q T Y	The number base used in the line assembler is the default base used by the system, except when the base is explicitly specified (H, O, Q, T, or Y).

H - Hexidecimal
O - Octal
Q - Octal
T - Decimal
Y - Binary

Example operand addressing modes:

ØA4H	Immediate addressing mode
WORD PTR 5634H	Direct addressing mode
BYTE PTR 9DC4H	Direct addressing mode
[SI]	Indexed addressing mode
[BX][SI]	Base indexed addressing mode
[BX+SI]	Base indexed addressing mode
[BX+5]	Base Displacement addressing mode
[BP+4]	Stack indexed addressing mode
[BX+DI+15]	Base displacement indexed
[-3*(23+4)+BX+SI]	Base displacement indexed

4.10.4 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 eiher END or X with the addressed prompt displayed (as shown here).

Example Use of ASM

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

```
>ASM<return>
**** 8Ø86/88/186/188 LINE ASSEMBLER VX.XLA ****

ØØØØ >
```

- To start line assembly beginning at the specified code segment and line assembly offset address:

```
>ASM ØC6A3<return>
**** 8Ø86/88/186/188 LINE ASSEMBLER VX.XLA ****
```

• To terminate line assembly:

```
9876 >X<return>
**** END OF LINE ASSEMBLY ****
>
```

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

Example
Using Addresses

```
>ASM 1ØØ<return>
**** 8Ø86/88/186/188 LINE ASSEMBLER VX.XLA ****

Ø1ØØ >CSEG 5ØØØ<return>
Ø1ØØ >MOV DX,8<return>
Ø1ØØ BAØ8ØØ      MOV DX,8
Ø1Ø3  89Ø9       MOV WORD PTR [BX+DI],CX
Ø1Ø5 >DEC DX<return>
Ø1Ø5 4A     DEC DX
Ø1Ø6 >JNE WORD PTR 1Ø3<return>
Ø1Ø6 75FB            JNE WORD PTR 1Ø3
Ø1Ø8 >X<return>
**** END OF LINE ASSEMBLY ****
```

Example
Using Local Symbols

```
>ASM 1ØØ<return>
**** 8Ø86/88/186/188 LINE ASSEMBLER VX.XLA ****

Ø1ØØ >CSEG 5ØØØ<return>
Ø1ØØ >MOV DX,8<return>
Ø1ØØ BAØ8ØØ    MOV DX,8
Ø1Ø3 >L5 MOV WORD PTR [BX+DI],CX<return>
Ø1Ø3 89Ø9   L5 MOV WORD PTR [BX+DI],CK
Ø1Ø5 >DEC DX<return>
Ø1Ø6 >JNE WORD PTR L5<return>
Ø1Ø6 75FB              JNE WORD PTR L5
Ø1Ø8 >X<return>
**** END OF LINE ASSEMBLY ****
```

Example
Using Assembler
Directives

```
>ASM<return>
**** 8Ø86/88/186/188 LINE ASSEMBLER VX.XLA ****

Ø1Ø8 >ORG $+1ØØØ<return>
11Ø8 >CSEG<return>
11Ø8 >CSEG 5ØØØ
11Ø8 >L2 EQU 4<return>
11Ø8 >L3 EQU L5-$+2<return>
11Ø8 >L1 DB "TEST",Ø<return>
11Ø8 54 45 53 54 ØØ
11ØD >ORG 2ØØØ<return>
2ØØØ >MOV SI, WORD  PTR L1<return>
2ØØØ 8B36Ø811   MOV SI, WORD   PTR L1
2ØØ4 >MOV DI,WORD   PTR L3<return>
2ØØ4 8B3EFCEF   MOV DI,WORD PTR L3
2ØØ8 >MOV CX,4<return>
2ØØ8 89Ø4ØØ    MOV CX,4
2ØØ8 >REP<return>
2ØØ8 F2
2ØØ9 >MOV SB
2ØØ9 A4      MOV SB
2ØØA >X<return>
**** END OF LINE ASSEMBLY ****
 >
```

Example
Error Message
In Response To ?

```
>ASM<return>
**** 8Ø86/88/186/188 LINE ASSEMBLER VX.XLA ****

2ØØD >´label JNZ WORD PTR 1Ø34 <return>
                             ?

>?
ERROR #9
ARGUMENT OUT OF RANGE
```

Example
Using Symbols With
Symbolic Debug
Hardware Installed

```
>ASM <return>
**** 8Ø86/88/186/188 LINE ASSEMBLER VX.XLA ****

2ØØD >JZ $+6<return>
2ØØD 74Ø4          JZ $+6
2ØØF >JMP WORD PTR 1Ø34<return>
2ØØF FF26341Ø     JMP WORD PTR 1Ø34
2Ø13 >´label EQU 4ØØØ<return>
2Ø13 >´page_table EQU 189Ø<return>
2Ø13 >´pt_blink EQU 4<return>
2Ø13 >MOV BX,´page_table<return>
2Ø13 BB9Ø18     MOV BX,page_table
2Ø16 >LEA BX,[BX+´pt_blink ] <return>
2Ø16 8D5FØ4     LEA BX,[BX+´pt_blink ]
2Ø19 >X<return>
**** END OF LINE ASSEMBLY ****
 >
```

4.11 STATEMENT CONTROL

You can save time and effort by using less keystrokes when you utilize the Repeat and Macro functions. Your individual need for these capabilities will appear as you identify sequences of commands that you use repeatedly.

- The Repeat command repeats one or more commands a specified number of times or indefinitely.
- The Macro capability allows you to define and save up to ten macros, calling them out as necessary.

4.11.1 Repeat Command

An asterisk (*) at the beginning of a command line repeats one or more commands. (The asterisk <u>must</u> be the first character of the input line.) If the asterisk is followed by an optional decimal argument, you can specify the number of times to repeat the buffer contents. A command having normal syntax succeeds this argument. If the argument is zero, the buffer content is not executed.

- The repeat argument must be specified in decimal and cannot be a variable
- There must be a space following the repeat argument if the first character in the macro is a decimal digit

For example:

```
>*5STP;DT

>* 5STP;DT

>* 5 STP;DT
```

In each of these three examples, the "STP;DT" command is repeated five times. Note that 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.

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 = IP-$C324
```

In this example, single stepping continues until the instruction pointer equals 0C324 Hex. If the IP does not reach 0C234, you can still use CNTL Z to stop the repeat.

4.11.2 Loop Counter

When a repeat is initialized, just before execution begins, the value of the repeat argument is assigned to a variable called "LIM;" another variable called "IDX" is reset to zero. After each execution "IDX" is incremented and then compared to "LIM;" if "IDX" is greater than or equal to "LIM" the repeat is stopped. Since "LIM" and "IDX" are variables, they may be used in commands or modified by the execution of the repeat.

Here are three examples:

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

>M $1000
$001000 $34         >*4 LIM-IDX-1
$001001 $C0
$001002 $BF
$001003 $00

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

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.

4.11.3 Macros

Defining Macros

You can define up to ten macros. They are referred to by decimal numbers 0-9. The ten macros are stored in one buffer with #1 beginning first then #2...#9 with #0 being last.

If the sum of the lengths of all ten macros is greater than the buffer length, then macros will be truncated in this order: 0, 9, 8, 7, 6, 5, 4, 3, 2, 1. (This truncation happens silently, without any indication to you.)

Here are some macro definition examples:

```
>_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 equal sign

If the syntax is not correct, 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.

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 #0 would result in those macros remaining null. Also, if the length of any macro from #1 to #7 was increased after filling the buffer, then macro #8 would be truncated 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 one 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=PCO;RET;STP;DT
>_2=PCO=$1000
>_3=M $2000+GDO
>_4=@(RR14+4)
>_6=
```

This is an example of macro display:

```
>MAC
_1=PCO;RET;STP;DT
_2=PCO=$1000
_3=M $2000+GDO
_4=@(RR14+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 can 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 "R0" register cannot be specified by taking the last character of one macro ("R") and concatenating it with the first character of the next macro ("0"). If several macros each contain a single command, and you want to execute them serially as a string of commands, use the semicolon 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. For example:

```
    _1=STP;DT;
    _2=DR

    (This is called out by _1_2)
```

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.

Saving Macros

To load and save macros, enter the following:

```
>LD 5<return>
>SAV 5<return>
```

Please see the LD/SAV section, 3.7.2, for information about initializing the EEPROM.

Clearing Macros

If you define a macro but only type a <return> following the equal sign, 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.

To nullify a specific macro, enter:

```
_n=<return>
```

To nullify all macros, enter:

```
>CMC<return>
```

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 user-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.

Remember that the Event Monitor System must be set up prior to its use. Comparator values and WHEN/THEN statements 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.

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

	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**
	CTL	count limit comparator	16
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	

*single address or address range
**includes Don´t Cares

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:

COMPARATOR TYPE (AMOUNT)	DATA TYPE
Address (2)	Integer, Internal Range External Range
Data (2)	Integer, Don't Care
Status (2)	Integer, Don't Care
Count Limit (1)	Integer
Logic State Analyzer (1)	Integer, Don't Care

Values contained in any other register in the system may be assigned to the comparators, as long as the data types are compatible, for example:

```
S1 = MMS
```

The ES Event Detectors can be set up to resolve these three conditions. Assume the byte in question is at $04001. This byte could be accessed by the address $04001 or $04000. If the address $04001 is on the bus, then the byte is accessed. If the address $04000 is on the bus, and the bus cycle is a 16-bit cycle, then the byte is accessed. If the address $04000 is on the bus, and the bus cycle is an 8-bit cycle, then the byte is not accessed. The following setup will handle this condition;

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

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

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).

NOTE

ODD BOUNDARIES

The address comparators in the 80186 may need to be specially set up because it is a 16-bit machine with a prefetch QUE and byte based instructions. This leads to problems with breaking on instructions that occur on odd boundries. When the 80186 prefetches the instruction, it outputs the even address. Both bytes are fetched, and the actual (odd) address of the byte in question is never seen. The traditional idea of setting the Event Detector to the odd address will obviously not work. If the 80186 jumps to the odd address, the odd address does appear on the bus, and that byte alone is fetched. In this case, the traditional sense of setting up the Event Detector does work. The final case is when the low byte only is read. In this case, the even address appears on the bus, but the odd byte is not read.

- To set an address comparator to a single address:

  >AC1.3 = $06FF<return>

  or

  >AC2.1 = $3488<return>

  or

  >AC1 = IP + $200 <return>

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

IRA
TO
LEN

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

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

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

In order to load the value from another CTL register into the hardware counter, a 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
CTL.2 = #10
AC1.2 = $7840
WHE[N] AC1 THE[N] RCT, GRO 2
2 WHE AC1 THE[N] CNT
2 WHE[N] CTL THE[N] BRK
```

Event groups are discussed in more detail in Section 5.5.

5.3.3 Data Comparators

The data comparators are set like the address comparators. D a t a 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
```

o To assign a Don't Care value

```
    DC2.3 = $FF00 DC $FF
          or
    DC2.3 = $FFXX
    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 80186/80188-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 5-2 shows how to set S1 and S2.

NOTE

Do not set S1 or S2 to break on a type 2 interrupt. This includes an NMI. You should set up the system to break on the vector fetch or the starting address of your interrupt routine or the emulator will not work correctly.

Table 5-1.
Status Mnemonics

MNEMONIC	**DESCRIPTION**
BYT	Byte Access
HLT	Halt Status
IAK	Interrupt Acknowledge
IF	Instruction Fetch
IOA	IO Access
MEM	Memory Access
NBC	No Bus Cycle
NMI	NMI Cycle
OVL	Overlay Access
QD1-6	Que Depth (1-6)
QF	Que Flush Cycle
RD	Read
RIO	Read IO Status
RM	Read Memory Status
TAR	Target Access
WIO	Write IO Status
WM	Write Memory Status
WR	Write
WRD	Word Access
DMA	DMA Cycle

Example 5-1.
Setting Status
Comparator

1	2		3		4		5		6		7		8
S1 =	TAR	+	RD	+	BYT	+	MEM	+	HLT	+	QD1	+	QF
S2	OVL		WR		WRD		IOA		IAK		QD2		NMI
									NBC		QD3		DMA
									RIO		QD4		
									RM		QD5		
									WIO		QD6		
									WM				
									IF				

Note the eight-column format. A status comparator may be set with a maximum of one constant from each of columns 2-7; and as many as you want from column 8.

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

- Some sample status comparators are:
  >S1 = TAR + RD + IF + QF
  >S2 = IOA + WR + DMA
  >S2 = OVL + RIO

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.

Figure 5-1.
Activated
Bit Values

	X4				X3				X2		X1				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	NMI			X87 DMA							$\overline{\text{QF}}$	MEM/ IOA	TAR/ OVL	RD/ WR	BYT WRD

X4	X3	X2	X1
ALT = 0	IAK = 0	QD1 = 2	IOA + OVL + WR + WRD = Ø
STA = 1	RIO = 1	QD2 = 4	BYT = 1
COD = 2	WIO = 2	QD3 = 6	RD = 2
DAT = 3	HLT = 3	QD4 = 8	RD + BYT = 3
	IF = 4	QD5 = A	TAR = 4
	RM = 5	QD6 = C	TAR + BYT = 5
	WM = 6		TAR + RD = 6
	NBC = 7		TAR + RD + BYT = 7
			MEM = 8
			MEM + BYT = 9
			MEM + RD = A
			MEM + RD + BYT = B
			MEM + TAR = C
			MEM + TAR + BYT = D
			MEM + TAR + RD = E
			MEM + TAR + RD + BYT = F

***NOTE:** The ALT, STA, COD and DAT statuses are used by 8086/8088 emulators only.

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) **$ 000000100 DC 0000F8FF**

The mask value shows that bits 8, 9, and 10 have been enabled. The 1 in the X_3 column to the left of the DC operator can be matched with the 1 in the same column of the table, indicating that the mnemonic entered was RIO.

b) **$ 00000100 DC 0000F8FB**

Mask values: all except 2, 8, 9, and 10 are masked

Mnemonic values: the 1 in X_3 corresponds to RIO. The 0 in column X_1 corresponds to some combination of IOA OVL, WR, and WRD; since only bit 2 is activated, though, the mnemonic entered must have been OVL.

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

c) **$ 00000454 DC 0000B8FB**

Mask values: all bits except 2, 8, 9, 10, and 14 are masked.

Mnemonic values: since bit 14 is activated and a 4 shows in X_4 NMI must have been entered; the 5 in X_3 corresponds to RM, and the 4 in X_1 matches with TAR.

Original entry: **S1 = NMI + RM + TAR**

d) **$ 00000009 DC 0000FFE4**

Mask values: all except 0, 1, 3, and 4 are masked.

Mnemonic values: the 0 in X_2 corresponds to QF in the table; the 9 in X_1 indicates that bits 0, 1 & 3 are activated.

Original entry: **S1=QF + MEM + WR + BYT**

e) **$ 00001000 DC 000068FF**

Mask values: bits 8, 9, 10, 12, 13 and 15 are enabled.

Mnemonic values: bit 14 is masked so X4 is used to indicate which segment register was used to form the address during this bus cycle. Since bit 12 is set, the chosen segment is STA. Bit 11 of X_3 is masked which selects the processor status and since X_3 is zero, IAK is the activated status.

Original Entry: **S1 = STA + IAK**

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 occurence of a specified event. RCT resets this counter.

- 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 the Timing Strobe, 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 possibilities the system is capable of.

- 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 occuring in the range AC1 will be traced.

- To use the pass counter

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

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 pattern appears at either of two addresses during a write cycle

```
>WHE[N] AC1 AND DC1 AND SI OR AC2 AND DCI AND SI 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
```

In some of the examples shown AC1 was used; however, AC2 could also have been used.

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 an "Interrupt Return" (IRET) instruction to properly return to the interrupted routine. The message FSI ACTIVE will be printed when an FSI occurs.

NOTE

When using the Event System FSI action, some internal cycles are traced immediately preceeding the jump to the SIA. This occurs because the cycles are not purged from trace. If they were purged, the FSIs would occur more slowly which is undesirable, and the disassembler would not be able to disassemble the code.

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 = $F2DO
>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 $4012) will be pushed onto the CPU stack as the return address. The program counter will be set to the value specified in SIA ($F2DO), 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 an "Interrupt Return" (IRET) instruction (this will vary from processor to processor) and the CPU will unstack the pushed program counter ($302E or $4012 in this example) and continue running from where it left off.

FSI is not enabled during single step.

5.5 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 by placing a number preceding the WHEN/THEN statement or by appending a group number to the event detectors. The following examples would both define an event for Group 2:

2 WHE[N] S1 AND LSA THE[N] BRK

WHE[N] S1.2 AND LSA.2 THE[N] BRK

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.

- Possible Event Groups:*
  Group 1 =AC1.1, AC2.1, DC1.1, DC2.1, S1.1, S2.1, LSA.1, CTL.1
  Group 2 =AC1.2, AC2.2, DC1.2, DC2.2, S1.2, S2.2, LSA.2, CTL.2
  Group 3 =AC1.3, AC2.3, DC1.3, DC2.3, S1.3, S2.3, LSA.3, CTL.3
  Group 4 =AC1.4, AC2.4, DC1.4, DC2.4, S1.4, S2.4, LSA.4, CTL.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.

Example 5-4. Sample Valid WHEN/THEN Statements

- 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>
```

- To use more than one event group

```
>WHE[N] AC1 AND S1 THE[N] CNT
>WHE[N] CTL THE[N] RCT, GROUP 2
>2 WHE[N] AC1 AND S1 THE[N] TRC, CNT
>2 WHE[N] CTL THE[N] BRK
>AC1.1 = $4010; AC1.2 = $4011
>CTL.1 = 3; CTL.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. Then twenty read cycles will be traced. When count limit is reached the event monitor will break halting emulation.

5.6 OPTIONAL LOGIC STATE ANALYZER LSA

The Logic State Analyzer option assembly includes 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 including:

- 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 trace when an activity occurs and turn off the trace when the activity is terminated. The two event groups are required to specify the on and off conditions.

```
>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 the figure 5-2.) to generate a trigger which is sent to the LSA pod and to the BNC connector on the rear panel. The trigger is a low-going-high signal for approximately one bus cycle and is generated approximately 70 ns after an event.

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.

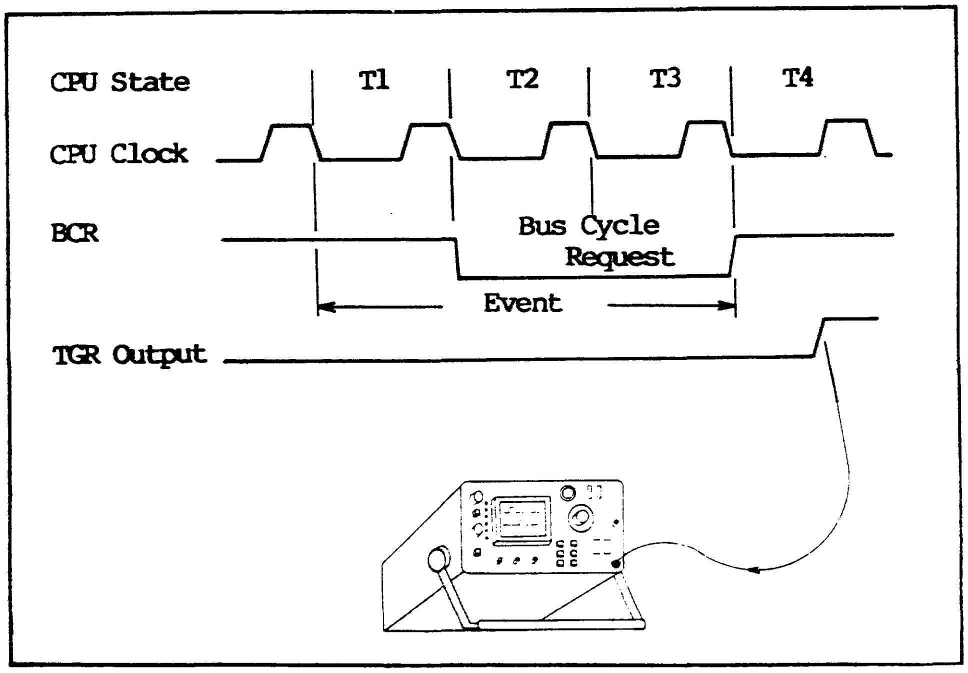
Another use for the trigger would be to connect two emulators, using the signal from the first to trigger a break in the second emulator. The Event Monitor System would be programmed as shown:

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

- Emulator 2: **WHE[N] LSA THE[N] BRK**

CPU State
CPU Clock

BCR
Bus Cycle
Request
Event
TGR Output
T1
T2
T3
T4



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 2.3.3).

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.

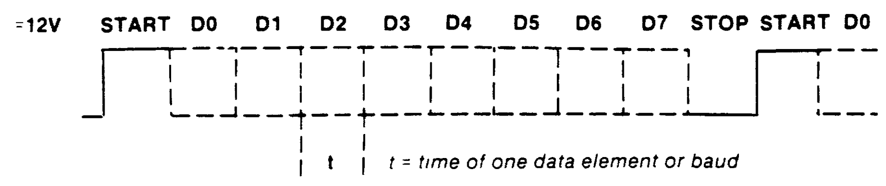


Figure 6-1.
Format of a Serial Word

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. Control is transferred to a host system interfaced through the computer port. This overrides the setting of the interface parameter switch.

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 and 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, we will 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).

o 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>

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

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

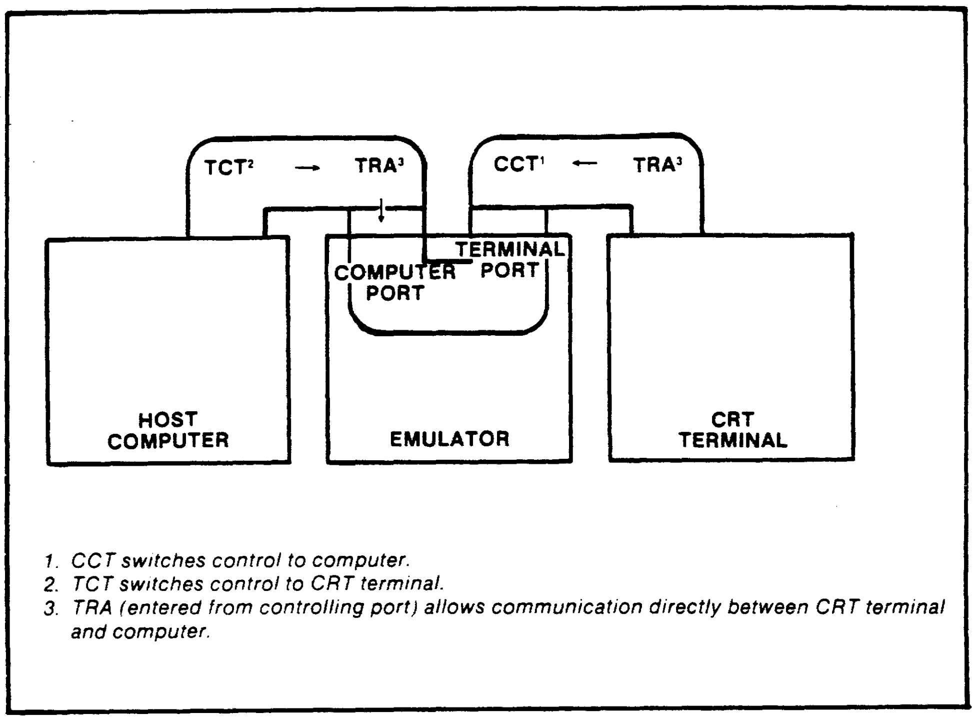

TCT²
TRA³
CCT¹
TRA³
TERMINAL PORT
COMPUTER PORT
HOST COMPUTER
EMULATOR
CRT TERMINAL
1. CCT switches control to computer.
2. TCT switches control to CRT terminal.
3. TRA (entered from controlling port) allows communication directly between CRT terminal and computer.

Figure 6-2.
System Control

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
- Upload Symbols - UPS

6.4.1 Upload and Download UPL DNL

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

Example 6-2.
Upload and Download

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

Example 6-3.
Verify

- 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 Special Function (SF) operator. These routines are used for debugging hardware or accommodating unusual hardware conditions.

7.2 RAM TESTS

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

7.2.1 SF 0, <RANGE> <RETURN>

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> <RETURN>

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> <RETURN>

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 "linefeed." 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> <RETURN>

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

7.3 SCOPE LOOPS (MEMORY, I/O)

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. The data length will be the global default.

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.

NOTE

All special functions for the 80188 are executed in BYM (byte mode).

MEMORY AND I/O SCOPE LOOPS

	Memory	I/O		Parameters Required	Description
SF	4	24	B/W	<addr><data><return>	Writes alternating zeroes and user specified data to the target system.
SF	5	25	B/W	<addr><return>	Executes "reads" into the target system. (Peeks)
SF	6	26	B/W	<addr><data><return>	Executes "writes" into the target system. (Pokes)
SF	7	27	B/W	<addr><pat><return>	Writes pattern to the target system, then complement pattern.
SF	8	28	B/W	<addr><pat><return>	Writes the pattern to the target system but the pattern is rotated one bit left after each "write."
SF	9	29	B/W	<addr><data><return>	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.
SF	10	30	B/W	<range><return>	Forces NOPs to the target system.
SF	11	31	B/W	<addr><data><return>	Writes an incrementing count value to a constant address.
SF	12	32	B/W	<range><return>	Read data over entire range.

7.4 CLOCK AND CRC

CLK <RETURN>
CRC <RETURN>
CRE/CRO <RANGE> <RETURN>

The CLK and CRC operators will be useful during diagnostic testing. CLK reads the target system clock and returns the value in KHz, accurate to 1 to 2 KHz.

CRC computes a cyclic redundancy check over an address range. It will be useful for checking if a block of memory has changed. The key sequences are shown in this example:

If your code is split into two PROMS with one even and the other one odd, the CRE/CRO operators allow you to do a cyclic redundancy check over each PROM.

***NOTE**

For Diagnostic purposes CRC will function for byte or word. Also, the CRC value may be calculated over only odd addresses or only even addresses.

Example 7-1.
Clock and CRC

- To read the target system clock:
  >CLK<return>

- To compute the CRC:
  >CRC <address range><return>

- To calculate a CRC over even bytes only:
  >CRE <address range><return>

- To calculate a CRC over odd bytes only:
  >CRO <address range><return>

7.5 Bus
BUS

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

- >BUS

```
 NMI   ARDY   SRDY   INTO   INT1   INT2/INTAO   INT3/INTA1   TEST
  0     0      0      0      0         0             0          0
```

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

7.6 COM AND DIA
COM

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 which 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 or 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.

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.

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. 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.

8.2 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 at Applied Microsystems Corporation if you experience any problems that do not fall within this range of items. Before you call our service department, display your software revision number by typing:

REV
You will be asked for this information when you call.

```
 >REV
Thu Oct 11 13:44:32 PDT 1984 - ESL 2.10
Tue Oct 16 17:11:50 PDT 1984 - FW  3.6
```

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. Emulator and target system not compatible	1.1
	3. 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
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	3.5 #20
	3. Wrong serial data format selected	3.5 #26

[1] Call Applied Microsystems (Customer Service for ES Emulators)
[2] Check Target System

8.3 PARTS LIST

The following parts are available for you to order:

Part	Number
Short Cable Set	600-11501-00
Long Cable Set	600-11142-00

APPENDIX A
SERIAL DATA FORMATS

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

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 various 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 the 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.

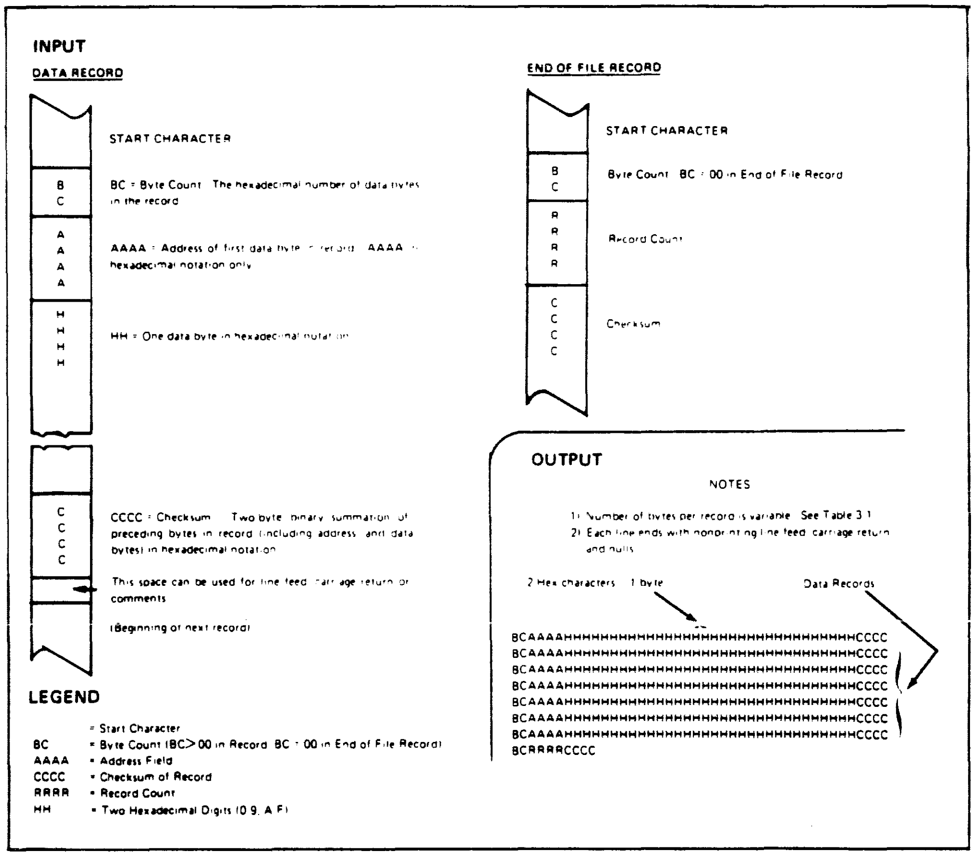
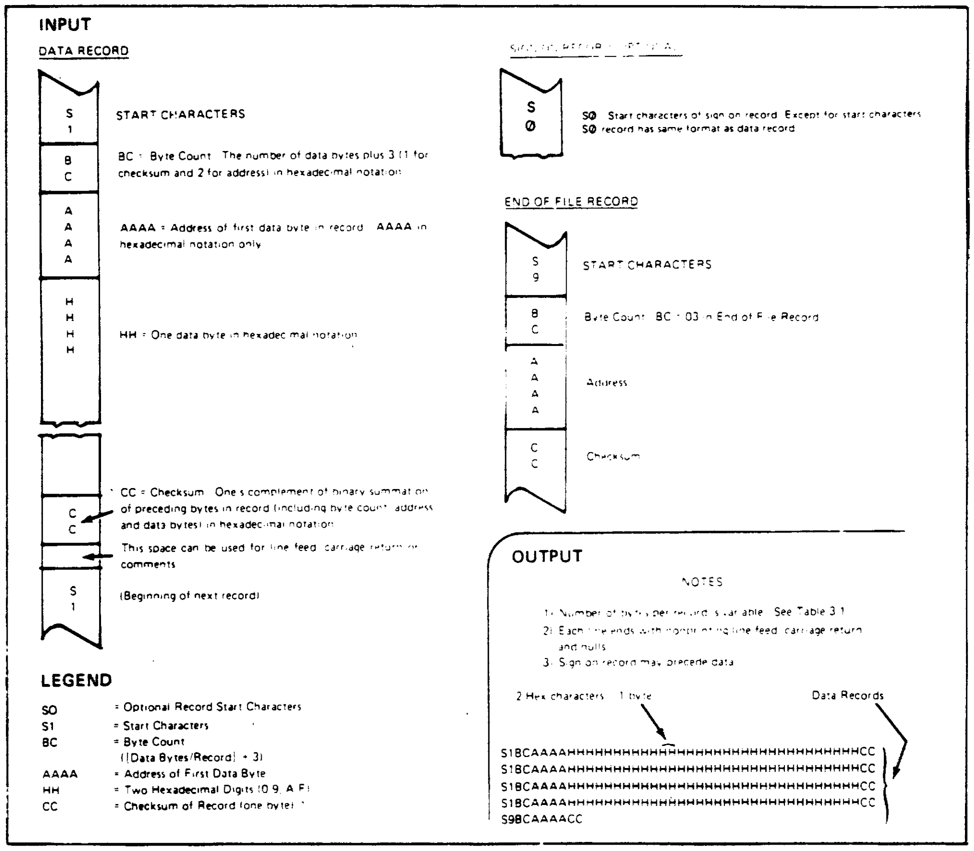

INPUT
DATA RECORD
START CHARACTER
B C
BC = Byte Count The hexadecimal number of data bytes in the record
A A A A
AAAA = Address of first data byte in record AAAA in hexadecimal notation only
H H H H
HH = One data byte in hexadecimal notation
C C C C
CCCC = Checksum Two byte binary summation of preceding bytes in record (including address and data bytes) in hexadecimal notation
This space can be used for line feed carriage return or comments
(Beginning of next record)
LEGEND
= Start Character
BC = Byte Count (BC>00 in Record BC = 00 in End of File Record)
AAAA = Address Field
CCCC = Checksum of Record
RRRR = Record Count
HH = Two Hexadecimal Digits (0 9, A F)
END OF FILE RECORD
START CHARACTER
B C
Byte Count BC = 00 in End of File Record
R R R R
Record Count
C C C C
Checksum
OUTPUT
NOTES
1) Number of bytes per record is variable See Table 3 1
2) Each line ends with nonprinting line feed carriage return and nulls
2 Hex characters 1 byte
Data Records
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCCC
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCCC
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCCC
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCCC
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCCC
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCCC
BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCCCC
BCRRRRCCCC

Figure A-1. Specifications for MOS Technology Data Files

A.2 MOTOROLA EXORCISER FORMAT

Motorola data files may begin with a sign-on record, initialized by the code SØ. 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.

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.



INPUT
DATA RECORD
S 1
START CHARACTERS
B C
BC = Byte Count The number of data bytes plus 3 (1 for checksum and 2 for address) in hexadecimal notation
A A A A
AAAA = Address of first data byte in record AAAA in hexadecimal notation only
H H H H
HH = One data byte in hexadecimal notation
C C
CC = Checksum One's complement of binary summation of preceding bytes in record (including byte count, address and data bytes) in hexadecimal notation
This space can be used for line feed, carriage return or comments
S 1
(Beginning of next record)
LEGEND
SO = Optional Record Start Characters
S1 = Start Characters
BC = Byte Count ([Data Bytes/Record] + 3)
AAAA = Address of First Data Byte
HH = Two Hexadecimal Digits (0-9, A-F)
CC = Checksum of Record (one byte)
S Ø
SØ Start characters of sign on record Except for start characters SØ record has same format as data record
END OF FILE RECORD
S 9
START CHARACTERS
B C
Byte Count BC = 03 in End of File Record
A A A A
Address
C C
Checksum
OUTPUT
NOTES
1) Number of bytes per record is variable See Table 3.1
2) Each line ends with nonprinting line feed, carriage return and nulls
3) Sign on record may precede data
2 Hex characters 1 byte
Data Records
S1BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
S1BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
S1BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
S1BCAAAAHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
S9BCAAAACC

Figure A-2. Specifications for Motorola EXORciser 16-BM Data Files

A.3 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.

A.3.1 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 teletypewritters, some programs may limit the number of bytes to as few as 28 (56 printable characters in S-record).
checksum	2	The least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields.

Each record may be terminated with a CR/LF/NULL. Additionally, an S-record may have an initial field to accommodate other data such as line numbers generated by some time-sharing systems.

Accuracy of transmission is ensured by the record length (byte count) and checksum fields.

A.3.2 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.

An S-record format module may contain S-records of the following types:

S0 The header record for each block of S-records. The code/data field may contain any descriptive information identifying the following block of S-records. Under VERSAdos, the resident linker's IDENT command can be used to designate module name, version number, revision number, and description information which will make up the header record. The address field is normally zeroes.

S1 A record containing code/data and the 2-byte address at which the code/data is to reside.

S2 A record containing code/data and the 3-byte address at which the code/data is to reside.

S3 A record containing code/data and the 4-byte address at which the code/data is to reside.

S5 A record containing the number of S1, S2, and S3 records transmitted in a particular block. This count appears in the address field. There is no code/data field.

S7 A termination record for a block of S3 records. The address field may optionally contain the 3-byte address of the instruction to which control is to be passed. There is no code/data field.

S8 A termination record for a block of S2 records. The address field may optionally contain the 3-byte address of the instruction to which control is to be passed. There is no code/data field.

S9 A termination record for a block of S1 records. The address field may optionally contain the 2-byte address of the instruction to which control is to be passed. Under VERSAdos, the resident linker's ENTRY command can be used to specify this address. If not specified, the first entry point specification encountered in the object module input will be used. There is no code/data field.

Only one termination record is used for each block of S-records. S7 and S8 records are usually used only when control is to be passed to a 3- or 4-byte address. Normally, only one header record is used, although it is possible for multiple header records to occur.

A.4 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:

```
S00600004844521B
S1130000285F245F2212226A000424290008237C2A
S11300100002000800082629001853812341001813
S113002041E900084E42234300182342000824A952
S107003000144Ed492
S9030000FC
```

The module consists of one SØ record, four S1 records, and an S9 record.

The S0 record is comprised of the following character pairs:

S0	S-record type S0, indicating that it is a header record.
06	Hexadecimal 06 (decimal 6), indicating that six character pairs (or ASCII bytes) follow.
00 00 }	Four-character 2-byte address field, zeroes in this example.
48 44 } 52	ASCII H, D, and R - "HDR".
1B	The checksum.

The first S1 record is explained as follows:

S1	S-record type S1, indicating that it is a code/data record to be loaded/verified at a 2-byte address.
13	Hexadecimal 13 (decimal 19), indicating that 19 character pairs, representing 19 bytes of binary data, follow.
00 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)
```

o (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:

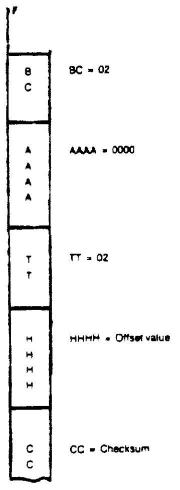
A.5 INTEL INTELLEC 8/MDS FORMAT

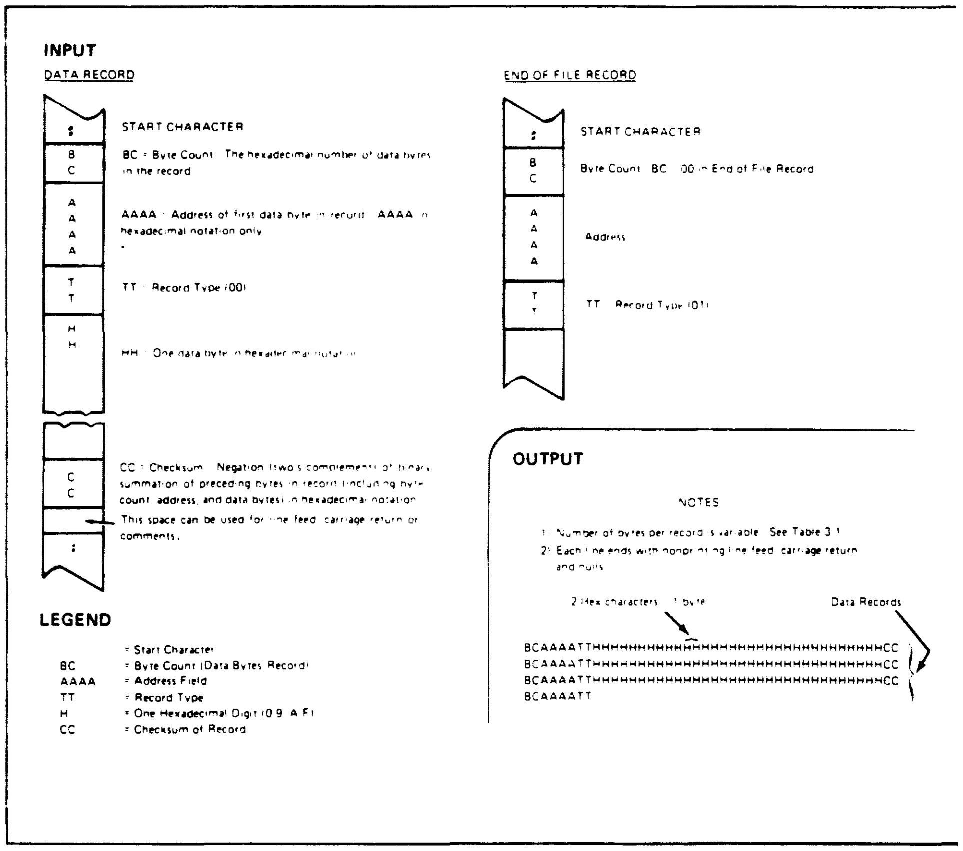
Intel data records begin with a nine-character prefix and end with a two-character suffix. The byte count must equal the number of data bytes in the record.

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.





START CHARACTER

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

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

TT = Record Type (00)

HH = One data byte in hexadecimal notation

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

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

START CHARACTER

Byte Count BC 00 in End of File Record

Address

TT Record Type (01)

LEGEND

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

NOTES

1) Number of bytes per record is variable See Table 3 1

2) Each line ends with nonprinting line feed carriage return and nulls

```
BCAAAATTHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
BCAAAATTHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
BCAAAATTHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
BCAAAATT
```

Figure A-3. Specifications for Intel Intellec 8/MDS Data Files

A.6 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 and a two-digit address check. Data is represented by pairs of hexadecimal characters. The byte count must equal the number of data bytes in the record. The suffix is a two-character data check. Data is represented by pairs of hexadecimal characters. The byte count must equal the number of data bytes in the record. The suffix is a two-character data check.

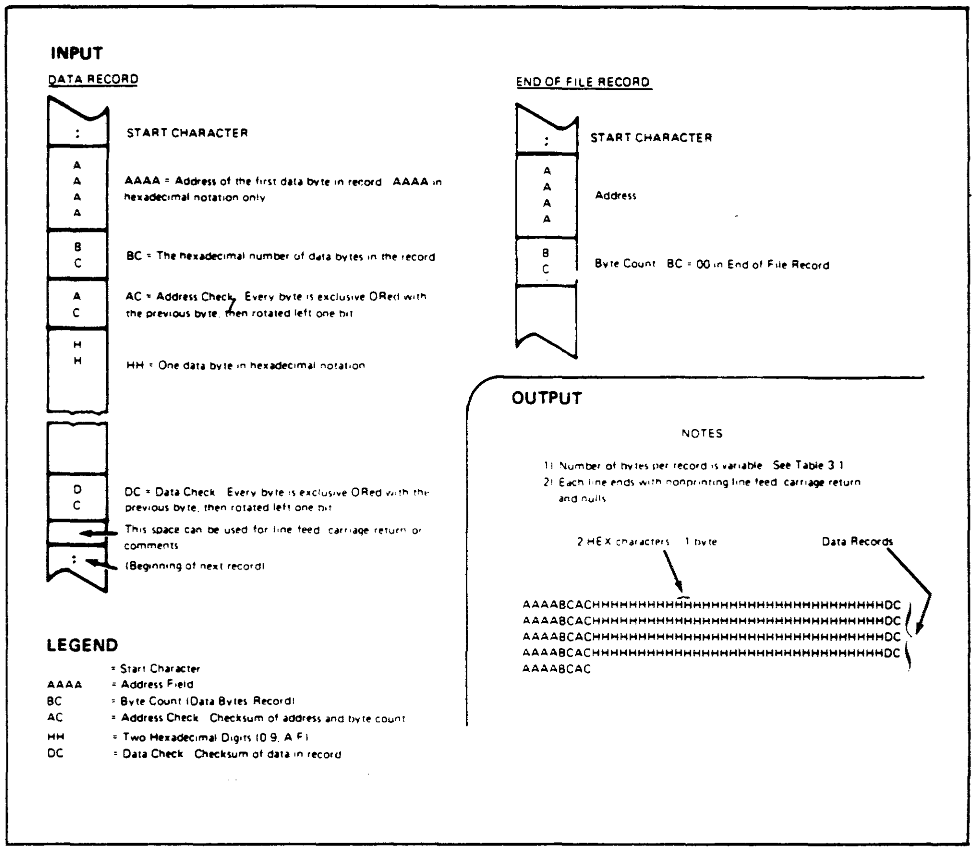

INPUT
DATA RECORD
:
START CHARACTER
A A A A
AAAA = Address of the first data byte in record AAAA in hexadecimal notation only
B C
BC = The hexadecimal number of data bytes in the record
A C
AC = Address Check Every byte is exclusive ORed with the previous byte, then rotated left one bit
H H
HH = One data byte in hexadecimal notation
D C
DC = Data Check Every byte is exclusive ORed with the previous byte, then rotated left one bit
This space can be used for line feed carriage return or comments
:
(Beginning of next record)
END OF FILE RECORD
:
START CHARACTER
A A A A
Address
B C
Byte Count BC = 00 in End of File Record
OUTPUT
NOTES
1) Number of bytes per record is variable See Table 3 1
2) Each line ends with nonprinting line feed carriage return and nulls
2 HEX characters 1 byte
Data Records
AAAABCACHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHDC
AAAABCACHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHDC
AAAABCACHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHDC
AAAABCACHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHDC
AAAABCAC
LEGEND
= Start Character
AAAA = Address Field
BC = Byte Count (Data Bytes Record)
AC = Address Check Checksum of address and byte count
HH = Two Hexadecimal Digits (0 9, A F)
DC = Data Check Checksum of data in record

Figure A-4. Specifications for Signetics Absolute Object Data Files

A.7 TEKTRONIX HEXADECIMAL FORMAT

Figure A-5 illustrates of valid Tektronix data file. The data in each record is sandwiched between the start character, a slash (/), and a two-character checksum. Following the start character, 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 verifications.

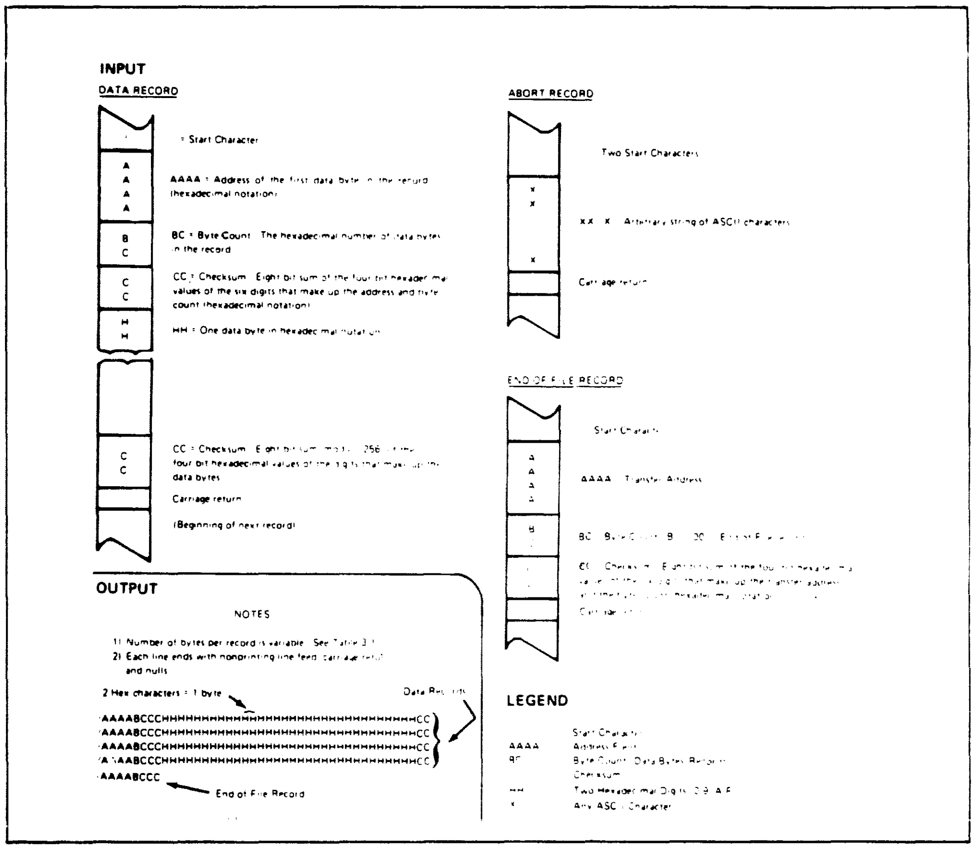

INPUT
DATA RECORD
= Start Character
AAAA = Address of the first data byte in the record (hexadecimal notation)
BC = Byte Count. The hexadecimal number of data bytes in the record
CC = Checksum. Eight bit sum of the four bit hexadecimal values of the six digits that make up the address and byte count (hexadecimal notation)
HH = One data byte in hexadecimal notation
CC = Checksum. Eight bit sum modulo 256 of the four bit hexadecimal values of the digits that make up the data bytes
Carriage return
(Beginning of next record)
ABORT RECORD
Two Start Characters
xx x Arbitrary string of ASCII characters
Carriage return
END OF FILE RECORD
Start Character
AAAA Transfer Address
CC Checksum
Carriage return
OUTPUT
NOTES
1) Number of bytes per record is variable. See Table 3.1
2) Each line ends with nonprinting line feed, carriage return and nulls
2 Hex characters = 1 byte
Data Records
/AAAABCCCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
/AAAABCCCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
/AAAABCCCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
/AAAABCCCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHCC
/AAAABCCC
End of File Record
LEGEND
Start Character
AAAA Address Field
BC Byte Count, Data Bytes Record
Checksum
HH Two Hexadecimal Digits 0-9, A-F
x Any ASCII Character

Figure A-5. Specifications for Tektronix Hexadecimal Data Files

A.8 EXTENDED TEKHEX

Extended Tekhex uses three types of message blocks:

1. A data block contains object code.
2. A symbol block contains information about a program section and the symbols associated with it. This information is needed only for symbolic debug.
3. 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 carriage 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 permit 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.8.1 Variable-Length Fields

In Extended Tekhex, certain fields may vary in length from 2 to 17 characters. This practice enables you to compress your data by eliminating leading zeros from numbers and trailing spaces from symbols. The first character of a variable-length field is a hexadecimal digit that indicates the length of the rest of the field. The digit 0 indicates a length of 16 characters.

For example, the symbols START, LOOP, and KLUDGESTARTSHERE are represented as 5START, 4LOOP and 0KLUDGESTARTSHERE. The values 0, 100H, and FF0000H are represented as 10, 3100, and 6FF0000.

A.8.2 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.8.3 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 symbolize 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 concept of sections is discussed in detail in the Assembler Core manuals for both A series and B series assemblers. The significance of sections with regard to symbolic debug is illustrated in the Emulation section of this manual.

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	5 to 35	Zero or more symbol definition fields, Definition(s) each as described in Table A-7.

Table A-6.
Extended Tekhex
Symbol Block
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.

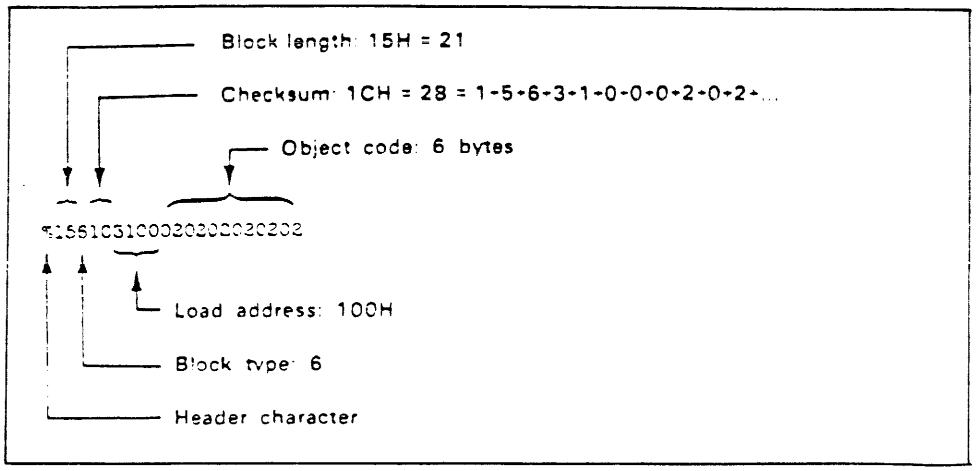


Figure A-6.
Tekhex Data Block

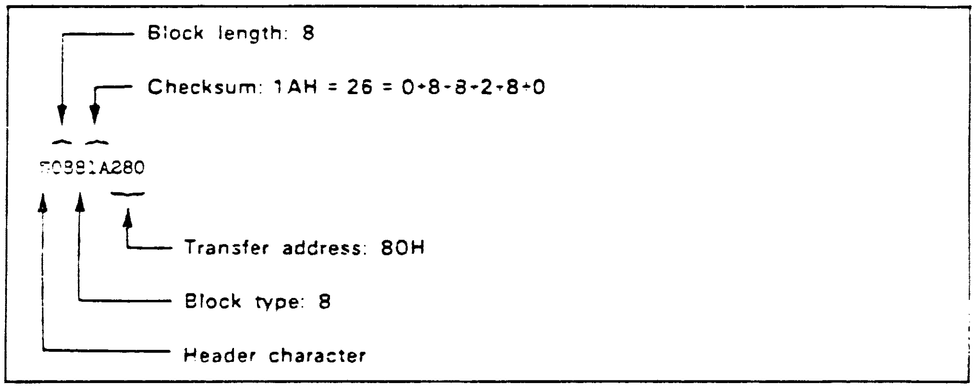


Figure A-7.
Tekhex Termination Block

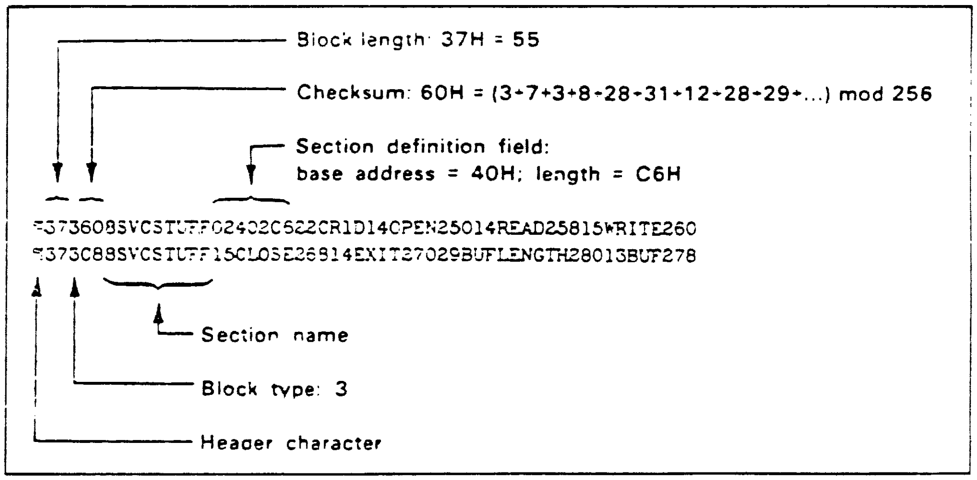

Block length: 37H = 55
Checksum: 60H = (3+7+3+8+28+31+12+28+29+...) mod 256
Section definition field:
base address = 40H; length = C6H
%373608SVCSTUFF02402C622CR1D14OPEN25014READ25815WRITE260
%373C88SVCSTUFF15CLOSE26814EXIT27029BUFLENGTH28013BUF278
Section name
Block type: 3
Header character

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 probe (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, control 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.

B.2 REFERENCE MATERIAL

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
	Addressed Commands	Universal Commands	Listen Addresses		Talk Addresses		Secondary Addresses or Commands	

KEY

octal	25 PPU	GPIB code
	NAK	ASCII character
hex	15 21	decimal

Table B-2
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> ;GR0 = ´<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).

Example C-1.

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 contains no defined symbols. The second disassembly shows the effect of the a symbolic definition. Note how the program address display mode changes as the addresses move out of the section.

Figure C-1. Disassembly Trace With Symbols

```
>DTE
SEQ# ADDR  OPCODE    MNEMONIC   OPERAND FIELDS BUS CYCLE DATA
-----------------------------------------------------------------
0069 PQM _8086_80186_Test
0069 1000  B90F00      MOV      CX,000F
0068 1003  BE0020      MOV      SI,2000
0066 1006  BF0022      MOV      DI,2200
0065 1009  A5          MOVS     WORD PTR   2000>FFF0  2200<FFF0
0064 100A  F3          REPZ
0064 100B  A4          MOVS     WORD PTR
                       2002>3E  2202<3E  2003>FF  2203<FF  2004>00  2204
                       2005>00  2205<00  2006>FF  2206<FF  2007>FF  2207
                       2008>00  2208<00  2009>00  2209<00  200A>FF  220A
                       2008>FF  2208<FF  200C>00  220C<00  200D>00  220D
                       200E>FF  220E<FF  200F>F5  220F<F5  2010>00  2210
0063  100C 038100FF    ADD      AX,WORD PTR [BX-100][DI]  2111>FF00
0059  1010 890200      MOV      CX,0002
0026  1013 F2          REPNZ
0025  1014 A7          CMPS     WORD PTR   2011>FF10   2211>FF10
0025  1015 C116002405  RCL      WORD PTR Data_Word,05  2400>A002  2400<0
0017  101A C8400004    ENTER    0040.04
                       17FE<0000   FFFE>FFFF  17FC<FFFF  FFFC>FFFF
                       17FA<FFFF   FFFA>FFFF  17F8<FFFF  17F6<17FE
0015  101E  E0E0       LOOPNE   SHORT PGM_8086_80186_Test
```

```
>
>´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 96ØØ 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.

INDEX TO TOPICS (by section number)