SPDU002

XDS/22 TMS9995 Emulator

XDS Extended Development Support

User's Guide

TEXAS INSTRUMENTS

PREFACE

This manual describes the software for the XDS TMS9995 Emulator for the TMS9995 family of sixteen-bit microcomputers. The manual serves as a guide to the design engineer in the use of this extended development support system (XDS) in their efforts to apply the emulator to development of systems using the TMS9995 microprocessor. Proper use of this guide should minimize the time and effort required to get the XDS system up and running productively.

The XDS TMS9995 Emulator User's Guide consists of eight sections and five appendices. These are briefly described as follows.

Section 1. This section introduces the XDS TMS9995 Emulator and its features. The first part of the section provides a brief description of the emulator. This is followed by an extensive walk through session which allows the user to become familiar with the system in a step-by-step process. This session provides hands-on experience with features that are most commonly used by the systems designer.

Section 2. The Monitor. The features of the TMS9995 Emulator's monitor program are described in detail to increase understanding of the software and its operation.

Section 3. Commands Reference. The monitor utilizes over fifty commands for configuring the emulator to the user's specific test conditions. These commands are presented alphabetically for easy access to pertinent reference information for each command. The command's purpose and parameters are described. A simple example is presented to illustrate how the command is entered and executed.

Section 4. Memory Mapping and Assembler. This section provides detailed explanations of the XDS TMS9995 Emulator's memory mapping feature and onboard symbolic assembler. It also provides design considerations for memory controllers being designed for use with the XDS TMS9995.

Section 5. Communication. The XDS TMS9995 Emulator is capable of communicating with a variety of external devices including terminals, logging devices (printers), PROM programmers, and host computer systems. This capability requires extensive manipulation of communication parameters for matching the external devices' specifications. This section provides the information necessary to communicate with these devices. A trouble-shooting guide is provided to assist in solving problems that can occur when electronic devices attempt communication.

Section 6. Multi-processing. The multi-processing capability of the XDS system is described in this section. Up to nine XDS units with various emulators can be linked together in a chain to emulate the multi-processing environment.

Section 7. Breakpoint and Trace Functions. The XDS system provides extensive breakpoint and tracing capabilities to assist in tracking bugs in both hardware and software. These functions are described with extensive walk-through examples to acquaint the user with these powerful design tools.

Section 8. Error Codes and Messages. The error codes, messages, and their interpretation are listed for quick reference. The monitor errors are presented as well as the assembler error and warning codes.

Appendix A. Diagnostic Mode. This appendix presents some basic information that allows the user to access the diagnostic support offered by the Texas Instrument's Regional Technology Centers.

Appendix B. Data and Address Masks. The masking used by several XDS TMS9995 monitor commands is explained in this appendix.

Appendix C. Extended Address and Data. The use and function of the breakpoint/trace board's extended trace cable are explained in this appendix.

Appendix D. Object File Formats. Details regarding the object file formats supported by the XDS system are presented for the following formats:

* TI normal and compressed
* Tektronix hexadecimal
* Intel Intellec
* Motorola

Appendix E. Interrupt Handling. Information regarding the methods in which the XDS TMS9995 Emulator handles target system interrupts is provided in this appendix.

The following list of reference manuals is provided for additional information related to the XDS TMS9995 Emulator.

MANUAL TITLE	TI PART NUMBER
XDS TMS9995 Emulator Hardware Installation and Operation Guide	1603437 - 9701
TMS9995 Data Manual	1603454 - 9701
XDS/22 System Installation and Operation Guide	1603443 - 9701
XDS Breakpoint/Trace Module Installation Guide	1603442 - 9701
TMS9995/99000 Assembly Language Programmer's Guide	1603753 - 9701
XDS Memory Expansion Module Installation Guide	1603441 - 9701

TMS9995 EMULATOR USER'S GUIDE

TABLE OF CONTENTS

SECTION PAGE

SECTION 1

XDS

TMS9995 EMULATOR SYSTEM

1.1 INTRODUCTION

The XDS is shown in Figure 1-1. Please refer to the appropiate Installation and Operation manuals listed in the preface when setting up the system. You must supply a video-display terminal.

FIGURE 1-1. THE XDS MODEL 22

Typically, development systems provide the following basic functions.

* Software development tools
* PROM programming capability
* Real time emulation

These functions have been combined into an integrated computer resource dedicated to the development of software and the support of in-circuit emulation for specific microprocessor applications.

Texas Instruments introduces the XDS, Extended Development Support System, for TI's TMS9995, TMS320, TMS99000, and TMS7000 family of microprocessors.

1.2 DESCRIPTION

The XDS TMS9995 Series Emulation System for the TMS9995 Microprocessor family is a powerful development tool providing in-circuit emulation with breakpoints, logic state trace, and target system debugging capabilities for both hardware and software systems development. Please refer to the XDS TMS9995 Emulator Hardware Installation and Operation Guide for more detailed information on the emulator board. Figure 1-2 shows a line diagram of a typical TMS9995 Emulator system configuration.

```
 ------------                                          -------------
| User's     |                     Software           |    Host     |
| Terminal   |                     Development------->|  Computer   |
|            |                                        |   System    |
 ------------                                          -------------
      |                                                      |
      |                                                      |
  --------------------               --------------------
                     |               |
               -----------------------
              |         XDS           |
Development   |       TMS9995         |
Facilities ---------->| Work Station  |
              |                       |
               -----------------------
                          |
                          |
                          V
                     ----------
                    |  Target  |
                    |  System  |
                     ----------
```

FIGURE 1-2. XDS TMS9995 EMULATOR SYSTEM

1.2.1 Utilizing Existing Host Computer Resources

Systems design engineers can use the following host computer facilities for creating, editing, assembling, and linking programs before beginning the emulation session for final testing and debugging.

* IBM 370 MVS
* DEC VAX series VMS
* TI 990 operating systems based on:
  - DX10
  - TX4
  - TX5

Section 5, Communications, provides detailed information on uploading and downloading object files using the host computer system.

1.2.2 Communication Links

Each XDS unit is equipped with four standard EIA RS-232-C ports for communicating with external devices.

* Port A is connected to the user's "dumb" terminal.
* Port B is reserved for future expansion and is not currently available to the emulator. (Not available on the Model 11.)
* Port C can be connected to a printer for logging emulator sessions or may be used with PROM programming devices. (Not available on the Model 11.)
* Port D can be connected to a host computer system for software development using more sophisticated editors, linkers, and other software development tools available on larger computer systems.

The user's terminal can be used to operate the XDS system or can be routed through XDS to function as a terminal directly connected to the host computer system.

1.2.3 The Monitor Program

The simple, but powerful XDS Monitor program provided in the firmware supports the entire spectrum of Texas Instruments' microprocessor family. The monitor commands give you the means to control emulator functions and communicate with a host system which allows access to mass storage and data generated by the system.

The monitor uses prompting to define te parameters for the commands. Commands and parameter data entries are checked for syntax and validity. Registers are readily accessible through the use of register variable names assigned to each register.

Section 2, The Monitor, provides background information regarding the operation of the TMS9995 Emulator Monitor program.

1.2.4 The Monitor Commands

More than 60 monitor commands provide you with extraordinary flexibility in defining the test conditions for emulation sessions. These commands can be combined in a variety of ways as short procedures which allow several commands to be executed sequentially. These procedures may be constructed in such a way that they are extremely efficient and effective in debugging complicated systems.

The commands are explained in detail in Section 3 and some applications of the commands are outlined in Section 4.

Repeat functions allow procedures or individual commands to be executed indefinitely until you stop them. Scope and analysis loops are simply constructed of command procedures facilitating isolation and rapid diagnosis of problems in test systems.

1.2.5 Independent Operation

The XDS is capable of independent operation using a dumb terminal, an optional printer, and the target system being developed. Using its own line-by-line assembler, a program can be developed which can be stored in either the emulator's or target-system's RAM. Programs are easily displayed in source code format using the emulator's reverse assembler. New programs can be coded, debugged and stored in PROM by connecting a PROM programmer to one of the RS-232-C ports on the XDS unit.

1.2.6 Software and Hardware Breakpoints

You can define up to 10 software breakpoints to halt program execution when instructions in specified locations are executed.

The Breakpoint/Trace module expands capability so that breakpoints can be defined on the basis of hardware activity. These hardware breakpoints halt program execution after a specified set of qualifying conditions is satisfied (i.e., memory read, memory write, instruction acquisition, I/O read, I/O write, address ranges, or data value ranges). Breakpoint event counts of up to 7FFF (32,767) events and delays totaling 7FF (2047) events can be specified for special test cases. Address and data masks can be used for multiple breakpoint ranges.

Please refer to Section 7 for details regarding hardware breakpoints.

1.2.7 Tracing and Extended Address/Data Probes

Traces provide a record of microprocessor activity during program execution. Trace samples are independently qualified using the same criteria as hardware breakpoint events. Address and data bits are controlled by compare and mask qualifiers. The masks allow you to select specific bits of the address or data under comparison to be checked, or ignored, as qualifiers for breakpoint events or trace samples.

The trace buffer may contain up to 7FE (2047) trace samples which are available for display by the user on request.

Tracing is discussed in detail in Section 7.

Eight Address/Data probes are provided which can be used to expand the address field or as a set of data points. They can be predefined as to te number of address bits or data bits.

1.2.8 Multi-processing

You can connect up to nine XDS units together in a serial daisy chain as illustrated in Figure 1-3. The emulators in the chain can be different processor types. When operating in the multi-processing mode, several combinations of synchronized start and stop conditions can be specified for global control of multiple-emulator operation. Specialized monitor commands allow you to track complex multi-processing operations with relative ease.

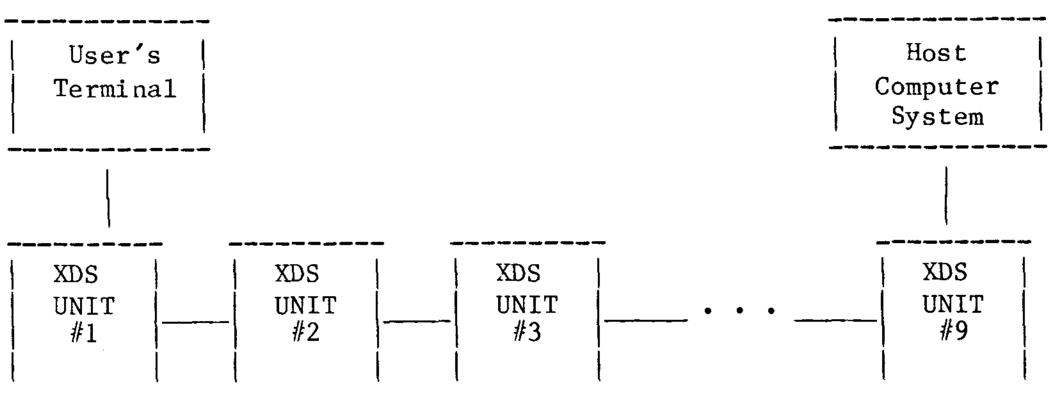


FIGURE 1-3. XDS MULTI-PROCESSING CONFIGURATION

Section 6 provides details for operating the XDS system in the multi-processing mode.

1.3 PRACTICE EXERCISE

Paragraphs 1.3.1 through 1.3.5, below, contain walk-through procedures to orient the new user of the XDS to the system and its features. You should refer to other sections in this manual for details concerning the commands that are presented here.

1.3.1 Introducing XDS to the New User

This part of the section introduces the new user of the XDS Systems to the TMS9995 Emulator. We will walk through a session to orient you to some of the commonly used features of the XDS System.

This walk-through exercise is executed in four phases:

* Phase 1 - Introduces emulator initialization and general memory utilities.
* Phase 2 - Entering and executing a simple program using memory utilities.
* Phase 3 - Hardware breakpoints and logic traces.
* Phase 4 - Entering a program using the TMS9995 Emulator Assembler.

1.3.1.1 Notation. The syntax conventions described below are used throughout this section to simplify the use of the commands.

* Angle brackets <> indicate that you must type something from the keyboard. If the text in brackets is in UPPER CASE, press the key named in the text. For example <CR> means to press the carriage return key and <SP> indicates that you should press the space bar.
* The cursor is represented by the symbol []. If the cursor is positioned over a character, it is symbolized by underlining the character (A).
* Control key sequences are represented by a CTRL followed by the second key. To enter a control-key sequence, you depress and hold the CTRL key while typing the alpha character. For example, CTRL-A represents the control-A key sequence.

1.3.1.2 Powering-Up XDS. Use the following steps to apply power to the XDS and start executing the monitor program.

1. Place the power switch to the ON position to apply power to the unit.
2. Wait approximately 5 seconds then press <CR> on the terminal keyboard twice.

3. The terminal displays a banner, a list of emulator commands, and a "?" prompt to indicate that the monitor is waiting for you to enter a monitor command.

```
                         TEXAS INSTRUMENTS

              TMS9995      XDS        VERSION 1.3.0

COMMANDS:
 INIT      IM     DR     RUN     BP      TR      HOST    IMP
 IPORT     DM     MR     CRUN    BPM     TRM     IHC     IMD
 IPRM      MM     DIO    SS      BPIO    TRIO    UL      ID
 ICC              MIO    SRR             TRIX    DL      BGND
 RCC                                     SOR
 RESTART

 MAP       FILL   XA     DPS     SSB     IT      LOG     GRUN
           FIND   XRA    DHS     DSB     DT      SNAP    TRUN
           DW            DTS     CSB             HELP    GHALT
                                 CASB            DV      THALT

VARIABLES:
 PC        R      LGT    C       INTM
 ST               AGT    OV
 WP               EQ     OP
?
```

4. You will now execute the INIT command, as instructed in Phase 1 below, to initailize the emulator. You will notice that after the command is entered, the monitor will display some prompts so that parameters associated with the command can be defined.

1.3.2 Walk-Through Excercise - Phase 1

Display: ?

Enter: INIT<CR>

Display: INIT
    EMU: CLOCK SOURCE [0=INTERNAL, 1=TARGET] = 0 []

Comment: This is an example of a parameter. The parameter name is displayed followed by acceptable values for the parameter. The current value is displayed following the equal sign. The 0 specifies that the internal clock is currently designated as the signal generator.

Enter: <CR>

Comment: When you enter the <CR>, you retain the current value of the parameter. In this case, the current value is zero and indicates that the internal clock is to be used by the emulator. If you wanted to change the value, you would type the new value before the carriage return, as follows:

Display:
```
INIT
    EMU: CLOCK SOURCE [0=INTERNAL, 1=TARGET]          = 0
         EMULATOR RAM? [0=NO, 1=YES]                  = 0 []
```

Comment: The monitor displays the next parameter.

Enter: 1<CR>

Comment: The current value (0) was changed to 1 by entering it before the carriage return. This allocates the 7K-byte block of emulator RAM to addresses 0000 through 1BFF.

Display:
```
INIT
    EMU: CLOCK SOURCE [0=INTERNAL, 1=TARGET]          = 0
         EMULATOR RAM? [0=NO, 1=YES]                  = 0  1
  EXMEM: EXPANSION PAGE [0="A", 1="B"]                = 0 []
```

Enter: <CR>

Display:
```
INIT
    EMU: CLOCK SOURCE [0=INTERNAL, 1=TARGET]          = 0
         EMULATOR RAM? [0=NO, 1=YES]                  = 0 1
  EXMEM: EXPANSION PAGE [0="A", 1="B"]                = 0
     BP: NUMBER OF EXTENDED ADDRESS BITS (0 - 8)      = 0 []
```

Enter: <CR>

Display:
```
INIT
    EMU: CLOCK SOURCE [0=INTERNAL, 1=TARGET]          = 0
         EMULATOR RAM? [0=NO, 1=YES]                  = 0 1
  EXMEM: EXPANSION PAGE [0="A", 1="B"]                = 0
     BP: NUMBER OF EXTENDED ADDRESS BITS (0 - 8)      = 0
?
```

Comment: The monitor is waiting for you to enter another command (note the "?" prompt).

Now that we have initialized the emulator, let's map the emulator memory to allocate memory space and fill some of the memory locations with data to demonstrate the MAP and FILL commands. First, the MAP command:

Enter: MAP<CR>

Display:
```
MAP
   BASE ADDRESS (1K BOUNDRY)          = 0000 []
```

Comment: The expansion memory is biased in 1K-byte blocks, enter the lower range of the 1KB block of addresses to biased into the expansion memory.

Enter: 2000<CR>

Comment: The lower range of the address block to be biased-in is 2000 (hexadecimal).

Display:
```
MAP
  BASE ADDRESS (1K BOUNDRY)              = 0000 2000
  NUMBER OF 1KB BLOCKS IN HEX (0-40) = 00   []
```

Comment: Select two 1K-byte blocks

Enter: 2<CR>

Display:
```
MAP
  BASE ADDRESS (1K BOUNDRY)              = 0000 2000
  NUMBER OF 1KB BLOCKS IN HEX (0-40) = 00   2
  [0=UNMAP, 1=ROM, 2=RAM, 3=COPY]        = 0    []
```

Comment: This parameter asks what type of memory map action is desired, UNMAP (delete block from memory map), ROM (write protect the portion of memory identified in the address parameter), RAM (use the portion of memory as normal RAM), or COPY (maps the specified block as ROM and fills it with the contents of the Target memory space). For this example, select RAM.

Enter: 2<CR>

Display:
```
MAP
  BASE ADDRESS (1K BOUNDRY)              = 0000 2000
  NUMBER OF 1KB BLOCKS IN HEX (0-40) = 00   2
  [0=UNMAP, 1=ROM, 2=RAM, 3=COPY]        = 0    2
  NUMBER OF EXTRA WAIT STATES (0-3)  = 0    []
```

Comment: Basically, the number of extra wait states (1 through 3) allows the emulator to more closely simulate the target environment. For the purposes of this demonstration, the wait states are irrelevent; enter a carriage return to retain the value of 0 (no extra wait states).

Enter: <CR>

Display:
```
MAP
  BASE ADDRESS (1K BOUNDRY)              = 0000 2000
  NUMBER OF 1KB BLOCKS IN HEX (0-40) = 00   2
  [0=UNMAP, 1=ROM, 2=RAM, 3=COPY]        = 0    2
  NUMBER OF EXTRA WAIT STATES (0-3)  = 0
?
```

The monitor is waiting for you to enter another command, in this case FILL to fill some of the memory locations with the data 2020.

```
Enter:    FILL<CR>

Display:  FILL
            START ADDRESS          = 0000 []
```

Comment The value in this parameter is the lower end of the expansion memory address range to be filled by the data specified later in this command.

```
Enter:    2000<CR>
```

Comment: The lower end of the expansion memory address range (established in the MAP command) was specified as the lower end of address range to be filled in this command.

```
Display:  FILL
            START ADDRESS          = 0000 2000
            END   ADDRESS          = 0000 []
```

Comment: The value in this parameter is the upper end of the address range to be filled by this command.

```
Enter:    2020<CR>

Display:  FILL
            START ADDRESS          = 0000 2000
            END   ADDRESS          = 0000 2020
            DATA                   = 0000 []

Enter:    2020<CR>

Display:  FILL
            START ADDRESS          = 0000 2000
            END   ADDRESS          = 0000 2020
            DATA                   = 0000 2020
          ?
```

Comment: At this point, the expansion memory locations specified are filled with 2020. We can assure ourselves of this by using another command, DM (Display Memory).

```
Enter:    DM<CR>

Display:  DM
            START ADDRESS = 0000 []

Enter:    2000<CR>

Display:  DM
            START ADDRESS = 0000 2000
            END   ADDRESS = 0000 []
```

Enter: 2020<CR>

Comment: We have requested the the monitor display expansion memory locations 2000 through 2020. The monitor will display these locations as follows:

Display:
```
DM
  START ADDRESS = 0000 2000
  END   ADDRESS = 0000 2020
    2000=2020 2020 2020 2020 2020 2020 2020 2020
    2010=2020 2020 2020 2020 2020 2020 2020 2020
    2020=2020
?
```

Comment: The contents of the locations are displayed to the right of the addresses. Since the addresses are byte-oriented, the contents are displayed in the order of locations 2000, 2002, 2004, 2006, 2008, 200A, 200C, and 200E on the first row; 2010, 2012, and so on through 201E on the second line. The third line displays only the value of location 2020.

Now let's fill expansion memory locations 2022 through 2050 with 5520 using the FILL command.

Enter: FILL<CR>

Display:
```
FILL
  START ADDRESS           = 2000 []
```

Enter: 2022<CR>

Display:
```
FILL
  START ADDRESS           = 2000 2022
  END   ADDRESS           = 2020 []
```

Enter: 2050<CR>

Display:
```
FILL
  START ADDRESS           = 2000 2022
  END   ADDRESS           = 2020 2050
  DATA                    = 2020 []
```

Enter: 5520<CR>

Display:
```
FILL
  START ADDRESS           = 2000 2022
  END   ADDRESS           = 2020 2050
  DATA                    = 2020 5520
?
```

Comment: You might notice that the values you entered earlier are displayed as the current values of the parameters (column immediately to the right of the equal sign). These remain fixed until you enter a new value as we have done with the START and END ADDRESS and DATA parameters.

You can now verify that expansion memory locations 2022 through 2050 are filled with 5520 if you like by reentering the DM command. Remember to specify the new START and END ADDRESSes for this command just as you did for the FILL command above.

There is one small difficulty you might encounter which can be eliminated at this point in our exploration of XDS. You may find that some terminal keyboards generate errors when the cursor movement keys are pressed. To avoid confusion, the ICC (Initialize Cursor Control) command is used to program the cursor movement functions using the cursor movement keys on your keyboard. Let's execute ICC now.

NOTE

If your keyboard does not have the standard UP and DOWN ARROW keys to move the cursor then do not execute ICC.

Enter: ICC<CR>

Display: ICC

```
DEPRESS KEY FOR CURSOR UP    = []
```

Enter: Press the UP-ARROW key on your terminal.

Display: ICC

```
DEPRESS KEY FOR CURSOR UP    = OK
DEPRESS KEY FOR CURSOR DOWN  = []
```

Enter: Press the DOWN-ARROW key on your terminal.

Display: ICC

```
DEPRESS KEY FOR CURSOR UP    = OK
DEPRESS KEY FOR CURSOR DOWN  = OK
DEPRESS KEY FOR CURSOR LEFT  = []
```

Enter: Press the LEFT-ARROW key on your terminal.

Display: ICC

```
DEPRESS KEY FOR CURSOR UP    = OK
DEPRESS KEY FOR CURSOR DOWN  = OK
DEPRESS KEY FOR CURSOR LEFT  = OK
DEPRESS KEY FOR CURSOR RIGHT = []
```

Enter: Press the RIGHT-ARROW key on your terminal.

```
Display: ICC
           DEPRESS KEY FOR CURSOR UP    = OK
           DEPRESS KEY FOR CURSOR DOWN  = OK
           DEPRESS KEY FOR CURSOR LEFT  = OK
           DEPRESS KEY FOR CURSOR RIGHT = []
         ?
```

Comment: The cursor control functions are now mapped into the cursor control keys on your terminal keyboard.

The ICC command provides you with another feature that allows you to edit parameter values before the command has completed its execution. If you discover an incorrect entry in the first parameter while you are entering the third parameter value, you can use the UP-ARROW to move the cursor into the data entry field of the incorrect parameter value. The data may then be corrected. We can use the FIND comand to illustrate this feature.

If the cursor control does not appear to operate properly, enter RCC<CR> and continue with the walk-through excercise.

Another utility is the FIND command which allows us to search for specific data in memory.

Enter: FIND<CR>

```
Display: FIND
           START ADDRESS            = 0000 []
```

Enter: 2028<CR>

```
Display: FIND
           START ADDRESS            = 0000 2028
           END   ADDRESS            = 0000 []
```

Enter: 2030<CR>

```
Display: FIND
           START ADDRESS            = 0000 2028
           END   ADDRESS            = 0000 2030
           DATA                     = 0000 []
```

Comment: The START and END ADDRESSes for the FIND operation are specified (note that they overlap the addresses of both FILL operations performed earlier).

Enter: 2020<CR>

Comment: The data being searched for is specified as 2020.

```
Display: FIND
           START ADDRESS               = 0000 2028
           END   ADDRESS               = 0000 2030
           DATA                        = 0000 2020
           DATA MASK (ONES ENABLE)     = FFFF []
```

Comment: If you notice that the START ADDRESS is incorrectly entered at this point you can reenter the value. The complete display will appear as shown above, the cursor is positioned in the data entry field for the DATA MASK. If you press the UP-ARROW key three times, the cursor will be positioned in the data entry field for the START ADDRESS. The display will look as follows:

```
Display: FIND
           START ADDRESS               = 0000 2028  <-- Cursor
           END   ADDRESS               = 0000 2030
           DATA                        = 0000 2020
           DATA MASK (ONES ENABLE)     = FFFF
```

You can now replace the incorrect entry of 2028 with the correct value of 2018 by pressing the RIGHT-ARROW twice and typing a 1 over the 2. Pressing the DOWN-ARROW key three times places the cursor back into the DATA MASK data entry field. Now you can resume entering the values for the FIND command.

Enter: <CR>

Comment: We are looking for an exact match between the specified data and that in memory and therefore are masking no data bits (the entry of a carriage return retains the current parameter value - FFFF).

```
Display: FIND
           START ADDRESS               = 0000 2018
           END   ADDRESS               = 0000 2030
           DATA                        = 0000 2020
           DATA MASK (ONES ENABLE)     = FFFF FFFF
             2018=2020
             201A=2020
             201C=2020
             201E=2020
             2020=2020
         ?
```

Comment: Note that only locations 2018 through 2020 were displayed instead of the 2018 through 2030 range specified. This is because only those locations contained exact matches to the data for which we had searched.

Data masking is a concept we can explore using the FIND command. You will notice that the DATA MASK parameter has a power-up value of FFFF. If we change any of the bits in the binary code for the data mask to zeroes, the monitor ignores the bit in that position when it is searching for matching data in memory. Let's see how this works using the data we entered into memory earlier.

```
Display: ?

Enter:   FIND<CR>

Display: FIND
           START ADDRESS          = 2018 []

Enter:   <CR>

Display: FIND
           START ADDRESS          = 2018
           END   ADDRESS          = 2030 []

Enter:   <CR>

Display: FIND
           START ADDRESS          = 2018
           END   ADDRESS          = 2030
           DATA                   = 2020 []

Enter:   <CR>
```

Comment: We have retained the last values entered for START ADDRESS, END ADDRESS, and DATA.

```
Display: FIND
           START ADDRESS          = 2018
           END   ADDRESS          = 2030
           DATA                   = 2020
           DATA MASK (ONE'S ENABLE)= FFFF []

Enter:   00FF<CR>
```

Comment: At this point we have changed the DATA MASK so that the first eight bits are ignored during the search for data fields that match 2020.

```
Display: FIND
           START ADDRESS          = 2018
           END   ADDRESS          = 2030
           DATA                   = 2020
           DATA MASK (ONE'S ENABLE)= FFFF 00FF
             2018=2020
             201A=2020
             201C=2020
             201E=2020
             2020=2020
             2022=5520
             2024=5520
             2026=5520
             2028=5520
             202A=5520
             202C=5520
             202E=5520
             2030=5520
         ?
```

So far, we have been working with Expansion Memory Page "A" (in the INIT command we selected zero, page "A", in response to the prompt EXMEM: EXPANSION PAGE [0="A", 1="B"]). We can illustrate this by displaying the contents of memory page "A" then those of memory page "B". One aspect of the use of two different memory pages might be for the storage of two different programs in memory, one in memory page "A" and the other in memory page "B". The user could then switch back-and-forth between the pages (programs) as required.

Display: ?

Enter: DM<CR>

```
Display: DM
           START ADDRESS = 2000 []
```

Enter: <CR>

Comment: The START ADDRESS is set to a previous value of 2000 or whatever the last entry was to the data field for that parameter. A carriage return was entered to retain that value.

```
Display: DM
           START ADDRESS = 2000
           END   ADDRESS = 2020 []
```

Enter: 2050<CR>

Comment: We changed the value (from the last entry in this data field) from 2020 to 2050.

Display:

```
DM
  START ADDRESS = 2000
  END   ADDRESS = 2020 2050
    2000=2020 2020 2020 2020 2020 2020 2020 2020
    2010=2020 2020 2020 2020 2020 2020 2020 2020
    2020=2020 5520 5520 5520 5520 5520 5520 5520    U U U U U U U
    2030=5520 5520 5520 5520 5520 5520 5520 5520  U U U U U U U U
    2040=5520 5520 5520 5520 5520 5520 5520 5520  U U U U U U U U
    2050=5520                                     U
?
```

Comment: The specified addresses and their contents are displayed below the parameter prompts. Decoded ASCII characters (U, from the hexadecimal 55's, in this case) are displayed to the right of the addresses/data.

We will now select Expansion Memory Page "B" by returning to the INIT command.

Display: ?

Enter: INIT<CR>

Display:
```
INIT
     EMU: CLOCK SOURCE [0=INTERNAL, 1=TARGET]        = 0 []
```

Enter: <CR>

Display:
```
INIT
     EMU: CLOCK SOURCE [0=INTERNAL, 1=TARGET]        = 0
          EMULATOR RAM? [0=NO, 1=YES]                = 1 []
```

Enter: <CR>

Comment: The current values are accepted by entering carriage returns for both prompts.

Display:
```
INIT
     EMU: CLOCK SOURCE [0=INTERNAL, 1=TARGET]        = 0
          EMULATOR RAM? [0=NO, 1=YES]                = 1
   EXMEM: EXPANSION PAGE [0="A", 1="B"]              = 0 []
```

Enter: 1<CR>

Comment: A one was entered to select the working memory page ("B").

Display:
```
INIT
     EMU: CLOCK SOURCE [0=INTERNAL, 1=TARGET]        = 0
          EMULATOR RAM? [0=NO, 1=YES]                = 1
   EXMEM: EXPANSION PAGE [0="A", 1="B"]              = 0 1
      BP: NUMBER OF EXTENDED ADDRESS BITS (0 - 8)    = 0 []
```

```
    Enter:   <CR>

    Display: INIT
                EMU: CLOCK SOURCE [0=INTERNAL, 1=TARGET]          = 0
                     EMULATOR RAM? [0=NO, 1=YES]                  = 1
              EXMEM: EXPANSION PAGE [0="A", 1="B"]                = 0 1
                 BP: NUMBER OF EXTENDED ADDRESS BITS (0 - 8)      = 0
             ?
```

In order to view the contents of memory page "B", we need to map it into memory.

```
    Display: ?

    Enter:   MAP<CR>

    Display: MAP
               BASE ADDRESS (1K BOUNDRY)             = 2000 []

    Enter:   <CR>

    Display: MAP
               BASE ADDRESS (1K BOUNDRY)             = 2000
               NUMBER OF 1KB BLOCKS IN HEX (0-40) = 02   []

    Enter:   <CR>

    Display: MAP
               BASE ADDRESS (1K BOUNDRY)             = 2000
               NUMBER OF 1KB BLOCKS IN HEX (0-40) = 02
               [0=UNMAP, 1=ROM, 2=RAM, 3=COPY]     = 2     []

    Enter:   <CR>

    Comment: The memory Page "B" space has been mapped as RAM.

    Display: MAP
               BASE ADDRESS (1K BOUNDRY)             = 2000
               NUMBER OF 1KB BLOCKS IN HEX (0-40) = 02
               [0=UNMAP, 1=ROM, 2=RAM, 3=COPY]     = 2
               NUMBER OF EXTRA WAIT STATES          = 0     []

    Enter:   <CR>

    Display: MAP
               BASE ADDRESS (1K BOUNDRY)             = 2000
               NUMBER OF 1KB BLOCKS IN HEX (0-40) = 02
               [0=UNMAP, 1=ROM, 2=RAM, 3=COPY]     = 2
               NUMBER OF EXTRA WAIT STATES          = 0
             ?
```

Now we can execute the Display Memory command (DM) to view the contents of memory Page "B".

Display: ?

Enter: DM<CR>

Display: DM
```
START ADDRESS = 2000 []
```

Enter: <CR>

Display: DM
```
START ADDRESS = 2000
END   ADDRESS = 2050 []
```

Enter: <CR>

Comment: We want to examine the same addresses from memory page "B" as were viewed from memory page "A", carriage returns were entered to retain the previous values.

Display:
```
DM
  START ADDRESS = 2000
  END   ADDRESS = 2050
    2000=0000 0000 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
    2010=0000 0000 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
    2020=0000 0000 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
    2030=0000 0000 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
    2040=0000 0000 0000 0000 0000 0000 0000 0000  .. .. .. .. .. .. .. ..
    2050=0000                                     ..
```

Comment: The memory page "B" addresses and their contents are displayed. All of the addresses contain zeroes, which are the power-up values, since no data has yet been entered into memory page "B". Note that periods are displayed to the right of the address/data display, this is because non-printable ASCII values are displayed as periods. If any of the displayed addresses did contain data that represented ASCII characters, those characters would be displayed.

Before we continue, FILL some of the emulator memory with NOP code 1000.

Enter: FILL<CR>

Display: FILL
```
START ADDRESS          = 2022 []
```

Enter: 0000<CR>

```
Display: FILL
           START ADDRESS          = 2022 0000
           END   ADDRESS          = 2050 []

Enter:    0102<CR>

Display: FILL
           START ADDRESS          = 2022 0000
           END   ADDRESS          = 2050 0102
           DATA                   = 5520 []

Enter:    1000<CR>

Display: FILL
           START ADDRESS          = 2022 0000
           END   ADDRESS          = 2050 0102
           DATA                   = 5520 1000
         ?
```

Now, locations 0000 through 0102 of the emulator memory have been filled with NOP codes. You are now ready to enter a simple program to illustrate some more features of XDS.

1.3.3 Walk-Through Excercise - Phase 2

We will use the IM (Inspect Memory) command to enter the program object code from your terminal keyboard.

```
Enter:    IM<CR>

Display: IM
            ADDRESS = 0000 []

Enter:    A<CR>
```

Comment: The object code will be entered in the locations starting with address 000A.

```
Display: IM
            ADDRESS = 0000 A
              000A=1000 []

Enter:    04C4<CR>
```

Comment: After IM is entered, the current value of the ADDRESS parameter is displayed. When you entered the A into ADDRESS, you indicated that the first line of the program will be in location 000A. The <CR> causes the monitor to step to the next location which is displayed for input.

```
Display: IM
           ADDRESS = 0000 A
             000A=1000 04C4
            +000C=1000 []
```

Enter: 0201<CR>

```
Display: IM
           ADDRESS = 0000 A
             000A=1000 04C4
            +000C=1000 0201
            +000E=1000 []
```

Enter: 000A<CR>

```
Display: IM
           ADDRESS = 0000 A
             000A=1000 04C4
            +000C=1000 0201
            +000E=1000 000A
            +0010=1000 []
```

Enter: 0202<CR>

As you can see, the new entry appears in the third column as previously noted. The address is shown in the first column and the current value is in the second column. To facilitate entry of the data the remaining entries are listed below without the Display and Entry sequences used so far.

Address	Data
+0012	1234
+0014	0203
+0016	0010
+0018	38C2
+001A	C803
+001C	0100
+001E	C804
+0020	0102
+0022	C0C1
+0024	0601
+0026	16F8
+0028	10F0

Enter: Q<CR>

Entering the Q indicates that you are finished with the IM command. At this point, the program we will use for illustrating many features of the XDS has been entered into memory. Let's display the memory contents to check the code for errors.

Enter: DM<CR>

```
    Display: DM
                START ADDRESS = 2000 []

    Enter:   A<CR>

    Display: DM
                START ADDRESS = 2000 A
                END   ADDRESS = 2050 []

    Enter:   28<CR>

    Display:
DM
  START ADDRESS = 2000 A
  END   ADDRESS = 2050 28
    000A=04C4 0201 000A 0202 1234 0203 0010 38C2  .. .. .. .. .4 .. .. 8.
    001A=C803 0100 C804 0102 COC1 0601 16F8 10F0  .. .. .. .. .. .. .. ..
?
```

Now we can list the program in reverse assembled format by using the XRA (Execute Reverse Assembler) command.

```
    Enter:   XRA<CR>

    Display: XRA
                START ADDRESS          = 0000 []

    Enter:   A<CR>

    Display: XRA
                START ADDRESS          = 0000 A
                NUMBER OF INSTRUCTIONS = 0000 []

    Enter:   B<CR>

    Display: XRA
                START ADDRESS          = 0000 A
                NUMBER OF INSTRUCTIONS = 0000 B
                  000A 04C4  CLR  R4
                  000C 0201  LI   R1,>000A
                  0010 0202  LI   R2,>1234
                  0014 0203  LI   R3,>0010
                  0018 38C2  MPY  R2,R3
                  001A C803  MOV  R3,@>0100
                  001E C804  MOV  R3,@>0102
                  0022 COC1  MOV  R1,R3
                  0024 0601  DEC  R1
                  0026 16F8  JNE  >0018
                  0028 10F0  JMP  >000A
             ?
```

Before you execute the program using the RUN command, we need to set up the Program Counter so that execution will begin at location 000A where the first instruction is located. This is done using the MR (Modify Registers) command.

```
Enter:   MR<CR>

Display: MR
           WP = 0000 []

Enter:   50<CR>

Display: MR
           WP = 0000 50
           PC = 0000 []

Enter:   A<CR>

Display: MR
           WP = 0000 50
           PC = 0000 A
           ST = 0000 []

Enter:   <CR>

Display: MR
           WP = 0000 50
           PC = 0000 A
           ST = 0000
         ?
```

Comment: When you entered the MR command and the carriage return, the first prompt to appear on the screen was for the Workspace Pointer register, which we set to 50, this set the address of the first Workspace Register (register R0) to location 0050. Registers R1 through R15 are located in the next successively higher locations. The second prompt was for the Program Counter, which we set to 000A (the location with the first instruction code). The last prompt was for the Status Register which we accepted as 0000 by entering a carriage return.

You can display the current contents of the registers by executing the DR (Display Registers) command.

Enter: DR<CR>

Display: DR

```
         WP=0050  PC=000A  ST=0000
```

The flowchart in Figure 1-2 illustrates what this program does. Very briefly, the program that starts at location 000A initializes Workspace Registers R1 - R4, multiplies the contents of registers R2 and R3, moves the values in registers R3 and R4 to locations 0100 and 0102 (respectively) moves the contents of register R1 to R3, decrements register R1 by one and repeats the multiply and move processes until the contents of register R1 are equal to zero at which time it starts the process over again. (Program execution halts when the <ESC> key on the keyboard is pressed.)

The SS (Single Step Execution) command allows you to execute the program one line at a time. Let's enter that command now.

Enter: SS<CR>

Display: SS

```
         WP=0050 PC=000C ST=0000  NEXT: 000C 0201  LI  R1,>000A
```

This command executes one instruction of the program and causes the display of the Workspace Pointer, Program Counter, and Status Registers as each instruction is executed; the data in this part of the display are the current values in these registers. To the right are displayed the next location, the data in that location, and the assembly instruction to be executed.

Enter: <SP>

Comment: Entering a space reexecutes the current Monitor Command stored in the input buffer (SS, in this case).

Display: SS

```
         WP=0050 PC=0010 ST=C000  NEXT: 0010 0202  LI  R2,>1234
```

Enter: DW<CR>

Display: DW

```
        R0 =  1000  R4 =  0000  R8  =  1000  R12 =  1000
        R1 =  000A  R5 =  1000  R9  =  1000  R13 =  1000
        R2 =  1000  R6 =  1000  R10 =  1000  R14 =  1000
        R3 =  1000  R7 =  1000  R11 =  1000  R15 =  1000
     ?
```

```
                               A
                             START
                               |
                               |
                               V
                   ----------------------------
                   |  Initialize Workspace    |
                   |  Registers (R1 - R4)     |
     INIT          |       (Clear R4,         |
|----------------->|  Load R1 with >000A,     |
|                  |  Load R2 with >1234,     |
|                  |  Load R3 with >0010)     |
|                  ----------------------------
|                              |
|                              |
|                              V
|          LOOP    ----------------------------
|        -------->|        Multiply          |
|        |        |        R2 X R3           |
|        |        ----------------------------
|        |                     |
|        |                     |
|        |                     V
|        |        ----------------------------
|        |        |          Move            |
|        |        | R3 to Memory Location    |
|        |        |          >0100           |
|        |        ----------------------------
|        |                     |
|        |                     |
|        |                     V
|        |        ----------------------------
|        |        |          Move            |
|        |        | R4 to Memory Location    |
|        |        |          >0102           |
|        |        ----------------------------
|        |                     |
|        |                     |
|        |                     V
|        |        ----------------------------
|        |        |          Move            |
|        |        |        R1 to R3          |
|        |        ----------------------------
|        |                     |
|        |                     |
|      N |                     V
|        ^        ----------------------------
|  Y              |        Decrement         |
|---< R1 = 0 >----|           R1             |
                  ----------------------------
         v
```

Figure 1-2. Flowchart for Example Program

Comment: By executing the DW (Display Workspace) command, we can view the current values of the Workspace Registers. Notice that the values in the registers are the NOP codes with which we FILLed memory earlier and that by executing the first two instructions in our program we have reset register R4 to zeroes (first instruction) then loaded the value 000A into register R1.

The TMS9995 Emulator allows us to enter more than one command at a time so that they may be executed sequentially. We can take advantage of this now and at the same time we will fix the display on the screen using the SNAP command so that the results of each instruction can be viewed. This command can be used only as the first in a string of commands entered sequentially. When you enter commands in this manner, they can be separated either by spaces or commas. For readability, we will use commas.

NOTE

The TMS9995 Emulator contains two input buffers, one is 60 characters in length and the other is twenty. Be sure that you are working in the large buffer if you are entering more than twenty characters. If the emulator refuses to accept input on the twentieth character, you are in the small buffer. Press either the "+" or "-" key to change to the large buffer and repeat the entry.

Enter: SNAP,SS,DW,DM(100,102)<CR>

Comment: The sequential commands we are entering are SNAP (explained above), SS, DW (Display Workspace Registers), and DM (Display Memory - locations 0100 and 0102). The execution of these commands will freeze the display on the screen (SNAP), allow Single-Step execution of the instructions, display the Workspace Register contents, and display memory locations 0100 and 0102.

NOTE

The ICC (Initialize Cursor Control) command must be executed before SNAP will work (accomplished earlier in this walk-through). If you weren't able to use the ICC command to initialize your terminal's cursor control, the SNAP command will not operate properly. Reenter the command without SNAP to continue with the walk-through

Display:

```
SNAP,SS,DW,DM(100,102)
 SS
   WP=0050 PC=0014 ST=C000  NEXT: 0014 0203  LI  R3,>0010
 DW
   R0 =  1000  R4 =  0000  R8  =  1000  R12 = 1000
   R1 =  000A  R5 =  1000  R9  =  1000  R13 = 1000
   R2 =  1234  R6 =  1000  R10 =  1000  R14 = 1000
   R3 =  1000  R7 =  1000  R11 =  1000  R15 = 1000
 DM
   0100=1000 1000                               .. ..
?
```

If you simply press the space bar at this point, the commands that have been entered will be executed again. Try it.

Enter: <SP>

Display:

```
SNAP,SS,DW,DM(100,102)
 SS
   WP=0050 PC=0018 ST=C000  NEXT: 0018 38C2  MPY R2,R3
 DW
   R0 =  1000  R4 =  0000  R8  =  1000  R12 = 1000
   R1 =  000A  R5 =  1000  R9  =  1000  R13 = 1000
   R2 =  1234  R6 =  1000  R10 =  1000  R14 = 1000
   R3 =  0010  R7 =  1000  R11 =  1000  R15 = 1000
 DM
   0100=1000 1000                               .. ..
?
```

Try the space bar one more time.

Display:

```
SNAP,SS,DW,DM(100,102)
 SS
   WP=0050 PC=001A ST=C000  NEXT: 001A C803  MOV R3,@>0100
 DW
   R0 =  1000  R4 =  2340  R8  =  1000  R12 = 1000
   R1 =  000A  R5 =  1000  R9  =  1000  R13 = 1000
   R2 =  1234  R6 =  1000  R10 =  1000  R14 = 1000
   R3 =  0001  R7 =  1000  R11 =  1000  R15 = 1000
 DM
   0100=1000 1000                               .. ..
?
```

You will note that the display of the registers remains fixed on the screen and no longer scrolls up the screen as it did before. This is the result of the SNAP command. The values in the registers change and thus you can watch these values change as the program executes without the problem of a moving display.

Another powerful feature of the TMS9995 Emulator Monitor is the repeat function. If we terminate a command or a string of commands with an asterisk (*) before the <CR>, the command, or string of commands, will execute repeatedly until we press <ESC>. Let's try this and see how it works. This time we can bring the contents of the input buffer back and just add the * to the end of the command string.

Enter: ?

Comment: Entering the question mark displays the contents of the input buffer which can then be edited.

Display: SNAP,SS,DW,DM(100,102)

Comment: Now move the cursor to the right so that it is positioned over the space following the last character in the command string and type the * followed by a <CR>.

Display:

```
SNAP,SS,DW,DM(100,102)*
 SS
   WP=0050 PC=____ ST=____  NEXT: ____ ____ ____ _________
 DW
   R0 =  1000  R4 =  ____  R8  =  1000  R12 =  1000
   R1 =  ____  R5 =  1000  R9  =  1000  R13 =  1000
   R2 =  ____  R6 =  1000  R10 =  1000  R14 =  1000
   R3 =  ____  R7 =  1000  R11 =  1000  R15 =  1000
 DM
   0100=____ ____                              __ __
```

Comment: The command string continues to execute until the <ESC> key is pressed. The fields that are underlined in the example display above are constantly being updated with the results of the execution of the program and for the purposes of this walk-through are shown underlined since a given value will not be present all of the time.

You can see the value of the repeat function for other purposes such as scope-looping in addition to the constantly updated fixed display.

Enter: <ESC>

Display: ?

Of course, one useful debugging tool that the TMS9995 Emulator provides is the software breakpoint. To use this feature all you need to do is to enter the location where the break is to occur using SSB (Set Software Breakpoint) command.

```
Enter:    SSB<CR>

Display: SSB
            BREAKPOINT ADDRESS = 0000 []

Enter:    C<CR>

Display: SSB
            BREAKPOINT ADDRESS = 0000 C
         ?
```

Now you have set a breakpoint at location 000C so that when the instruction in location 000C is executed, the program execution will halt. You have the option of setting up to ten software breakpoints at one time. To see what breakpoint locations have been set you must execute the DSB (Display Software Breakpoint) command.

```
Enter:    DSB<CR>

Display: DSB
             000C
         ?
```

Now let's enter some more software breakpoints.

```
Enter:    SSB<CR>

Display: SSB
            BREAKPOINT ADDRESS = 000C []

Enter:    24<CR>

Display: SSB
            BREAKPOINT ADDRESS = 000C 24
         ?

Enter:    SSB<CR>

Display: SSB
            BREAKPOINT ADDRESS = 0024 []

Enter:    26<CR>

Display: SSB
            BREAKPOINT ADDRESS = 0024 26

Enter:    SSB<CR>

Display: SSB
            BREAKPOINT ADDRESS = 0026 []
```

```
    Enter:   28<CR>

    Display: SSB
                BREAKPOINT ADDRESS = 0026 28
```

We can again display the software breakpoints.

```
    Enter:   DSB<CR>

    Display: DSB
                 000C 0024 0026 0028
             ?
```

As you can see, we have defined four software breakpoint addresses.

Before issuing the command to run, we first must reset the program counter to its initial value using the MR (Modify Register) command. However, rather than executing the command again, the Monitor provides a second method of executing the command without having to reenter the values for each register as we did before. You simply enter the command followed by a period (.) before the <CR> and the register values will be set to the same values that were entered the last time he MR command was executed. Let's try this feature now.

```
    Enter:   MR.RUN<CR>

    Display: MR.RUN
              MR
               WP=0050
               PC=000A
               ST=0000
             RUN
                RUNNING
                SBP
                WP=0050  PC=000C  ST=0000
             ?
```

The MR command is executed first and all of the current values for the parameters (the registers in this command) are displayed. Then the RUN command is executed causing the program to execute. When the instruction in location 000C is executed, program execution is halted and the SBP message is displayed. This indicates that the reason the execution stopped was that a Software Breakpoint was encountered. The current contents of the Workspace Pointer, Program Counter, and Status Registers are displayed below the SBP message.

When a Software Breakpoint is encountered, and program execution is halted, the Software Breapoint that caused the halt is cleared. This can be illustrated by executing the DSB command at this time.

Enter: DSB<CR>

Display: DSB
0024 0026 0028
?

Comment: Only the Software Breakpoint Addresses 0024, 0026, and 0028 are displayed now since the one at location 000C has already been encountered.

If you enter the RUN command now, the program resumes execution at the current PC location (000C).

Enter: RUN<CR>

Display: RUN
RUNNING
SBP
WP=0050 PC=0024 ST=C000
?

If you have no more use for a software breakpoint, you can clear it by using either the CSB (Clear Software Breakpoint) or the CASB (Clear All Software Breakpoints) command. Obviously, the CASB command will eliminate all software breakpoints, while the CSB command is used to clear a single Software Breakpoint Address.

Enter: CSB<CR>

Display: CSB
BREAKPOINT ADDRESS = 0028 []

Comment: The value displayed in the prompt is the last Breakpoint Address that was entered (in this case, in the last SSB command).

Enter: 26<CR>

Display: CSB
BREAKPOINT ADDRESS = 0028 26
?

Enter: DSB<CR>

Display: DSB
0028
?

Enter: CASB<CR>

Display: CASB
?

Enter: DSB<CR>

```
Display: DSB
         ?
```

If you enter RUN along with the MR command again, the program will run indefinitely, reseting the registers with each cycle and starting over again, until you stop it by entering any character from the keyboard.

```
Enter:   MR;RUN<CR>

Display: RUN
            RUNNING

Enter:   <any keystroke>

Display: RUN
            RUNNING
            KEY
            WP=0050  PC=0022  ST=D000
```

Comment: The values shown in the Program Counter and Status registers in this display are purely arbitrary and may not agree with your display since the actual values will reflect the register contents when you pressed a key.

You may notice that we used a semicolon (;) instead of a period after the MR command when we entered the string. This operates in the same way as the period except that the display of the parameters (the registers in this case) is suppressed.

When a character is entered from the keyboard, program execution is interrupted and the KEY message is displayed to indicate the reason for the interruption.

1.3.4 Walk-Through Exercise - Phase 3

Another valuable tool provided by the XDS System is trace and hardware breakpoint capability. There are several monitor commands that are used to set up qualifying criteria for both traces and hardware breakpoints. These are discussed in detail in Section 7.

We will continue our walk-through excercise to demonstrate the trace and breakpoint conditions.

```
Display: ?

Enter:   TR<CR>

Display: TR
           TRACE COUNT (1 - 7FF, 0=INFINITE)       = 000  []
```

```
Enter:    20<CR>

Display: TR
            TRACE COUNT (1 - 7FF, 0=INFINITE)          = 000  20
            ADDRESS MASK (ONES ENABLE)                 = FFFF []

Enter:    <CR>

Dispaly: TR
            TRACE COUNT (1 - 7FF, 0=INFINITE)          = 000  20
            ADDRESS MASK (ONES ENABLE)                 = FFFF
            TRACE MODE [0=TRM & TRIO, 1=TRIX]          = 0    []

Enter:    <CR>

Display: TR
            TRACE COUNT (1 - 7FF, 0=INFINITE)          = 000  20
            ADDRESS MASK (ONES ENABLE)                 = FFFF
            TRACE MODE [0=TRM & TRIO, 1=TRIX]          = 0
```

The TRACE COUNT specifies the number of traces that will be taken; we asked for 20 samples. The ADDRESS MASK is fully enabled (all mask bits are set to ones), this means that only those addresses that exactly match our compare values will qualify as trace samples.

```
Display: ?

Enter:    TRM<CR>

Display: TRM
            QUALIFIER [ OFF, MA, MR, MW, IAQ ]   = 0     []

Enter:    MA<CR>
```

Comment: By entering the MA selection you have indicated that any memory access operations may qualify as trace samples.

```
Display: TRM
            QUALIFIER [ OFF, MA, MR, MW, IAQ ]   = 0     MA
            TRACE ADDRESS #1                     = 0000  []

Enter:    <CR>

Display: TRM
            QUALIFIER [ OFF, MA, MR, MW, IAQ ]   = 0     MA
            TRACE ADDRESS #1                     = 0000
            TRACE ADDRESS #2                     = 0000  []

Enter:    1000<CR>
```

```
Display: TRM
           QUALIFIER [ OFF, MA, MR, MW, IAQ ]    = 0      MA
           TRACE ADDRESS #1                      = 0000
           TRACE ADDRESS #2                      = 0000   1000
           RANGE INDICATOR [0=NO, 1=YES]         = 0      []

Enter:   1<CR>

Display: TRM
           QUALIFIER [ OFF, MA, MR, MW, IAQ ]    = 0      MA
           TRACE ADDRESS #1                      = 0000
           TRACE ADDRESS #2                      = 0000   1000
           RANGE INDICATOR [0=NO, 1=YES]         = 0      1
           EMU DATA COMPARE WORD                 = 0000   []

Enter:   <CR>

Display: TRM
           QUALIFIER [ OFF, MA, MR, MW, IAQ ]    = 0      MA
           TRACE ADDRESS #1                      = 0000
           TRACE ADDRESS #2                      = 0000   1000
           RANGE INDICATOR [0=NO, 1=YES]         = 0      1
           EMU DATA COMPARE WORD                 = 0000
           EMU DATA COMPARE MASK (ONES ENABLE)   = 0000   []

Enter:   <CR>

Display: TRM
           QUALIFIER [ OFF, MA, MR, MW, IAQ ]    = 0      MA
           TRACE ADDRESS #1                      = 0000
           TRACE ADDRESS #2                      = 0000   1000
           RANGE INDICATOR [0=NO, 1=YES]         = 0      1
           EMU DATA COMPARE WORD                 = 0000
           EMU DATA COMPARE MASK (ONES ENABLE)   = 0000
           EXT DATA COMPARE BYTE                 = 00     []

Enter:   <CR>

Display: TRM
           QUALIFIER [ OFF, MA, MR, MW, IAQ ]    = 0      MA
           TRACE ADDRESS #1                      = 0000
           TRACE ADDRESS #2                      = 0000   1000
           RANGE INDICATOR [0=NO, 1=YES]         = 0      1
           EMU DATA COMPARE WORD                 = 0000
           EMU DATA COMPARE MASK (ONES ENABLE)   = 0000
           EXT DATA COMPARE BYTE                 = 00
           EXT DATA COMPARE MASK (ONES ENABLE)   = 00     []

Enter:   <CR>
```

```
Display: TRM
           QUALIFIER [ OFF, MA, MR, MW, IAQ ]     = 0      MA
           TRACE ADDRESS #1                       = 0000
           TRACE ADDRESS #2                       = 0000   1000
           RANGE INDICATOR [0=NO, 1=YES]          = 0      1
           EMU DATA COMPARE WORD                  = 0000
           EMU DATA COMPARE MASK (ONES ENABLE)    = 0000
           EXT DATA COMPARE BYTE                  = 00
           EXT DATA COMPARE MASK (ONES ENABLE)    = 00
         ?
```

We have now configured the trace mode to accept trace samples on all memory accesses that occur within the range of address 0000 and 1000 all of the expansion memory. Now let's run the program and look at a display of the trace and see what is happening.

```
Enter:    MR;RUN<CR>

Display: RUN
            RUNNING
            TMF
            WP=0050  PC=0018  ST=D000
         ?
```

Comment: The TMF message informs us that the specified number of trace samples were taken which results in a program interruption.

At this point we can display the trace memory using the IT (Inspect Trace) command.

```
Enter:    IT<CR>

Display: IT
           FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000 []

Enter:    <CR>
```

Display:

```
IT
 FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000

   INDEX CYCLE QUAL  EXTQUALS     ADDR DATA  RVRS ASSEMBLY

   0000        IAQ   11111111     000A 04C4  CLR  R4
   0001        IAQ   11111111     000C 0201  LI   R1,>000A
   0002        MW    11111111     0058 0000
   0003        MR    11111111     000E 000A
   0004        IAQ   11111111     0010 0202  LI   R2,>1234
   0005        MW    11111111     0052 000A
   0006        MR    11111111     0012 1234
   0007        IAQ   11111111     0014 0203  LI   R3,>0010
   0008        MW    11111111     0054 1234
   0009        MR    11111111     0016 0010
   000A        IAQ   11111111     0018 38C2  MPY  R2,R3
   000B        MW    11111111     0056 0010
   000C        MR    11111111     0054 1234
   000D        MR    11111111     0056 0010
   000E        MW    11111111     0056 0001
   000F        IAQ   11111111     001A C803  MOV  R3,@>0100
   0010        MW    11111111     0058 2340
   0011        MR    11111111     0056 0001
   0012        MR    11111111     001C 0100

[]
```

We can view more trace samples if we so desire by entering the number of samples by which the display is to be incremented.

Enter: +13<CR>

Display:

```
INDEX CYCLE QUAL  EXTQUALS     ADDR DATA  RVRS ASSEMBLY

000D        MR    11111111     0056 0010
000E        MW    11111111     0056 0001
000F        IAQ   11111111     001A C803  MOV  R3,@>0100
0010        MW    11111111     0058 2340
0011        MR    11111111     0056 0001
0012        MR    11111111     001C 0100
0013        IAQ   11111111     001E C804  MOV  R4,@>0102
0014        MW    11111111     0100 0001
0015        MR    11111111     0058 2340
0016        MR    11111111     0020 0102
0017        IAQ   11111111     0022 C0C1  MOV  R1,R3
0018        MW    11111111     0102 2340
0019        MR    11111111     0052 000A
001A        IAQ   11111111     0024 0601  DEC  R1
001B        MW    11111111     0056 000A
001C        MR    11111111     0052 000A
001D        IAQ   11111111     0026 16F8  JNE  >0018
001E        MW    11111111     0052 0009
001F        IAQ   11111111     0018 38C2  MPY  R2,R3
```

[]

Comment: You will notice that although you requested the next >13 samples to be displayed, samples 000D through 0012 are repeated with samples 0013 through 001F shown below them. This occurred because 20 traces were requested in the TRACE COUNT parameter of the TR command, the full display is >13 lines long, and the full display is shown each time (causing the first and second displays to overlap). If you were to enter "-13<CR>", the original >13 samples would be displayed again. You must enter Q to quit the IT command.

Enter: Q

We can display the trace status by using the DTS (Display Trace Status) command.

Enter: DTS<CR>

Display: DTS

```
    TRACE COUNT    EVENTS LEFT    DELAYS LEFT
       0020           0000           0000
?
```

We can set up hardware breakpoints to see what effect this will have on our trace. To conserve space, only newly displayed information will be presented rather than the complete display. We begin defining breakpoint criteria by executing the BP (BreakPoint) command.

Enter: BP<CR>

Display: EVENT COUNT (0 - 7FFF) = 0000 []

Enter: 5<CR>

Comment: You entered a new value of 5 for the breakpoint EVENT COUNT.

Display: DELAY COUNT (0 - 7FF) = 000 []

Enter: 20<CR>

Comment: You have set the DELAY COUNT to 20.

Display: ADDRESS MASK (ONES ENABLE) = FFFF []

Enter <CR>

Now that the BP commands parameters have been defined, we can proceed to set other breakpoint criteria using the BPM (BreakPoint Memory) command.

Enter: BPM<CR>

Display: QUALIFIER [OFF, MA, MR, IAQ] = 0 []

Enter: MA<CR>

Comment: You have designated the Memory Access option for one qualifying criterion for a breakpoint.

Display: BREAKPOINT ADDRESS #1 = 0000 []

Enter: 18<CR>

Display: BREAKPOINT ADDRESS #2 = 0000 []

Enter: 1E<CR>

Display: RANGE INDICATOR [0=NO, 1=YES] = 0 []

Enter: 1<CR>

Comment: By turning the RANGE INDICATOR ON, we have selected the range of addresses with ADDRESS #1 as the lower boundary and ADDRESS #2 as the upper boundary. Those locations within these bounds qualify as criteria for breakpoint events.

```
Display: EMU DATA COMPARE WORD                  = 0000  []

Enter:   <CR>

Display: EMU DATA COMPARE MASK (ONES ENABLE)    = 0000 []

Enter:   <CR>

Display: EXT DATA COMPARE BYTE                  = 00   []

Enter:   <CR>

Display: EXT DATA COMPARE MASE (ONES ENABLE)    = 00   []

Enter:   <CR>
```

Now that the breakpoint and trace criteria are defined, let's run our program and check our trace display.

```
Enter:   MR;RUN<CR>

Display: RUN
            RUNNING
            TMF
            WP=0050  PC=0018  ST=D000
         ?

Enter:   IT<CR>

Display: FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000

Enter:   <CR>
```

Display:

```
IT
 FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000

   INDEX CYCLE QUAL  EXTQUALS     ADDR DATA  RVRS ASSEMBLY

   0000        IAQ   11111111     000A 04C4  CLR  R4
   0001        IAQ   11111111     000C 0201  LI   R1,>000A
   0002        MW    11111111     0058 0000
   0003        MR    11111111     000E 000A
   0004        IAQ   11111111     0010 0202  LI   R2,>1234
   0005        MW    11111111     0052 000A
   0006        MR    11111111     0012 1234
   0007        IAQ   11111111     0014 0203  LI   R3,>0010
   0008        MW    11111111     0054 1234
   0009        MR    11111111     0016 0010
   000A  EVNT  IAQ   11111111     0018 38C2  MPY  R2,R3
   000B        MW    11111111     0056 0010
   000C        MR    11111111     0054 1234
   000D        MR    11111111     0056 0010
   000E        MW    11111111     0056 0001
   000F  EVNT  IAQ   11111111     001A C803  MOV  R3,@>0100
   0010        MW    11111111     0058 2340
   0011        MR    11111111     0056 0001
   0012  EVNT  MR    11111111     001C 0100
[]
```

Enter: <CR>

Display:

```
   INDEX CYCLE QUAL  EXTQUALS     ADDR DATA  RVRS ASSEMBLY

   000D        MR    11111111     0056 0010
   000E        MW    11111111     0056 0001
   000F  EVNT  IAQ   11111111     001A C803  MOV  R3,@>0100
   0010        MW    11111111     0058 2340
   0011        MR    11111111     0056 0001
   0012  EVNT  MR    11111111     001C 0100
   0013  EVNT  IAQ   11111111     001E C804  MOV  R4,@>0102
   0014        MW    11111111     0100 0001
   0015        MR    11111111     0058 2340
   0016        MR    11111111     0020 0102
   0017        IAQ   11111111     0022 C0C1  MOV  R1,R3
   0018        MW    11111111     0102 2340
   0019        MR    11111111     0052 000A
   001A        IAQ   11111111     0024 0601  DEC  R1
   001B        MW    11111111     0056 000A
   001C        MR    11111111     0052 000A
   001D        IAQ   11111111     0026 16F8  JNE  >0018
   001E        MW    11111111     0052 0009
   001F  *EVT  IAQ   11111111     0018 38C2  MPY  R2,R3
[]
```

Enter: Q (to quit the IT command)

You can see that the display is very similar to the earlier display we saw before setting any hardware breakpoints. The one difference is in the CYCLE column. In this case any trace sample that qualified as a breakpoint is marked with the EVNT tag. In this example the program execution was halted when the trace memory was full. If we set the trace count to 0=INFINITE, we will see that the last event (the one that was recorded last and actually caused the break) will be marked with an asterisk. Let's try this one last excercise to see what happens.

Enter: TR<CR>

```
Display: TR
           TRACE COUNT (1 - 7FF, 0=INFINITE)          = 020  []
```

Enter: 0<CR>

```
Display: TR
           TRACE COUNT (1 - 7FF, 0=INFINITE)          = 020  0
           ADDRESS MASK (ONES ENABLE)                 = FFFF []
```

Enter: <CR>

```
Display: TR
           TRACE COUNT (1 - 7FF, 0=INFINITE)          = 020  0
           ADDRESS MASK (ONES ENABLE)                 = FFFF
           TRACE MODE [0=TRM & TRIO, 1=TRIX]          = 0    []
```

Enter: <CR>

```
Display: TR
           TRACE COUNT (1 - 7FF, 0=INFINITE)          = 020  0
           ADDRESS MASK (ONES ENABLE)                 = FFFF
           TRACE MODE [0=TRM & TRIO, 1=TRIX]          = 0
         ?
```

At this point we have set the number of trace samples to an infinite number of samples, the address mask is enabled which indicates that only those addresses that match our compare values will qualify as trace samples, and trace on memory and/or I/O access has been selected for tracing.

Enter: MR;RUN<CR>

Display:

```
RUN
   RUNNING
   HBP
   INDEX CYCLE QUAL  EXTQUALS      ADDR DATA  RVRS ASSEMBLY
         *EVT  IAQ   11111111      0018 38C2  MPY  R2,R3
?
```

Comment: The program stopped executing because it satisfied the hardware breakpoint criteria. The HBP prompt indicates this and shows the last event which caused the event counter to go to zero.

Enter: IT<CR>

Display: IT
FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 0000 []

Enter: <CR>

Display:

```
IT
  FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000

    INDEX CYCLE QUAL  EXTQUALS      ADDR DATA  RVRS ASSEMBLY

    0000        IAQ   11111111      000A 04C4  CLR  R4
    0001        IAQ   11111111      000C 0201  LI   R1,>000A
    0002        MW    11111111      0058 0000
    0003        MR    11111111      000E 000A
    0004        IAQ   11111111      0010 0202  LI   R2,>1234
    0005        MW    11111111      0052 000A
    0006        MR    11111111      0012 1234
    0007        IAQ   11111111      0014 0203  LI   R3,>0010
    0008        MW    11111111      0054 1234
    0009        MR    11111111      0016 0010
    000A  EVNT  IAQ   11111111      0018 38C2  MPY  R2,R3
    000B        MW    11111111      0056 0010
    000C        MR    11111111      0054 1234
    000D        MR    11111111      0056 0010
    000E        MW    11111111      0056 0001
    000F  EVNT  IAQ   11111111      001A C803  MOV  R3,@>0100
    0010        MW    11111111      0058 2340
    0011        MR    11111111      0056 0001
    0012  EVNT  MR    11111111      001C 0100

[]
```

Enter: <CR>

Display:

```
INDEX CYCLE QUAL  EXTQUALS     ADDR DATA  RVRS ASSEMBLY

0013  EVNT  IAQ   11111111     001E C804  MOV  R4,@>0102
0014        MW    11111111     0100 0001
0015        MR    11111111     0058 2340
0016        MR    11111111     0020 0102
0017        IAQ   11111111     0022 C0C1  MOV  R1,R3
0018        MW    11111111     0102 2340
0019        MR    11111111     0052 000A
001A        IAQ   11111111     0024 0601  DEC  R1
001B        MW    11111111     0056 000A
001C        MR    11111111     0052 000A
001D        IAQ   11111111     0026 16F8  JNE  >0018
001E        MW    11111111     0052 0009
001F  *EVT  IAQ   11111111     0018 38C2  MPY  R2,R3
0020        MR    11111111     0054 1234
0021        MR    11111111     0056 000A
0022        MW    11111111     0056 0000
0023  EVNT  IAQ   11111111     001A C803  MOV  R3,@>0100
0024        MW    11111111     0058 B608
0025        MR    11111111     0056 0000
```

[]

Enter: <CR>

Display:

```
INDEX CYCLE QUAL  EXTQUALS     ADDR DATA  RVRS ASSEMBLY

0026  EVNT  MR    11111111     001C 0100
0027  EVNT  IAQ   11111111     001E C804  MOV  R4,@>0102
0028        MW    11111111     0100 0000
0029        MR    11111111     0058 B608
002A        MR    11111111     0020 0102
002B        IAQ   11111111     0022 C0C1  MOV  R1,R3
002C        MW    11111111     0102 B608
002D        MR    11111111     0052 0009
002E        IAQ   11111111     0024 0601  DEC  R1
002F        MW    11111111     0056 0009
0030        MR    11111111     0052 0009
0031        IAQ   11111111     0026 16F8  JNE  >0018
0032        MW    11111111     0052 0008
0033  EVNT  IAQ   11111111     0018 38C2  MPY  R2,R3
0034        MR    11111111     0054 1234
0035        MR    11111111     0056 0009
0036        MW    11111111     0056 0000
0037  EVNT  IAQ   11111111     001A C803  MOV  R3,@>0100
0038        MW    11111111     0058 A3D4
```

[]

Enter: <CR>

Display:

INDEX	CYCLE	QUAL	EXTQUALS	ADDR	DATA	RVRS	ASSEMBLY
002E		IAQ	11111111	0024	0601	DEC	R1
002F		MW	11111111	0056	0009		
0030		MR	11111111	0052	0009		
0031		IAQ	11111111	0026	16F8	JNE	>0018
0032		MW	11111111	0052	0008		
0033	EVNT	IAQ	11111111	0018	38C2	MPY	R2,R3
0034		MR	11111111	0054	1234		
0035		MR	11111111	0056	0009		
0036		MW	11111111	0056	0000		
0037	EVNT	IAQ	11111111	001A	C803	MOV	R3,@>0100
0038		MW	11111111	0058	A3D4		
0039		MR	11111111	0056	0000		
003A	EVNT	MR	11111111	001C	0100		
003B	EVNT	IAQ	11111111	001E	C804	MOV	R4,@>0102
003C		MW	11111111	0100	0000		
003D		MR	11111111	0058	A3D4		
003E		MR	11111111	0020	0102		
003F		IAQ	11111111	0022	C0C1	MOV	R1,R3
0040		MW	11111111	0102	A3D4		

[]

As you can see, index number 001F is the line of code that caused the event counter to go to zero and is denoted by the *EVT tag in the CYCLE column. This is the same line displayed after the HBP prompt when the program stopped executing. There are, however, several more trace samples following index 001F. This is because the DELAY COUNT parameter of the BP command was set to 20 which causes >20 samples to be taken after the event counter reaches zero.

We can look at trace 001F again by jumping backward >13 traces:

Enter: -13<CR>

Display:

INDEX	CYCLE	QUAL	EXTQUALS	ADDR	DATA	RVRS	ASSEMBLY
001B		MW	11111111	0056	000A		
001C		MR	11111111	0052	000A		
001D		IAQ	11111111	0026	16F8	JNE	>0018
001E		MW	11111111	0052	0009		
001F	*EVT	IAQ	11111111	0018	38C2	MPY	R2,R3
0020		MR	11111111	0054	1234		
0021		MR	11111111	0056	000A		
0022		MW	11111111	0056	0000		
0023	EVNT	IAQ	11111111	001A	C803	MOV	R3,@>0100
0024		MW	11111111	0058	B608		
0025		MR	11111111	0056	0000		
0026	EVNT	MR	11111111	001C	0100		
0027	EVNT	IAQ	11111111	001E	C804	MOV	R4,@>0102
0028		MW	11111111	0100	0000		
0029		MR	11111111	0058	B608		
002A		MR	11111111	0020	0102		
002B		IAQ	11111111	0022	C0C1	MOV	R1,R3
002C		MW	11111111	0102	B608		
002D		MR	11111111	0052	0009		

[]

Enter: Q

1.3.5 Walk-Through - Phase 4

This phase of the walk-through introduces the user to the TMS9995 Emulator Assembler and Reverse Assembler. It includes instructions for entering assembly language programs and for reverse assembling programs contained in memory.

When you enter a complete assembly language instruction it consists of an optional label, mnemonic, and possible operand(s). You also have the option of entering a comment following the operand field. You enter a label starting in column 1. If the instruction you are using does not use a label you must enter a space before the mnemonic text which must begin in column 2. Any entry in column 1 is assumed to be a label.

Labels can be one or two characters in length. If more than two characters are entered, all but the last two are ignored. The label must start with an alphabetical character and may be followed by an alphanumeric character.

The mnemonic field is separated from the label by at least one space.

The operand field is separated from the mnemonic by at least one space. A comment may be added by following the operand field with at least one space.

The TMS9995 Emulator Assembler accepts all of the TMS9995 Assembler directives and instructions which are fully covered in the TMS9995/99000 Assembly Language Programmer's Guide, Part Number 1603753-9701. Details concerning the TMS9995 line-by-line assembler can be found in Section 4 of this manual.

1.3.5.1 Entering an Assembly Language Program

Display: ?

Enter: XA<CR>

Comment: Cursor movement is not available in the XA command. Do not attempt to move the cursor while in the XA command.

Comment: For the purpose of clarity, the columns have been expanded in the following displays.

Display:
```
XA
  START ADDRESS = 0000 []
    CONTROL-E EXITS
```

Enter: A<CR>

Display:
```
XA
  START ADDRESS = 0000
    CONTROL-E EXITS

    ADDR      DATA      LB      INST
    000A               []
```

Enter: IN CLR R4<CR>

Display:
```
XA
  START ADDRESS = 0000
    CONTROL-E EXITS

    ADDR      DATA      LB      INST
    000A      04C4      IN      CLR R4
    000C               []
```

Enter: <SP>LI R1,10<CR>

Comment: The entry of a space at the begining of the statment above causes the cursor to move to the begining of the instruction column (INST).

```
Display: XA
           START ADDRESS = 0000
             CONTROL-E EXITS

             ADDR       DATA      LB       INST
             000A       04C4      IN       CLR R4
             000C       0201               LI R1,10
             000E       000A
             0010                []

Enter:    <SP>LI R2,>1234<CR>

Display: XA
           START ADDRESS = 0000
             CONTROL-E EXITS

             ADDR       DATA      LB       INST
             000A       04C4      IN       CLR R4
             000C       0201               LI R1,10
             000E       000A
             0010       0202               LI R2,>1234
             0012       1234
             0014                []

Enter:    <SP>LI R3,>10<CR>

Display: XA
           START ADDRESS = 0000
             CONTROL-E EXITS

             ADDR       DATA      LB       INST
             000A       04C4      IN       CLR R4
             000C       0201               LI R1,10
             000E       000A
             0010       0202               LI R2,>1234
             0012       1234
             0014       0203               LI R3,>10
             0016       0010
             0018                []

Enter:    LP MPY R2,R3<CR>
```

```
Display: XA
           START ADDRESS = 0000
             CONTROL-E EXITS

             ADDR      DATA      LB      INST
             000A      04C4      IN      CLR R4
             000C      0201              LI R1,10
             000E      000A
             0010      0202              LI R2,>1234
             0012      1234
             0014      0203              LI R3,>10
             0016      0010
             0018      38C2      LP      MPY R2,R3
             001A                []
```

Enter: <SP> MOV R3,@>100<CR>

```
Display: XA
           START ADDRESS = 0000
             CONTROL-E EXITS

             ADDR      DATA      LB      INST
             000A      04C4      IN      CLR R4
             000C      0201              LI R1,10
             000E      000A
             0010      0202              LI R2,>1234
             0012      1234
             0014      0203              LI R3,>10
             0016      0010
             0018      38C2      LP      MPY R2,R3
             001A      C803              MOV R3,@>100
             001C      0100
             001E                []
```

Enter: <SP>MOV R4,@>102<CR>

```
Display: XA
           START ADDRESS = 0000
             CONTROL-E EXITS

             ADDR      DATA      LB      INST
             000A      04C4      IN      CLR R4
             000C      0201              LI R1,10
             000E      000A
             0010      0202              LI R2,>1234
             0012      1234
             0014      0203              LI R3,>10
             0016      0010
             0018      38C2      LP      MPY R2,R3
             001A      C803              MOV R3,@>100
             001C      0100
             001E      C804              MOV R4,@>102
             0020      0102
             0022                []
```

Enter: <SP>MOV R1,R3<CR>

Display: XA

```
          START ADDRESS = 0000
            CONTROL-E EXITS

            ADDR      DATA      LB      INST
            000A      04C4      IN      CLR R4
            000C      0201              LI R1,10
            000E      000A
            0010      0202              LI R2,>1234
            0012      1234
            0014      0203              LI R3,>10
            0016      0010
            0018      38C2      LP      MPY R2,R3
            001A      C803              MOV R3,@>100
            001C      0100
            001E      C804              MOV R4,@>102
            0020      0102
            0022      C0C1              MOV R1,R3
            0024              []
```

Enter: <SP>DEC R1<CR>

Display: XA

```
          START ADDRESS = 0000
            CONTROL-E EXITS

            ADDR      DATA      LB      INST
            000A      04C4      IN      CLR R4
            000C      0201              LI R1,10
            000E      000A
            0010      0202              LI R2,>1234
            0012      1234
            0014      0203              LI R3,>10
            0016      0010
            0018      38C2      LP      MPY R2,R3
            001A      C803              MOV R3,@>100
            001C      0100
            001E      C804              MOV R4,@>102
            0020      0102
            0022      C0C1              MOV R1,R3
            0024      0601              DEC R1
            0026              []
```

Enter: <SP> JNE LP<CR>

```
Display: XA
          START ADDRESS = 0000
            CONTROL-E EXITS

            ADDR      DATA      LB      INST
            000A      04C4      IN      CLR R4
            000C      0201              LI R1,10
            000E      000A
            0010      0202              LI R2,>1234
            0012      1234
            0014      0203              LI R3,>10
            0016      0010
            0018      38C2      LP      MPY R2,R3
            001A      C803              MOV R3,@>100
            001C      0100
            001E      C804              MOV R4,@>102
            0020      0102
            0022      COCi              MOV R1,R3
            0024      0601              DEC R1
            0026      16F8              JNE LP
            0028               []
```

```
Enter:   <SP>JMP IN<CR>
```

```
Display: XA
          START ADDRESS = 0000
            CONTROL-E EXITS

            ADDR      DATA      LB      INST
            000A      04C4      IN      CLR R4
            000C      0201              LI R1,10
            000E      000A
            0010      0202              LI R2,>1234
            0012      1234
            0014      0203              LI R3,>10
            0016      0010
            0018      38C2      LP      MPY R2,R3
            001A      C803              MOV R3,@>100
            001C      0100
            001E      C804              MOV R4,@>102
            0020      0102
            0022      COC1              MOV R1,R3
            0024      0601              DEC R1
            0026      16F8              JNE LP
            0028      10F0              JMP IN
            002A               []
```

```
Enter:   <SP>END<CR>
```

```
Display: XA
           START ADDRESS = 0000
             CONTROL-E EXITS

             ADDR      DATA      LB      INST
             000A      04C4      IN      CLR R4
             000C      0201              LI R1,10
             000E      000A
             0010      0202              LI R2,>1234
             0012      1234
             0014      0203              LI R3,>10
             0016      0010
             0018      38C2      LP      MPY R2,R3
             001A      C803              MOV R3,@>100
             001C      0100
             001E      C804              MOV R4,@>102
             0020      0102
             0022      C0C1              MOV R1,R3
             0024      0601              DEC R1
             0026      16F8              JNE LP
             0028      10F0              JMP IN
             002A                        END 0000
         ?
```

Notice that there are two ways to exit the assembler (that is, the XA command): one way is to enter the <SP>END instruction into the last statement of the program, the other is to enter <CTRL>E. Since control is in the hands of the assembler, all the rules governing the use of the monitor program are no longer operative. After entering the END instruction (or <CTRL>E), control of the emulator returns to the Monitor.

We can see the program in reverse assembled code by using the XRA (Execute Reverse Assembler) command. If you enter this command now, you will see the following reverse assembled output.

```
Enter:    XRA<CR>

Display: XRA
           START ADDRESS              = 000A [ ]

Enter:    <CR>

Display: XRA
           START ADDRESS              = 000A
           NUMBER OF INSTRUCTIONS     = 000B [ ]

Enter:    <CR>
```

```
Display: XRA
          START ADDRESS              = 000A
          NUMBER OF INSTRUCTIONS     = 000B
            000A 04C4  CLR  R4
            000C 0201  LI   R1,>000A
            0010 0202  LI   R2,>1234
            0014 0203  LI   R3,>0010
            0018 38C2  MPY  R2,R3
            001A C803  MOV  R3,@>0100
            001E C804  MOV  R4,@>0102
            0022 C0C1  MOV  R1,R3
            0024 0601  DEC  R1
            0026 16F8  JNE  >0018
            0028 10F0  JMP  >000A
         ?
```

SECTION 2

THE XDS MONITOR PROGRAM

2.1 INTRODUCTION

The TMS9995 Emulator is controlled by the XDS monitor program. This section introduces the XDS monitor program and provides general information on command entry, command parameters, special character directives, command terminators, editor features, the use of commands in procedures, and register variables as a means of register access.

This section also contains a discussion of the process used to enter values for command parameters. The user will find that parameter value entry has been simplified by using interactive prompting messages or parentheses.

The information in this section is presented in a general, hypothetical manner to acquaint the user with the monitor's convenient features and commands. A few specific examples are provided to avoid confusion about some of the features presented. For more detailed information and examples of these features, refer to the appropriate sections of this manual.

2.2 NOTATION

The syntax conventions described below are used throughout this manual to simplify the descriptions of the XDS monitor program for the TMS9995 Emulator.

* Angle brackets < > indicate that something must be typed from the keyboard. If the text in the brackets is in UPPER CASE, press the key named in the text. For example <CR> means to press the carriage return key. If the text in the brackets is in lower case then the entry must be typed out. For example, <value> means to type a value such as a hexadecimal address.

* Square brackets [<value>] indicate an optional entry that is to be typed from the keyboard. The information in these brackets does not have to be entered.

* Ellipses "..." indicate that a procedure can be repeated as many times as needed.

* The cursor is represented by the symbol []. When positioned over a character, the cursor is represented by underlining the character (0).

* Control key sequences are represented by the abbreviation CTRL-alphabetic character. To enter a control-key sequence, the CTRL (or CONTROL) key is depressed while the alpha character key is typed. For example, CTRL-A represents the control-A key sequence.

2.3 XDS START-UP PROCEDURE

Use the following steps to apply power to the XDS and to start execution of the monitor program:

1) Place the POWER switch in the ON position.

2) Press the RESET button and wait at least five seconds.

3) Press <CR> on the terminal keyboard twice. This allows the monitor program to set the baud rate on the XDS unit to conform to that of the user's terminal automatically through an AUTOBAUD operation.

4) The terminal displays a banner, a list of emulator commands, and a ? prompt to indicate that the monitor is awaiting input. The display appears as follows.

```
                     TEXAS INSTRUMENTS

                 TMS-9995  XDS  VERSION 1.3.0

  COMMANDS:
   INIT    IM     DR     RUN    BP     TR     HOST   IMP
   IPORT   DM     MR     CRUN   BPM    TRM    IHC    IMD
   IPRM    MM     DIO    SS     BPIO   TRIO   UL     ID
   ICC            MIO    SRR           TRIX   DL     BGND
   RCC                                 SOR
   RESTART

   MAP     FILL   XA     DPS    SSB    IT     LOG    GRUN
           FIND   XRA    DHS    DSB    DT     SNAP   TRUN
           DW            DTS    CSB           HELP   GHALT
                                CASB          DV     THALT

  VARIABLES:
   PC      R      LGT    C      INTM
   ST             AGT    OV
   WP             EQ     OP

?
```

2.4 XDS MONITOR COMMANDS

Monitor commands fall into two broad categories: those with parameters and those without parameters. These commands are used to control the functions of the XDS monitor program. All of the commands available for the TMS9995 are listed in the display that appears after the power-up sequence is completed.

To illustrate command entry and execution, several hypothetical commands will be used. These commands and their parameters are shown in Table 2-1. Parameters contain values used to control execution of a command. For example, when displaying a segment of memory, the user needs to specify the first and last address of the segment. In this case, the first parameter is the beginning address and the second parameter is the ending address.

TABLE 2-1. HYPOTHETICAL COMMANDS AND THEIR PARAMETERS.

COMMAND PARAMETER	LAST ENTRY	PRESENT ENTRY
CMD1		
PARAMETER 1A	= XXXX	XXXX
PARAMETER 1B	= XXXX	XXXX
CMD2		
PARAMETER 2A	= XXXX	XXXX
PARAMETER 2B	= XXXX	XXXX
CMD3		
PARAMETER 3A	= XXXX	XXXX
CMD4		
PARAMETER 4A	= XXXX	XXXX
PARAMETER 4B	= XXXX	XXXX
PARAMETER 4C	= XXXX	XXXX
PARAMETER 4D	= XXXX	XXXX
CMD5		
No parameters	None	None

2.5 REGISTER VARIABLES

The XDS monitor also provides a set of register variables used to access registers. The term variable is used because they serve much the same function as variables in higher-level programming languages. Each register is represented by a specific variable name (code) which can be assigned values (write mode) or which can return values (read mode). An example of a register variable is PC, which is the symbol for the Program Counter. These variables simplify the process of displaying or changing the contents of registers.

For this discussion, hypothetical variables will be used. Two general types of register variable appear: fixed register variables (FR) and indexed register variables (IRnn). A fixed register variable is a code assigned to one of the processor's registers. An indexed register variable is a code assigned to a memory location designated as a register. Indexed registers use decimal index numbers (nn) to identify the individual registers.

These variables can be entered into the command line in either their read or write mode.

2.5.1 Read Mode

The read mode displays the contents of a register and takes the form of the register variable followed by a terminator or separator. The monitor returns the following in the display:

Enter: FR<CR> or IRnn<CR>

Display: FR=000 or IRnn=0000

2.5.2 Write Mode

The write mode is used to change the contents of a register.

Enter: RV=<new value><CR> or IVnn=<new value><CR>

Comment: This form changes the contents of the register to the new value.

Refer to REGISTER VARIABLES in Section 3 of this manual for details regarding the register variables for the TMS9995.

2.6 BUFFER ENTRY

The monitor is controlled by a set of commands which are entered into an input buffer for execution. The process involves three major steps as follows:

1. Entering a command or register variable
2. Defining the command's parameters (for those commands with parameters)
3. Executing the command

Commands and register variables are entered into a command line. When the user enters a <CR>, if a command has parameters, it proceeds into the parameter definition phase. At this point parameters are assigned values used by the command during execution. The following modes are used for parameter definition.

* Interactive mode - The parameters and their current values are displayed for user input.
* Default mode - Parameters retain the values that were last entered.
* Parenthetical mode - Parameter values are entered as a list of values separated by commas or spaces, enclosed in parentheses.

In addition to the parameter definition modes above, the user can preview the parameter values without executing the command using the <u>preview mode</u>.

After the parameter values are defined and the command is executed, the monitor takes some action that can change or display internally stored information, configure some component of the XDS system, or run a program.

2.6.1 The Command Line

The XDS monitor is ready to accept data into the input buffer when it displays the ? prompt. Information typed from the keyboard is stored in one of two input buffers. This allows a series of commands (called a procedure) to be entered and stored in one of the buffers for sequential execution. Each input buffer can contain a procedure which is then available for execution. Typing a <+> or a <-> produces a display of the contents of the alternate buffer. The buffers are explained later in this section.

Commands or register variables are entered into a command line with the following general format.

<CMD><term>[<CMD><term>] ... [<CMD><term>][*][<nnnn>]<CR>

Where:

<CMD> may be any one of the following.

* A command name (i.e., CMD1)

* A command name followed by a list of parameter values enclosed in parentheses (i.e., CMD1(p1,p2))

* A register variable (in either read or write mode)

<term> is a command terminator which is a special character providing additional control over command execution. When more than one CMD is entered in the command line, the terminators also act as separators. The rules governing the use of terminators are presented below.

<nnnn> is a decimal number (0 - 9999)

2.6.1.1 Single Commands in the Command Line

When a single command is entered into the command line:

* The <CR> acts as a valid command terminator and invokes the interactive parameter definition phase.

* Other valid command terminators are:

  - . (Period) invokes the default parameter definition mode and displays the parameter values before executing the command.

  - ; (Semicolon) invokes the default parameter definition mode and suppresses the display of the parameters and their values before executing the command.

! (Exclamation point) invokes the preview mode which displays the parameter values without executing the command.

, (Comma) is accepted but not required, and it invokes the interactive parameter definition mode.

SPACE Is accepted but not required, and it invokes the interactive parameter definition mode.

* The asterisk (*) is a command-line terminator and results in repeated execution of the command line. If the (*) is followed by a decimal number (1 to 9999), the command line is executed the specified number of times.

* A command entered in the parenthetical parameter definition mode will accept only the comma, space, or <CR> as valid terminators. Other terminators result in error conditions.

2.6.1.2 Multiple Commands in the Command Line

When more than one command is entered into the command line:

* Additional commands must be separated by one of the command terminators listed above (except <CR>).

* Commands in the parenthetical parameter definition mode use the right parenthesis) as the separator. The comma or space are optional separators that can follow the right parenthesis.

* The asterisk (*) is a command line terminator and results in repeated execution of the command line. If the (*) is followed by a decimal number (1 to 9999), the command line is executed the specified number of times.

2.6.1.3 Register Variables in the Command Line

The following rules apply when entering register variables into the command line.

* Register variables can be entered in either the read or write mode.

* Register variables use only the <SP> or the comma as valid terminators.

2.6.2 Defining Parameters

Most of the monitor commands have parameters that allow user definition for proper command execution. When the start-up procedure is completed, these parameters assume power-up values. In some instances, the power-up value will be sufficient for the user's needs and will not require resetting.

Other cases may require that the parameter values be redefined by the user to conform to his needs. In this case the user has two modes available to reset the parameter values. The interactive mode allows the user to see the parameters and their values before making a change.

The parenthetical mode allows the user to define the parameters as he enters the command into the command line by enclosing the values within parentheses following the command name.

Once parameter values are defined (either by default on power up or by being reset by the user) the values remain in effect until changed by the user. Once the user has defined a set of parameters, he can then use the default mode to retain these values when the command is executed.

Each of these modes is summarized in Table 2-2 and discussed in the following paragraphs.

TABLE 2-2. COMMAND TERMINATORS AND PARAMETER DEFINITION MODES

Terminator	Parameter Definition	Parameter Display	Enter Parameter Value	Effect on Command Execution
,	Interactive	Yes	Yes	Uses new par. values
<SP>	Interactive	Yes	Yes	Uses new par. values
.	Default	Yes	No	Uses previous values
;	Default	No	No	Uses previous values
!	Preview	Yes	No	Not executed

Note: <CR> acts as a terminator for single commands and goes into the interactive mode for defining parameters.

2.6.2.1 The Interactive Mode

The interactive mode provides a simple method for entering parameter values prior to command execution. This entry mode allows the user to see a display of the parameter and its current value before deciding whether a change is necessary.

2.6.2.1.1 Entering Parameter Values

In the interactive entry mode, the XDS monitor displays each of the command's parameters for user input before the command is executed. If the user wants to enter a new value into a parameter, he responds to the prompt by entering the new value followed by a carriage return. If the user wants to keep the parameter's current value, he enters a carriage return.

As shown in Table 2-2, a command entered with either a <SPACE>, a <,>, or a <CR> terminator forces the XDS monitor into the interactive mode. Generally, when a single command is entered into the buffer, the terminating SPACE or comma is not used. They are most often used to separate commands when more than one command is entered.

The general pattern for entering parameter values in the interactive mode is as follows:

PARAMETER PROMPT = CURRENT VALUE [<new value>]<CR> ERROR MESSAGE

An error message appears only if the new value is not valid. The cursor moves back to the first character position in the <new value> entry field so that the user can retype the entry. When entering a correction, any extra characters that appear to the right of the new characters must be deleted to avoid a subsequent invalid entry.

2.6.2.1.2 Exiting the Interactive Mode

There are two ways to exit the interactive parameter definition mode and continue with command execution. When the value of the final parameter for a command is entered with the <CR>, the command is executed. The user can also execute the command with its existing parameter values by entering a Q<CR> in response to a parameter prompt. In this case the current parameter values include those new values entered before the Q<CR> as well as the old values of the remaining parameters.

Typing an <ESC> during the interactive parameter definition mode aborts both parameter definition and command execution. None of the parameter values are changed. However, if <ESC> is typed after the parameters are defined and while the command is executing, only command execution is aborted (the display of memory, for example) and the parameters retain the values that were entered by the user.

The following examples illustrate the process of parameter definition.

Example 1: Defining parameters using the interactive mode

Display: ?

Enter: CMD1<CR>

```
Display:                 Enter:

CMD1
   PARAMETER 1A = 0000    11<CR>
   PARAMETER 1B = 0000    13<CR>
      [OUTPUT FROM CMD1 IS DISPLAYED]
     ?
```

Example 2: Aborting parameter definition and command execution

Enter: CMD1<CR>

```
Display:                 Enter:

CMD1
   PARAMETER 1A = 0011    AA<CR>
   PARAMETER 1B = 0013    <ESC>
?
```

Comment: At this point, the value of PARAMETER 1A has not been changed to AA and remains 0011. This is illustrated by entering the command in preview mode using the ! terminator to display the parameters.

Enter: CMD1!<CR>

Display:

```
CMD1
   PARAMETER 1A = 0011
   PARAMETER 1B = 0013
?
```

Example 3: Aborting command execution after defining parameters

Enter: CMD1<CR>

```
Display:                  Enter:

CMD1
   PARAMETER 1A = 0011    AA<CR>
   PARAMETER 1B = 0013    BB<CR>
      [OUTPUT FROM CMD1 IS DISPLAYED]
```

Enter: <ESC> (Entered while the output is being displayed.)

Display: ?

Comment: The parameter values are shown using the preview mode.

Enter: CMD1!<CR>

Display:

```
CMD1
   PARAMETER 1A = 00AA
   PARAMETER 1B = 00BB
?
```

Comment: Note that the parameter values are changed to the values entered above.

Example 4: Quitting parameter definition and proceeding to execute the command using Q<CR>

Enter: CMD4<CR>

```
Display:                  Enter:

CMD4
   PARAMETER 4A = 0000    CC<CR>
   PARAMETER 4B = 0000    FFFF<CR>
   PARAMETER 4C = 0000    Q<CR>
      [OUTPUT FROM CMD1 IS DISPLAYED]
?
```

Comment: The Q <CR> terminates the interactive entry mode and completes command execution. The resulting parameter values can be seen by using the preview mode.

Enter: CMD4!<CR>

Display:

```
CMD4
   PARAMETER 4A = 00CC
   PARAMETER 4B = FFFF
   PARAMETER 4C = 0000
   PARAMETER 4D = 0000
?
```

Comment: After entering the Q<CR>, PARAMETER 4A is set to a new value of 00CC, PARAMETER 4B is set to FFFF, and the remaining two parameters retain their power-up values of 0000 when the command is executed.

2.6.2.2 Default Mode

The default mode is a convenient way to execute a command without direct user input. This entry mode can be used when existing parameter values do not need to be redefined.

Terminating a command with a period or semicolon executes the command using its current parameter values. The period results in a display of the parameters and their current values, while a semicolon suppresses this display.

Example 5: Use of the period (.) terminator

Display: ?

Enter: CMD1.<CR>

Display:

```
CMD1
  PARAMETER 1A = 00CC
  PARAMETER 1B = 00BB
    [OUTPUT FROM CMD1 IS DISPLAYED]
?
```

Comment: The . tells the monitor to display the current values of the parameters for CMD1 and then to execute the command using these values.

Example 6: Use of the semicolon terminator

Enter: CMD1;<CR>

Display:

```
    [OUTPUT FROM CMD1 IS DISPLAYED]
?
```

Comment: The ; specifies that the command is to execute using the current values, and the parameters are not to be displayed.

2.6.2.3 Parenthetical Entry Mode

Parameter values can be entered using the parenthetical mode of entry. When parameter values are entered using parentheses, there is no display of the parameters.

2.6.2.3.1 General Format

The general format for a command entered using parenthetical entry is as follows:

CMD(p1,p2,p3,...,pn)

The command name is followed by a list of values for each parameter. The values are separated by commas (or spaces) and enclosed in parentheses. If the user wants the parameter to default to its current value, a period <.> is used instead of a new value. Also, if the user wants to retain the current values of parameters remaining after changing one parameter, the Q can be used in place of the value of the first parameter that is not changed (for example, CMD4(22,AAAA,Q)).

NOTE

The <SPACE> can be used instead of the comma to separate parameter values inside parentheses. To avoid confusion, the examples in this section will use only commas as separators.

2.6.2.3.2 Terminating Commands Entered Using Parentheses

The command-line terminators (*) and (*nnnn) can be used with the parenthetical entry mode when a single command is entered in the command line. The comma or space can be used as separators following the right parenthesis, but are considered optional. The default-mode terminators (period and semicolon) and the preview mode (!) are not acceptable when using parentheses.

The following examples illustrate various conditions for parenthetical entry. The preview mode is used to reveal the parameter values that existed when the command executed.

Example 7: Entering a command using parentheses ( )

Display: ?

Enter: CMD1(.,1)<CR>

Display: [OUTPUT FROM CMD1 IS DISPLAYED]
?

Comment: In this case, the period in the P1 position indicates that the current value of the first parameter is to be retained. The new value of 1 is entered into the second parameter. The <CR> causes the command to be executed using the parameter values defined within the parentheses.

Enter: CMD1!<CR>

```
CMD1
  PARAMETER 1A = 00CC
  PARAMETER 1B = 0001
?
```

When a single command is entered using the parenthetical mode, it is executed immediately after the carriage return. If the command is part of a command-line procedure, it will be executed in its proper sequence.

2.6.3 Preview Mode

Commands entered with an exclamation point terminator (!) have their parameters displayed along with their current values. The command is not executed and there is no output generated. These rules are illustrated in the examples that follow.

Example 8: Use of <!> to display the specified command parameters

Enter: CMD1!<CR>

Display:

```
CMD1
  PARAMETER 1A = 00CC
  PARAMETER 1B = 00BB
?
```

2.7 REPEATING EXECUTION OF THE COMMAND LINE

There are two ways to repeat execution of the command line.

* Terminating the command line with the repeat terminator:
  - <*> The command line executes indefinitely until <ESC> is typed
  - <*><nnnn> The command line executes nnnn times

* Entering one of the following after the ? prompt:

  - <SPACE> The command line is executed once

  - <*> The command line is executed indefinitely until <ESC> is typed

Execution of the command line can be aborted at any time by typing <ESC>.

Example 9: Repeating command-line execution with the (*) terminator

Enter: CMD1.*2<CR>

Display:

```
CMD1
  PARAMETER 1A = 00CC
  PARAMETER 1B = 00BB
     [OUTPUT FROM CMD1 IS DISPLAYED]

CMD1
  PARAMETER 1A = 00CC
  PARAMETER 1B = 00BB
     [OUTPUT FROM CMD1 IS DISPLAYED]
?
```

Comment: The user specified that CMD1 was to be executed twice (*2) and that the parameters were to be displayed (.) each time.

Terminating the command line with only the <*> executes the command line indefinitely until the user types <ESC>.

Example 10: Using <SPACE> to repeat execution of the command line once

Enter: <SPACE>

Display:

```
CMD1
  PARAMETER 1A = 00CC
  PARAMETER 1B = 00BB
     [OUTPUT FROM CMD1 IS DISPLAYED]
?
```

2.8 EDITING ENTRIES

Command-line entry and parameter values are the only data entries required of the user. The monitor provides a means for error correction for each of these types of data entry. Usually, to make a correction requires that the cursor be positioned at the error location. Cursor movement requirements are slightly different for command-line editing and parameter value correction. These two types of cursor movement are explained below.

2.8.1 Editing the Command Line

Editing the command line requires the features of a line editor. This means that only left and right cursor movement is needed.

Since cursor movement functions can be implemented differently on various terminals, the XDS monitor recognizes characters commonly used on most terminals to position the cursor. These characters and their corresponding cursor control functions are listed in Table 2-3. The monitor can also insert or delete characters by using the control characters, which are also listed in Table 2-3.

TABLE 2-3. CHARACTERS USED TO POSITION THE CURSOR AFTER POWER-UP

Key	Cursor Position
< CTRL-H	Moves cursor one position to the left. Characters are not erased.
>	Moves cursor one position to the right. Characters are not erased.
-	Moves the cursor up.
+	Moves the cursor down.
CTRL-D	Deletes the character under the cursor.
CTRL-I	Inserts one space and moves text to the right of the cursor.

To take advantage of cursor control functions on terminals that have cursor movement features, the ICC (Initialize Cursor Control) command should be executed. This command allows the use of cursor control keys for cursor positioning, such as UP ARROW, DOWN ARROW, LEFT ARROW, and RIGHT ARROW. These keys replace the power-up cursor control characters listed in Table 2-3. The use of the ICC command is explained in paragraph 2.9, Initializing Cursor Control.

2.8.1.1 Editing During Entry Mode

The entry mode is invoked when the user begins typing a new command line after the ? prompt appears. At that time, the buffer is cleared in preparation for the new entry.

If the user notices an error while entering the command line, he can use the < key to move the cursor to the left. When the cursor is in the proper position, the user takes the necessary action to make the correction.

2.8.1.2 Editing in the Edit Mode

The following conditions place the command line into edit mode.

* Typing <?> when the ? prompt is displayed.

* A syntax error is encountered in the command line.

* Typing <+> or <-> to display the alternate buffer contents.

If the user types a <?> when the monitor displays the ? prompt, the command line is displayed with the cursor over the first character. The user can position the cursor at the location of the error and make the correction. Any unnecessary characters appearing in the command line must be deleted before it is executed. If the buffer is empty, the monitor returns the ? prompt.

If the user types a <+> or a <-> while in the command-line entry mode, the contents of the alternate buffer are displayed for editing. The line can be edited the same as when the <?> is typed.

If a syntax error is encountered in a command line, the command line is displayed with the cursor positioned over the offending character. The user can then enter the correction.

Once a correction has been made, the command line can be executed by entering a <CR> from any position in the line. If the user types a <+> or <->, the command line is stored in the current buffer and the alternate buffer is displayed in the edit mode. The command line is not executed until the <CR> is entered.

2.8.2 Editing Erroneous Parameter Entries

There are two types of errors that can occur when parameter values are entered. The first is an invalid value entry. In this case, the monitor immediately informs the user of the condition by displaying an error code.

The second type of error is more subtle and occurs when the user enters a valid parameter value which does not conform to his specific needs. If this type of error occurs during the interactive parameter definition mode, the user can take corrective action before executing the command. If the error is undetected or is introduced in the parenthetical-parameter definition mode, the command will be executed with the erroneous parameter value. The user must redefine the parameter to correct the error.

2.8.2.1 Correcting Invalid Parameter Values

When an invalid parameter value is detected, an error message is displayed and the cursor is placed over the first character in the new-value data entry field. The corrected value is then typed and entered with a carriage return. If there are any extra characters displayed, they must be either replaced with spaces or deleted using CTRL-D (or the DELETE key).

2.8.2.2 Correcting Previous Entries in the Interactive Mode

If the user notices an error after entering the first parameter value, he can return to the incorrect parameter value. For example, if he has entered the first two parameter values and discovers an error in the first parameter, he can type a <-> twice. In this case, the second parameter is returned as a prompt, then the first parameter. When the first parameter is displayed, the user can type the correct value and enter it with a <CR>.

The sequence appears as follows:

```
Enter:    CMD4<CR>

Display:                      Enter:

CMD4
  PARAMETER 4A = 0000         ABAB<CR>
  PARAMETER 4B = 0000         FADE<CR>
  PARAMETER 4C = 0000         -
- PARAMETER 4B = FADE         -
- PARAMETER 4A = ABAB         BBBB<CR>
  PARAMETER 4B = FADE         <CR>
  PARAMETER 4C = 0000         <CR>
     [OUTPUT FROM CMD4 IS DISPLAYED]
```

2.9 INITIALIZING CURSOR CONTROL

The Initialize Cursor Control command (ICC) changes the cursor control functions from the power-up values to user-defined values. This allows the user to define the cursor movement functions to conform to the capabilites of his terminal.

2.9.1 Mapping Cursor Movement Keys

The ICC command is intended for terminals that have the cursor control functions of the UP-ARROW, DOWN-ARROW, LEFT-ARROW, and RIGHT-ARROW keys. This command is necessary because the ASCII codes generated by cursor control keys are not standardized and can vary for different terminals.

The following guidelines apply to the use of the ICC command.

* UP-, DOWN-, LEFT-, and RIGHT-ARROW keys are most frequently used for cursor control.

* Alternative control-key sequences can be used in place of the arrow keys. The following rules apply to these key sequences.

  - Only single ASCII control characters can be used.

  - Complex control-key sequences used by some intelligent terminals are not acceptable.

- If the user experiences any problems with ICC cursor control, he should execute the RCC (Reset Cursor Control) command to return to power-up conditions.

* When ICC is successfully implemented, the power-up characters will no longer move the cursor. Attempts to use these characters will result in the character being typed.

* If the CTRL-I (insert) or CTRL-D (delete) character is defined for cursor movement, it will no longer function as an insert or a delete key.

* For editing parameters in the interactive mode, the up and down cursor functions move the cursor up and down into the data entry fields. This is in contrast with the power-up mode in which the parameter prompts are rewritten. Table 2-4 summarizes the uses of cursor control.

TABLE 2-4. SUMMARY OF CURSOR CONTROL FUNCTIONS

	Command Entry		Parameter Entry	
Cursor Direction	With ICC	Without ICC	With ICC	Without ICC
UP	NA	NA	UP ARROW or CTRL-Char	-
DOWN	NA	NA	DOWN ARROW or CTRL-Char	+
RIGHT	>	RIGHT ARROW or CTRL-Char	RIGHT ARROW or CTRL-Char	>
LEFT	< and CTRL-H	LEFT ARROW or CTRL-Char and CTRL-H	LEFT ARROW or CTRL-Char and CTRL-H	< and CTRL-H

CTRL-Char = any control character specified by the user
CTRL-H = the universal ASCII backspace character
NA = not applicable

Example 11: Defining Cursor Control Functions

Enter: ICC<CR>

```
Display:                                   Enter:          Response:

ICC
  DEPRESS KEY FOR CURSOR UP    =           <UP-ARROW>         OK
  DEPRESS KEY FOR CURSOR DOWN  =           <DOWN-ARROW>       OK
  DEPRESS KEY FOR CURSOR LEFT  =           <LEFT-ARROW>       OK
  DEPRESS KEY FOR CURSOR RIGHT =           <RIGHT-ARROW>      OK
?
```

Comment: The cursor control functions are now mapped into the cursor control keys on the user's terminal keyboard.

Example 12: Using cursor control functions in the interactive mode

Enter: <CMD4><CR>

Display: PARAMETER 4A = 0006 []

Enter: 5<DOWN ARROW>

Comment: The DOWN ARROW acts as a <CR> in this case.

Display:
PARAMETER 4A = 0006 5
PARAMETER 4B = 0000 []

Enter: 8<DOWN ARROW>

Display:
PARAMETER 4A = 0006 5
PARAMETER 4B = 0000 8
PARAMETER 4C = 0000 []

Enter: <UP ARROW>

Display:
PARAMETER 4A = 0006 5
PARAMETER 4B = 0000 0008 <---- Cursor
PARAMETER 4C = 0000

Comment: The user has moved the cursor up to the data entry field for PARAMETER 4B. Once the cursor is positioned in a data entry field, the LEFT- and RIGHT-ARROW keys can be used to move the cursor to the desired position within the data entry field. Individual characters can be changed by positioning the cursor over the character and typing in a new character. Characters can also be inserted or deleted using the insert or delete function.

Enter: <RIGHT-ARROW><RIGHT-ARROW><RIGHT-ARROW><7><UP-ARROW>

Display:
PARAMETER 4A = 0006 0005 <---- CURSOR
PARAMETER 4B = 0000 0007
PARAMETER 4C = 0000

Comment: The user has corrected PARAMETER 4B and moved the cursor up to the data entry field for PARAMETER 4A.

Enter: <RIGHT-ARROW><RIGHT-ARROW><RIGHT-ARROW><4><DOWN ARROW>

Display:
PARAMETER 4A = 0006 0004
PARAMETER 4B = 0000 0007 <---- Cursor
PARAMETER 4C = 0000

Comment: The user has replaced "5" with "4" in PARAMETER 4A and moved the cursor into the data entry field of the next parameter.

```
Enter:      <CR>

Display:    PARAMETER 4A  = 0006 0004
            PARAMETER 4B  = 0000 0007
            PARAMETER 4C  = 0000 []      <---- Cursor

Enter:      Q<CR>
```

The user has completed the parameter definition phase and the command is executed when the Q<CR> is typed. The command is executed using the values of the parameters as they finally appear in the data entry fields. Any attempt to use the <DOWN-ARROW> to exit the last parameter is ignored.

2.9.2 Terminating Cursor Control Function

Entering the RCC (Reset Cursor Control) command restores the cursor control characters to their power-up values. This command can be entered at any time except during command execution. The cursor control functions can be activated again by using the ICC command.

2.10 SPECIAL MONITOR FEATURES

A number of special XDS monitor features are provided to make using the XDS system simpler and easier. These features are discussed in the paragraphs that follow.

2.10.1 Dual Input Buffers

The TMS9995 Emulator provides two input buffers. One is 60 characters long, the other is 20 characters long. When the user enters data into one buffer (the current buffer), the second buffer (alternate buffer) can be used to store a second command line. The contents of the alternate buffer can be displayed in the edit mode at any time by typing a <+> or <->. These keys act as recall functions for the alternate buffer and store functions for the current buffer.

NOTE

> If the command line being entered is more than 20 characters long, and the 20-character buffer is the current buffer, input will stop on the twentieth character. If this occurrs, the user should either truncate the command line at the last complete command, or enter <-> or <+> to change to the 80-character buffer and reenter the command line.

If the user is entering a command line into the current buffer and presses a <+> or <->, the information that has been entered is stored in the current buffer without any execution, and the contents of the alternate buffer are displayed in the edit mode. The previous alternate buffer now becomes the current buffer.

The command line can be stored with no execution by typing a <+> or <->. This makes it possible to store two command-line procedures in the buffers so that the user can control the order of their execution.

2.10.2 Controlling Output Displays

Many XDS monitor commands produce some output to be displayed. The following features provide control of these displays.

2.10.2.1 Pause Control

When lengthy outputs from command execution are displayed, the display will scroll toward the top of the screen. The user can temporarily stop the display by pressing any key (except <ESC>). The <ESC> key immediately aborts execution of the command, the display is discontinued, and the monitor returns to the ? prompt mode. To continue with the display after a pause, the user simply presses any key (except <ESC>).

2.10.2.2 Fixing the Display

It can be annoying if the user repeats execution of a command-line procedure with output displays that continuously scroll up the screen. Updated data can be rewritten to the screen so that the display remains in a fixed position by using the SNAP command. When the last command in the command-line procedure is executed, the final 22 lines of output become the fixed display. If output exceeds the 22 line capacity of the screen, the output preceding the last 22 lines will not re-appear on the screen.

When using the SNAP function, the user must conform to the following set of rules.

* SNAP must be the first command in a command line.

* The terminal must have cursor control functions and the ICC (Intitialize Cursor Control) command must be successfully executed prior to using SNAP.

* The display will scroll on the first execution of the command line. Subsequent executions of the command line result in a fixed display.

* The fixed display will include only the last 22 lines of the command-line output. If the command line generates more than 22 lines of output, only the last 22 lines will be displayed on subsequent updates of the output.

* SNAP may cause confusing displays because previously displayed data is not cleared from the screen when the screen is rewritten. This situation occurs under the following conditions:

  - When commands are included which use the interactive parameter definition mode.

  - When commands are executed which require varying amounts of user interaction for entering data such as the Inspect Memory (IM) command.

2.10.3 Special Display Features

When data in segments of memory are displayed, the ASCII character of the code in each byte is displayed to the right of the hexadecimal representation of the data.

Example 14: A display of memory using the DM command.

Enter: DM<CR>

```
Display:                     Enter:

DM
  START ADDRESS = 0000       <CR>
  END   ADDRESS = 0020       24<CR>
    0000=2260 6458 0000 9548 0E25 2120 2020 2020  "` dX .. aH .% !
    0010=2020 001A 2020 5E1A 2020 0022 2020 2020    ..    ^.    ."
    0020=0120 0026 06B4                           .  .& ..
?
```

Comment: The ASCII characters appear to the right of the display. Non-printable characters are represented by the period. Location 0000 contains the data >22 which is the ASCII code for the quote (").

2.11 ERRORS

Problems with errors have been minimized by the many error trapping features of the XDS system. The monitor program scans the command line for syntax errors prior to its execution. When syntax errors are detected, an error message is displayed that includes an error code that is explained in detail in Section 8 of this guide. The monitor program enters the edit mode and displays the command line with the cursor positioned over the erroneous character.

Parameter values are checked for validity when they are entered. Invalid values cause an error code to be displayed. These error codes are explained in Section 8. Parameter-value entry fields are limited to a specified number of characters, and the monitor will not allow more than the specified number of characters to be entered.

When the monitor displays the ? prompt, the user can type a control character that results in an error condition. Generally, the monitor ignores these characters and such an entry is rare.

In some instances, the user can enter a command in the command line that is not acceptable because the XDS system is not operating in the proper mode. The commands used in the multi-processing mode are examples of this condition. If the user attempts to execute some of these commands in the stand-alone mode, an error may result.

Commands entered in the parenthetical mode may be especially prone to errors. The parameter values are easily entered in the wrong order, with invalid values, with incorrect separators, with incorrect terminators, and with the incorrect number of parameters specified. Special care must be taken when using this entry mode to avoid these problems.

Summary of Error Conditions

* Command line syntax errors
    - Misspelled or invalid command names
    - Commands not properly separated
    - Invalid separator characters
    - Invalid command terminators
    - Invalid placement of the SNAP command, it must be the first command in the command line
    - Invalid placements of the "*"

* Parameter definition errors
    - Invalid value entered
    - Value exceeds the maximum allowable entry

* Register variable errors
    - Fixed Register variables
        Assigned value is not valid
        Index assigned to fixed register variable
    - Indexed register variables
        No index assigned to indexed register variable
        Index too large
        Writing to a read-only register variable

* Miscellaneous error conditions
    - Breakpoint errors
        Breakpoint/trace board not installed
    - Multi-processing errors
    - Memory mapping errors
    - Hardware errors
        Memory parity
        Wrong clock
        No clock
        Slow operation

A review of the error messages in Section 8 is strongly recommended.

SECTION 3

COMMANDS
REFERENCE

3.1 INTRODUCTION

This section presents the TMS9995 Emulator commands. The commands are listed alphabetically in a reference format for easy access to the basic information necessary for each command. The reference material is followed by one or more examples to illustrate the command.

3.2 NOTATION

The syntax conventions described below are used throughout this section to simplify the descriptions of the TMS9995 Command Language.

* The user's response to prompts, and any other entries required of the user, are highlighted in the text by shading.

* Angle brackets < > indicate that something must be typed from the keyboard. If the text in the brackets is in UPPER CASE, press the key named in the text. For example <CR> means to press the carriage return key. If the text in the brackets is in lower case then the entry must be typed out. For example, <command> means to type the command code.

* Ellipses (...) indicate that a procedure may be repeated as many times as needed.

* The cursor is represented by the symbol []. If the cursor is positioned over a character, it is symbolized by underlining the character (<u>A</u>)

* Control key sequences are represented by CTRL followed by the second key. To enter a control-key sequence, the CONTROL (or CTRL) key is depressed while the alpha character key is typed. For example, CTRL-A represents the CONTROL-A key sequence.

The descriptions of the TMS9995 Emulator commands are formatted as follows.

COMMAND: The mnemonic code is provided along with the title of the command. The command code must be entered exactly as specified.

PURPOSE: The function of the command is presented.

PARAMETERS: The parameters associated with the command are presented in tabular form as follows.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
PARAMETER 1	0 - 7FF	0
PARAMETER 2	0 - FFF	FFF
PARAMETER 3	0 = NO 1 = YES	0

PARAMETER VALUES: The appropriate values for each parameter are listed. If a range of values is used for a parameter, the upper and lower limits are shown.

POWER-UP VALUE: The value that the parameter assumes when the emulator is powered up. In most cases the power-up value is zero (0).

Several parameters interact with the parameters of other commands. These are tabulated as follows.

INTERACTION WITH OTHER COMMANDS AND PARAMETERS

PARAMETER	COMMAND AFFECTED	PARAMETER AFFECTED

The PARAMETER in column one refers to the current command parameter. The affected command is listed in the second column, and the specific parameter that is affected is shown in column three.

In some cases the parameters of a given command may be affected by other parameters. These parameters are provided in a table as follows.

COMMANDS AND PARAMETERS THAT AFFECT CMD PARAMETERS

INTERACTING COMMAND	INTERACTING PARAMETER	AFFECTED PARAMETER

EXAMPLE: One or more examples are provided to illustrate the use of the command. If the user enters the information as directed, most examples will provide "hands-on" experience with the command in an interactive mode. Note that the shaded areas of the displays indicate the entries that must be made.

NOTE

The displays illustrated in this manual approximate the actual display seen on the video terminal screen.

3.3 COMMAND: BGND - Background Mode

PURPOSE: Places a specified unit in the multi-processor chain into background operation. This allows the emulator to operate off line. The unit is addressed using #n (where n is the identification number assigned when the multi-processor mode was initialized).

NOTE: This command is used in multi-processing mode only.

NO PARAMETERS.

EXAMPLES:

Situation: Five emulators are operating in a multi-processing chain, each with the AUTOPOLL feature enabled. Emulators #2 through #5 are currently running while the user is communicating with emulator #1. The user puts emulator #1 into the background mode and waits for one of the other emulators to finish running. The entire session would occur as follows:

Enter: ID<CR>

Display:

```
ID
    #1
    TMS-9995    XDS    VERSION    1.3.0
```

So far, the user is talking to emulator #1.

Enter: BGND<CR>

Display:

```
BGND
AUTOPOLLING
???
```

Emulator #1 goes into the BGND mode and indicates that it is AUTOPOLLING. Another emulator (emulator #2, in this case) finishes processing, then comes into the foreground and displays:

```
    #2
    TMS-9995    XDS    VERSION    1.3.0
    TMF
    WP=0000   PC=F000   ST=2000
?
```

3.4 COMMAND: BP - Hardware Breakpoint

PURPOSE: Configures the hardware breakpoint conditions.

NOTE: The breakpoint-trace board must be installed to use this command. This command is used in conjunction with the BPM, BPIO, TRM, and TRIO commands.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
EVENT COUNT	0 - 7FFF	0
DELAY COUNT	0 - 7FF	0
ADDRESS MASK (ONES ENABLE) NUMBER OF EXTENDED ADDRESS BITS = 0 NUMBER OF EXTENDED ADDRESS BITS = 8	 0 - FFFF 0 - FFFFFF	 FFFF

PARAMETER DESCRIPTIONS:

EVENT COUNT

An event is defined as the satisfaction of all conditions of a specific breakpoint command (BPM or BPIO). The event count is decremented by one each time a breakpoint event is encountered. When the event count reaches zero the delay counter begins counting. If the event count is changed, any previous trace samples will be flushed from the trace memory when the next SS (Single Step Execution) or CRUN (Continue RUN) command is executed.

DELAY COUNT

Sets the number of trace samples that are stored after the event count reaches zero. When the number of trace samples specified by DELAY COUNT have been stored (after the EVENT COUNT reaches zero) a hardware breakpoint occurs. If the delay count is changed, any previous trace samples will be flushed from the trace memory when the next SS (Single Step Execution) or CRUN (Continue RUN) command is executed.

ADDRESS MASK (ONES ENABLE)

This hexadecimal bit mask allows all or part of the BREAKPOINT ADDRESSES specified in the BPM and/or BPIO command(s) to be ignored during comparison. For every bit value of 0, the corresponding address bit is ignored. A value of all 0's causes the entire address value to be ignored; a value of all F's causes the entire address value to be compared. (Refer to Appendix B for details on masking.)

INTERACTION WITH OTHER COMMANDS AND PARAMETERS

PARAMETER	COMMAND AFFECTED	PARAMETER AFFECTED
ADDRESS MASK	BPM	BREAKPOINT ADDRESS 1
ADDRESS MASK	BPM	BREAKPOINT ADDRESS 2
ADDRESS MASK	BPIO	BREAKPOINT ADDRESS #1
ADDRESS MASK	BPIO	BREAKPOINT ADDRESS #2

COMMANDS AND PARAMETERS THAT AFFECT BP PARAMETERS

INTERACTING COMMAND	INTERACTING PARAMETER	AFFECTED PARAMETER
INIT	EXT. ADDRESS BITS	BREAKPOINT ADDRESS 1
INIT	EXT. ADDRESS BITS	BREAKPOINT ADDRESS 2

EXAMPLE:

Situation: This example stops program execution immediately after the third occurrence of an event (events are defined by the BPM and/or BPIO commands). Execution stops immediately after the third event since the DELAY COUNT is set to zero; the ADDRESS MASK is enabled to allow only the exact address specified in the BPM or BPIO command to qualify as a breakpoint event. Refer to Section 7 for more examples of the BP command.

Enter: BP<CR>

Display:		Enter:
BP		
EVENT COUNT (0 - 7FFF)	= 0000	3<CR>
DELAY COUNT (0 - 7FF)	= 000	<CR>
ADDRESS MASK (ONES ENABLE)	= FFFF	<CR>

?

3.5 COMMAND: BPIO - Breakpoint, Input/Output

PURPOSE: To define a hardware breakpoint event based on address and data qualifications for I/O (CRU - Communications Register Unit) operations. All conditions specified must be met before the cycle will qualify as an event.

NOTE: The breakpoint-trace board must be installed to use this command. This command is used in conjunction with the BP command.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
QUALIFIER [OFF, IOA]	0 = OFF 1 = IOA	0
BREAKPOINT ADDRESS #1	0000 - FFFF	0
BREAKPOINT ADDRESS #2	0000 - FFFF	0
RANGE INDICATOR [0=NO, 1=YES]	0 = NO 1 = YES	0
EXTENDED DATA COMPARE BYTE NUMBER OF EXTENDED ADDRESS BITS = 0 NUMBER OF EXTENDED ADDRESS BITS = 8	 0 - FF 0	 0
EXTENDED DATA COMPARE MASK (ONE'S ENABLE) NUMBER OF EXTENDED ADDRESS BITS = 0 NUMBER OF EXTENDED ADDRESS BITS = 8	 0 - FF 0	 0

PARAMETER DESCRIPTIONS:

QUALIFIER [OFF, IOA]

This parameter restricts the I/O cycles which can qualify a breakpoint event.

OFF - Disqualifies any I/O access as a breakpoint event.

IOA - Allows any I/O access to qualify a breakpoint event.

BREAKPOINT ADDRESS #1

Allows a specific I/O address to qualify a breakpoint event.

PARAMETER DESCRIPTIONS (CONTINUED):

BREAKPOINT ADDRESS #2

Allows a second I/O address to qualify a breakpoint event. To select a single address, set RANGE INDICATOR to OFF and BREAKPOINT ADDRESS #1 to the same value as BREAKPOINT ADDRESS #2.

RANGE INDICATOR [0=NO, 1=YES]

NO - Each address entered above qualifies an individual breakpoint event.

YES - The addresses act as boundaries for a range of addresses as follows:

Case 1 (ADDRESS 1 < ADDRESS 2)

All addresses from ADDRESS 1 to ADDRESS 2 (inclusive) qualify breakpoint events. The addresses outside the range do not qualify.

Case 2 (ADDRESS 1 > ADDRESS 2)

No addresses between ADDRESS 1 and ADDRESS 2 (inclusive) qualify breakpoint events. The addresses outside the range do qualify.

EXT DATA COMPARE BYTE

External probe data must match this value (based on the mask below) for an access to qualify a breakpoint event.

EXT DATA COMPARE MASK (ONES ENABLE)

This hexadecimal bit mask allows all or part of the EXTERNAL DATA COMPARE BYTE to be ignored during comparison. For every bit value of 0 the corresponding data bit is ignored. A value of all 0's causes the entire probe value to be ignored; a value of all F's causes the entire probe value to be compared. (Refer to Appendix B for details on masking.)

COMMANDS AND PARAMETERS THAT AFFECT BPIO PARAMETERS

INTERACTING COMMAND	INTERACTING PARAMETER	AFFECTED PARAMETER
INIT	NUMBER OF EXT ADDRESS BITS	EXT DATA COMPARE BYTE EXT DATA MASK

EXAMPLES:

Situation 1: The following example sets a breakpoint at I/O (CRU) address 200.

Enter: BPIO<CR>

```
Display:                                              Enter:

BPIO
  QUALIFIER [ OFF, IOA ]                   = 0        IOA<CR>
  BREAKPOINT ADDRESS  #1                   = 0000     200<CR>
  BREAKPOINT ADDRESS  #2                   = 0000     200<CR>
  RANGE INDICATOR [0=NO, 1=YES]            = 0        NO<CR>
  EXT DATA COMPARE BYTE                    = 00       <CR>
  EXT DATA COMPARE MASK (ONES ENABLE)      = 00       <CR>

?
```

Situation 2: The following example sets a breakpoint on I/O accesses outside the address range of >200 through >202. (Note that address bit A15 is a "don't care" for I/O addresses.)

Enter: BPIO<CR>

```
Display:                                              Enter:

BPIO
  QUALIFIER [ OFF, IOA ]                   = 1        IOA<CR>
  BREAKPOINT ADDRESS  #1                   = 0200     202<CR>
  BREAKPOINT ADDRESS  #2                   = 0200     200<CR>
  RANGE INDICATOR [0=NO, 1=YES]            = 0        YES<CR>
  EXT DATA COMPARE BYTE                    = 00       <CR>
  EXT DATA COMPARE MASK (ONES ENABLE)      = 00       <CR>

?
```

3.6 COMMAND: BPM - Breakpoint, Memory

PURPOSE: To define a hardware breakpoint event based on address and data qualifications for memory accesses.

NOTE: The breakpoint-trace board must be installed to use this command. This command is used in conjunction with the BP command.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
QUALIFIER [OFF,MA, MR, MW, IAQ]	0 = OFF 1 = MA 2 = MR 3 = MW 4 = IAQ	0
BREAKPOINT ADDRESS #1		
NUMBER OF EXTENDED ADDRESS BITS = 0	0 - FFFF	0
NUMBER OF EXTENDED ADDRESS BITS = 8	0 - FFFFFF	0
BREAKPOINT ADDRESS #2		
NUMBER OF EXTENDED ADDRESS BITS = 0	0 - FFFF	0
NUMBER OF EXTENDED ADDRESS BITS = 8	0 - FFFFFF	0
RANGE INDICATOR [0=NO, 1=YES]	0 = NO 1 = YES	0
EMU DATA COMPARE WORD	0 - FFFF	0
EMU DATA COMPARE MASK (ONES ENABLE)	0 - FFFF	0
EXTENDED DATA COMPARE BYTE		
NUMBER OF EXTENDED ADDRESS BITS = 0	0 - FF	0
NUMBER OF EXTENDED ADDRESS BITS = 8	0	
EXTENDED DATA COMPARE MASK (ONES ENABLE)		
NUMBER OF EXTENDED ADDRESS BITS = 0	0 - FF	0
NUMBER OF EXTENDED ADDRESS BITS = 8	0	

PARAMETER DESCRIPTIONS:

QUALIFIER [OFF,MA, MR, MW, IAQ]

This parameter restricts the type of memory cycle that will qualify breakpoint events.

OFF Disqualifies any memory access as a breakpoint event.

MA Allows all memory accesses to qualify breakpoint events.

MW Allows only memory writes to qualify breakpoint events.

MR Allows only memory reads (including IAQ's) to qualify breakpoint events.

IAQ Allows only instruction acquisitions to qualify breakpoint events.

BREAKPOINT ADDRESS #1

Allows a specific memory address to qualify a breakpoint event.

BREAKPOINT ADDRESS #2

Allows a second memory address to qualify as a breakpoint event. To select a single address , set the RANGE INDICATOR to OFF and BREAKPOINT ADDRESS #1 to the same value as BREAKPOINT ADDRESS #2.

RANGE INDICATOR [0=NO, 1=YES]

NO - Each address entered above qualifies as an individual breakpoint event.

YES - The addresses act as boundaries for a range of addresses as follows:

Case 1 (ADDRESS 1 < ADDRESS 2)

All addresses from ADDRESS 1 to ADDRESS 2 (inclusive) qualify as breakpoint events. The addresses outside the range do not qualify.

Case 2 (ADDRESS 1 > ADDRESS 2)

No addresses between ADDRESS 1 and ADDRESS 2 (inclusive) qualify as breakpoint events. The addresses outside the range do qualify.

EMU DATA COMPARE WORD

Processor data must match this value (based on the mask below) for an access to qualify a breakpoint event.

PARAMETER DESCRIPTIONS (CONTINUED):

EMU DATA COMPARE MASK (ONES ENABLE)

This hexadecimal bit mask allows all or part of the EMU DATA COMPARE WORD to be ignored during comparison. For every bit value of 0 the corresponding data bit is ignored. A value of all 0's causes the entire data value to be ignored; a value of all F's causes the entire data value to be compared. (Refer to Appendix B for details on masking.)

EXT DATA COMPARE BYTE

External probe data must match this value (based on the mask below) for an access to qualify a breakpoint event.

EXT DATA COMPARE MASK (ONES ENABLE)

This hexadecimal bit mask allows all or part of the EXT DATA COMPARE BYTE to be ignored during comparison. For every bit value of 0 the corresponding data bit is ignored. A value of all 0's causes the entire probe value to be ignored; a value of all F's causes the entire probe value to be compared. (Refer to Appendix B for details on masking.)

COMMANDS AND PARAMETERS THAT AFFECT BPM PARAMETERS

INTERACTING COMMAND	INTERACTING PARAMETER	AFFECTED PARAMETER
INIT	NUMBER OF EXT ADDRESS BITS	BREAKPOINT ADDRESS EXT DATA COMPARE BYTE EXT DATA MASK
BP	ADDRESS MASK	BREAKPOINT ADDRESS

EXAMPLE:

Situation: The BPM command can be used to set a breakpoint on a specific data pattern when it is written into a specific address as shown below. The data pattern is specified as the hexadecimal value >DEAD (EMU DATA COMPARE WORD); the breakpoint address is specified as address >50 by setting both address parameters to 50 and the RANGE INDICTOR to NO. The EMU DATA COMPARE MASK is enabled to allow only an exact match with the compare word to qualify for a breakpoint event.

Enter: BPM<CR>

```
Display:                                                     Enter:

BPM
  QUALIFIER [ OFF, MA, MR, MW, IAQ ]          = 0            MW<CR>
  BREAKPOINT ADDRESS  #1                      = 0000         50<CR>
  BREAKPOINT ADDRESS  #2                      = 0000         50<CR>
  RANGE INDICATOR [0=NO, 1=YES]               = 0            NO<CR>
  EMU DATA COMPARE WORD                       = 0000         DEAD<CR>
  EMU DATA COMPARE MASK (ONES ENABLE)         = FFFF         <CR>
  EXT DATA COMPARE BYTE                       = 00           <CR>
  EXT DATA COMPARE MASK (ONES ENABLE)         = 00           <CR>

?
```

3.7 COMMAND: CASB - Clear All Software Breakpoints

PURPOSE: Clears all software breakpoints.

NO PARAMETERS.

EXAMPLE:

Situation: The following example clears all software breakpoints which are currently set and which have not been encountered during a run. Software breakpoints are set at five addresses as shown by the execution of the DSB command, below. The DSB command is executed after the CASB command to show that all breakpoints have been cleared.

Enter: DSB<CR>

Display:

```
DSB
    F010  F113  0034  F4FE  FFFE
```

Enter: CASB<CR>

Display:

```
CASB
```

Enter: DSB<CR>

Display:

```
DSB
?
```

3.8 COMMAND: CRUN - Continue RUN

PURPOSE: Continues execution of a program after the processor was halted.

NOTE

* CRUN differs from RUN under the following conditions:

  - CRUN leaves all data in the trace buffer intact and continues adding trace samples to the existing trace buffer. CRUN can be used to run from software breakpoint to software breakpoint so that more data can be added to the trace buffer. RUN, on the other hand, clears the trace buffer and starts storing trace samples with index 0000.

  - If a hardware breakpoint (HBP) stopped the processor, a CRUN will continue program execution until some other halt condition is encountered. Since all of the hardware breakpoint criteria have been satisfied, and the event counter and delay counter are both zero, , the HBP interrupt is essentially disabled. When CRUN is issued, new trace samples will be added to the existing trace buffer. Execution can be interrupted by any other interrupt procedure.

  - If a trace-memory-full (TMF) condition stops the processor, CRUN will continue executing as though the TRACE COUNT were set to zero. The TMF interrupt is essentially disabled. When CRUN is issued, an indefinite number of new trace samples will be added to the trace buffer. When the number of trace samples taken exceeds the trace buffer capacity, sample storage continues with index 0000. Execution may be interrupted by any other interrupt procedure.

* CRUN executes the same as RUN under either of the following conditions:

  - The EVENT COUNT and/or DELAY COUNT paraneters from the BP command are changed after an interrupt and before the CRUN is issued. The trace buffer is cleared and new trace samples start at the beginning of the buffer.

  - The trace count from the TR command is changed after and interrupt and before the CRUN is issued.

NO PARAMETERS.

EXAMPLES:

```
CRUN<CR>
  [Any output generated by the execution of the program is
   displayed.]
?
```

3.9 COMMAND: CSB - Clear One Software Breakpoint

PURPOSE: Clears a single software breakpoint at the address specified.

NOTE: CSB is normally used to clear a software breakpoint which was set in error, or to allow for additional breakpoints when the limit of ten has been reached. It is not necessary to clear every software breakpoint since each is automatically cleared when encountered during execution.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
BREAKPOINT ADDRESS	0 - FFFF	0

PARAMETER DESCRIPTIONS:

BREAKPOINT ADDRESS
This parameter specifies the memory location at which the software breakpoint will be cleared. This parameter value is shared with the SSB (Set Software Breakpoint) command.

EXAMPLE:

Situation: Software breakpoints are set at addresses F084, F0FA, F0BC, F010, and F000. The software breakpoints can be displayed by executing the DSB (Display Software Breakpoint) command.

Enter: DSB<CR>

Display:

```
DSB
    F0FA  F010  F000  F0B4  F0BC  F084
```

After displaying the breakpoints (using the DSB command), the user discovers that the breakpoint set at F010 should have been set at F012. The breakpoint can be cleared and reset as shown below. Note that the user erroneously attempts to clear a non-existent breakpoint, and an error message is displayed.

Enter: CSB<CR>

```
Display:                                   Enter:

CSB
  BREAKPOINT ADDRESS = F084                F020<CR>
ERROR 1145 NON-EXISTENT BREAKPOINT
```

Enter: CSB<CR>

```
Display:                                   Enter:

CSB
  BREAKPOINT ADDRESS = F020                F010<CR>
```

Enter: SSB(F012)<CR>

Enter: DSB<CR>

Display:

```
DSB
    F0FA   F000   F0B4   F0BC   F084   F012
```

3.10 COMMAND: DHS - Display Halt Status

PURPOSE: Displays a code that indicates why the emulator stopped running. The display may include more than one code.

NO PARAMETERS.

The codes that may be displayed are as follows:

Code	Meaning
HBP	Hardware breakpoint interrupted program execution. The trace header and the trace sample that were in the pipeline when the last event occurred are displayed. The program counter, status, and workspace pointer registers are not displayed.
SBP	Software breakpoint interrupted program execution.
TMF	The specified number of trace samples were taken.
KEY	A character typed from the keyboard interrupted program execution.
Z-MID	Zero Opcode. An unprogrammed software breakpoint interrupted program execution.
MULTI	Multi-processing configuration halted program execution.
PERR	A parity error occurred during data transfer.
POR	System is powered up and DHS is executed prior to a run.
RES	Operator pressed the XDS RESET button.

EXAMPLES:

Situation 1: The following example shows the results of executing the DHS command after initiating a RUN on a program, then pressing any key on the terminal keyboard:

Enter: RUN<CR>

Display:

```
RUN
    RUNNING
```

Enter: <any keystroke>

Display:

```
RUN
    RUNNING
    KEY
    WP=0000  PC=F200  ST=2000
```

Enter: DHS<CR>

Display:

```
DHS
    KEY
    WP=0000  PC=F200  ST=2000

?
```

Situation 2: The emulator is reset and DHS is executed prior to a RUN.

Enter: DHS<CR>

Display:

```
DHS
    RES
    WP=0000  PC=F200  ST=2000
```

Situation 3: This example shows the effects of the emulator encountering three halt conditions at once (HBP - Hardware Breakpoint, SBP - Software Breakpoint, and TMF - Trace Memory Full). The TMF indicates that the specified number of trace samples were taken and displays the event that caused the halt.

Enter: DHS<CR>

Display:

```
DHS
    HBP  SBP  TMF
    INDEX CYCLE QUAL  EXTQUALS     ADDR DATA  RVRS ASSEMBLY
          *EVT  IAQ   11111111     F100 1000  NOP

?
```

3.11 COMMAND: DIAG - Diagnostic Mode

PURPOSE: Places the emulator into diagnostic mode to establish communication with a Texas Instruments Regional Technology Center service technician.

NOTE: This command cannot be used in Multi-processing Mode. Refer to the Installation and Reference Guide for further information regarding the use of this command.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
ECHO ONLY [0=NO, 1=YES]	0 = NO 1 = YES	0

PARAMETER DESCRIPTIONS

ECHO ONLY [0=NO, 1=YES]
This parameter gives a Texas Instruments service technician access to the emulator using remote data communications equipment. When enabled, all entries from the terminal are echoed to the display terminal in the remote service station.

EXAMPLES:

Situation 1: The emulator is placed in the diagnostic mode with the ECHO ONLY function disabled as follows:

```
DIAG<CR>
  ECHO ONLY [0=NO, 1=YES] = 0 NO<CR>
```

The user has placed the emulator in the diagnostic mode, ECHO ONLY disabled, which means it is under the control of the Texas Instruments service technician. In this mode, the technician may send any command to the emulator and review the output on his terminal. The service technician enters the MESG command to send a message to the user, or the QDIAG command to quit the diagnostic mode. All commands and output are echoed to the user's terminal.

Situation 2: The emulator is placed in the diagnostic mode with the ECHO ONLY function enabled.

```
DIAG<CR>
  ECHO ONLY [0 = NO, 1 = YES] = 0 YES<CR>
```

The user retains control of the emulator. All commands and output are echoed to the service technician's terminal. The user may terminate the diagnostic session with QDIAG.

3.12 COMMAND: DIO - Display I/O (CRU)

PURPOSE: Allows the user to read CRU bits.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
BASE ADDRESS	0 - 7FFFF	0
NUMBER OF BITS (0=16, 1-F)	0 = 16 1 = 1 . . . . . . F = 15	0

PARAMETER DESCRIPTIONS:

BASE ADDRESS
Specifies base CRU (Communication Register Unit) address at which the read begins.

NUMBER OF BITS (0=16, 1 - F)
Specifies number of bits to be read from CRU. Set to 0 for field width of 16 bits. Set to 1 through F for field width of 1 through 15 bits (respectively).

EXAMPLE:

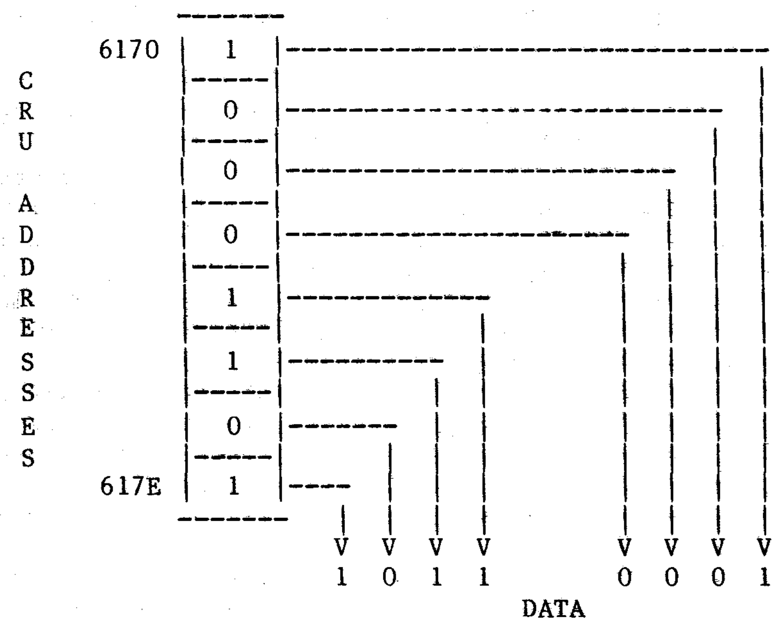
Situation: The following example reads 8 data bits beginning at CRU address >6170.

Enter: DIO<CR>

Display:

```
DIO
  BASE ADDRESS                   = 39B9 6170
  NUMBER OF BITS (0=16, 1 - F)   = 0000 8
    1011 0001
```

The data display is arranged starting with the bit from the highest address and continuing through the lowest. Data is displayed in groups of four for user convenience. Data transfer is as shown in the following diagram.


CRU ADDRESSES
6170
617E
1
0
0
0
1
1
0
1
1 0 1 1 0 0 0 1
DATA

3.13 COMMAND: DL - Download

PURPOSE: To set up parameters for a download of object code from an external device.

NOTE: All software breakpoints should be cleared using the CASB (Clear All Software Breakpoints) command prior to performing a download because all software breakpoints set before a download are still valid afterwards. Software breakpoints will work only when set on IAQ's (Instruction Acquisitions); if an uncleared software breakpoint falls on other than an IAQ in the downloaded code, it will result in confusion and may erronously appear to be a hardware failure.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
LOAD BIAS	0 - FFFF	0
FORMAT [0=TI, 1=TEK]	0 = TI 1 = TEK	0
DESTINATION[0=MEMORY, 1=PROM]	0=MEMORY 1=PROM	0
PROTOCOL [0=NONE, 1=TEK, 2=ASR, 3=VAX]	0 = NONE 1 = TEK 2 = ASR 3 = VAX	0
SOURCE [0=HOST, 1=USER]	0 = HOST 1 = USER	0

PARAMETER DESCRIPTIONS:

LOAD BIAS
The beginning address location where the downloaded data is to be stored. Values other than zero are applicable only to TI formats with relocatable object files.

FORMAT [0 = TI, 1 = TEK]
The format for the object code files.
VALUES: 0 = TI Normal ASCII or TI-Compressed Format
1 = Tektronix Hexadecimal Format

PARAMETER DESCRIPTIONS (CONTINUED):

DESTINATION [0 = MEMORY, 1 = PROM]

Specifies the destination of the downloaded data. If MEMORY is chosen, the downloaded data is stored in memory; if PROM is chosen the data is sent out Port C to a PROM programmer. If a download is attempted to space that is not RAM (i.e., if expansion memory is not mapped, emulator RAM is not turned on, and/or target memory does not exist), the downloaded data will not be stored.

VALUES:
- 0 = On-board emulator memory
- 1 = PROM programmer or logging device on Port C

PROTOCOL [0 = NONE, 1 = TEK, 2 = ASR, 3 = VAX]

Specifies the handshaking protocols used during data transfers. The selections for this parameter provide standard control-character codes for beginning downloads, ending downloads, beginning uploads, ending uploads, and pass-through characters.

VALUES:
- 0 = No handshake protocol predefined. Specific file control-characters are defined using the IHC command. IHC must be done before executing this command.

- 1 = Tektronix handshakes are enabled.
  ASCII "0" - Record transferred without error.
  ASCII "7" - Error detected. Retransmit record.
  Specific control-characters are defined using IHC command. IHC must be done before executing this command.

- 2 = ASR terminal handshake is enabled.
  CTRL-R - Start download
  CTRL-S - End Download
  CTRL-Q - Start upload
  CTRL-@ - End upload
  CTRL-P - Pass through next control character

- 3 = VAX or PDP-11 system handshake is enabled.
  CTRL-A - Start upload
  CTRL-B - End upload
  CTRL-V - Start download
  CTRL-W - End download
  CTRL-P - Pass through character

SOURCE [0=HOST, 1=USER]

Allows the user to define the source of the downloaded data.

VALUES:
- 0 = Host computer is declared the source of the data to be downloaded into the emulator.
- 1 = User's terminal is declared the source of the data to be downloaded into the emulator. This feature is provided for those using intelligent terminals or personal computers as terminals. Data stored in these terminals may then be

downloaded into the emulator through port A.

EXAMPLE:

Situation: The following example defines a download from a host computer with the data to be stored starting at memory location 0000. The object code will be in TI format (ASCII or compressed) and TEK handshaking protocol is specified. Note that prior to executing the DL command the IHC (Initialize Host Control) command must be executed to specify the characters that control the download.

Enter: DL<CR>

```
Display:                                                  Enter:

DL
  LOAD BIAS                                   = 0000      <CR>
  FORMAT      [0=TI, 1=TEK]                   = 0         TI<CR>
  DESTINATION [0=MEMORY, 1=PROM]              = 0         <CR>
  PROTOCOL    [0=NONE, 1=TEK, 2=ASR, 3=VAX]   = 0         TEK<CR>
  SOURCE      [0=HOST, 1=USER]                = 0         HOST<CR>

?
```

3.14 COMMAND: DM - Display Memory

PURPOSE: Provides a compact display of a section of the processor's memory space.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
START ADDRESS	0000 - FFFF	0
END ADDRESS	0000 - FFFF	0

PARAMETER DESCRIPTIONS:

START ADDRESS
The lowest address, in hexadecimal format, of the processor memory space to be displayed.

END ADDRESS
The highest address, in hexadecimal format, of the processor memory space to be displayed. Note that if the END ADDRESS is less than the START ADDRESS, only one memory location will be displayed.

EXAMPLES:

NOTE
The memory display can be made to pause at any time by pressing any key on the terminal keyboard except the ESCAPE key <ESC>. Resume the memory display after pausing by pressing any key on the terminal keyboard (except <ESC>). The ESCAPE key allows the user to exit from the display memory command.

Enter: DM<CR>

Display:		Enter:
DM		
START ADDRESS	= 0014	<CR>
END ADDRESS	= 001C	40<CR>

Display:

```
DM
  START ADDRESS    = 0014
  END ADDRESS      = 001C 40
   0014=3031 3233 3435 3637 3839 3A3B 3C20 2020   01 23 45 67 89 :; <
   0024=3D3E 3F40 4142 4344 4546 4748 4920 2020   => ?@ AB CD EF GH I
   0034=4A4B 4C4D 4E4F 5051 1A32 617A             JK LM NO PQ .. ..
```

The left-hand column of the memory display is the address of the first memory word in each row. Each row contains eight words unless the display completes before the row is full. ASCII characters corresponding to the data contained in the addresses are displayed to the right of the data/address display. Unprintable characters are shown in the ASCII display as periods.

3.15 COMMAND: DPS - Display Processor Status

PURPOSE: Displays the current contents of all processor registers. This provides a quick inspection of the processor's current status.

NO PARAMETERS

EXAMPLES:

Situation 1: The DPS command is executed with the power-up values in all variables and registers below.

Enter: DPS<CR>

Display:

```
DPS
   WP=0000  PC=F100  ST=2000

   R0 =  FFFF  R4 =  FFFF  R8  =  FFFF  R12 =  FFFF
   R1 =  FFFF  R5 =  FFFF  R9  =  FFFF  R13 =  FFFF
   R2 =  FFFF  R6 =  FFFF  R10 =  FFFF  R14 =  FFFF
   R3 =  FFFF  R7 =  FFFF  R11 =  FFFF  R15 =  FFFF

   LGT  AGT  EQ  C  OV  OP  INTM
    0    0   1   0  0   0    0
```

Situation 2: In this example, the workspace pointer is set to address F300. DPS is executed and the data contained in addresses >F300 through >F318 is shown in the workspace pointer registers.

Enter: WP=F300<CR>

Display:

```
WP=F300
?
```

Enter: DPS<CR>

Display:

```
DPS
    WP=F300  PC=F100  ST=2000

    R0 =  1230  R4 =  FADE  R8  =  0000  R12 =  0000
    R1 =  00AB  R5 =  DEAF  R9  =  0003  R13 =  0000
    R2 =  000D  R6 =  DADE  R10 =  3456  R14 =  0000
    R3 =  FEDE  R7 =  4321  R11 =  000F  R15 =  0000

    LGT  AGT  EQ  C  OV  OP  INTM
     0    0   1   0  0   0    0

?
```

The contents of each of the processor's registers are displayed along with the status bits from the status register:

WP	Workspace Pointer
PC	Program Counter
ST	Status Register

Status register bits:

LGT	Logical Greater Than
AGT	Arithmetic Greater Than
EQ	Equal
C	Carry
OV	Overflow
OP	Odd Parity
INTM	Interrupt Mask

3.16 COMMAND: DR - Display Registers

PURPOSE: Displays the current contents of the work space pointer, program counter, and status register.

NO PARAMETERS

EXAMPLE:

Enter: DR<CR>

Display:

```
DR
    WP  = ____    PC = ____    ST = ____
```

The contents of each of the emulator's registers are displayed.

WP	Workspace Pointer
PC	Program Counter
ST	Status Register

3.17 COMMAND: DSB - Display All Software Breakpoints

PURPOSE: Displays all software breakpoints currently set.

NO PARAMETERS

EXAMPLE:

Situation: In the following example, software breakpoints are set at addresses F084, F0FA, F0BC, F010, and F000. The user displays the breakpoints using the DSB command.

Enter: DSB<CR>

Display:

```
DSB
    F084  F0FA  F0BC  F010  F000
```

NOTE

When a software breakpoint is encountered during a run, the breakpoint is eliminated.

3.18 COMMAND: DT - Display Trace

PURPOSE: Displays a section of trace as specified by the user. This command is more convenient than IT (Inspect Trace) when looking at a short trace on a repetitive basis. Refer to Section 7 for details concerning the Breakpoint/Trace functions.

NOTE: The breakpoint-trace board must be installed to use this command.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
FIRST SAMPLE (0-7FE, 0=OLDEST 7FF=NEWEST)	0 - 7FF	0
NUMBER OF SAMPLES	0 - 7FF	0

PARAMETER DESCRIPTIONS:

FIRST SAMPLE (0-7FE, 0=OLDEST 7FF=NEWEST)
Specifies where the display starts. A value of 0 refers to the oldest sample. A value of 7FF specifies a special option which starts "NUMBER OF SAMPLES" from the end and shows the newest or most recent samples (instead of starting at a specified sample).

NUMBER OF SAMPLES
Specifies the number of samples to be displayed. If there are more samples than can be displayed on one screen, the display will scroll up the screen. If there are fewer samples available than specified, only those samples available will be displayed.

DISPLAY FORMAT:

The Display Trace output is formatted as follows:

INDEX	CYCLE	QUAL	EXTQUALS	ADDR	DATA	RVRS ASSEMBLY
000A	EVNT	IAQ	11111111	0018	38C2	MPY R2,R3

INDEX - Index of the trace.

CYCLE - EVNT shows that the conditions of an event have been satisfied.

*EVT means that this event caused the event count to go to zero. Depending on the delay count, this may not be the last event in the trace.

DISPLAY FORMAT (CONTINUED):

QUAL	-	(Qualifier) Identifies the type of trace cycle executed.
EXT QUALS	-	(External Qualifiers) The external data bits are displayed in binary. Any of the bits which are defined as extended address bits are not shown here.
ADDR	-	Value on address bus.
DATA	-	Value on data bus. When the trace involves memory cycles, the entire value on the bus is shown in hexadecimal format. If the trace is on I/O cycles, two 1-bit columns appear below the DATA heading: the bit on the left is the CRUOUT (Communication Register Unit Output) data; the bit on the right is the CRUIN (Communication Register Unit Input) data.
RVRS ASSEMBLY	-	(Reverse Assembly) On an IAQ line, the instruction is reverse assembled when no extra address bits have been declared. In this case, the data shown is as fetched from memory at display time, and may not correspond to the actual data traced.

EXAMPLES:

NOTE

To conserve space the displays in the following examples have been condensed. Notice that the trace indexes indicate the full range of traces requested; however, the intermediate traces between the second trace and the next-to-last trace have been replaced with elipsis.

Situation 1: Display the first >20 samples of trace.

Enter: DT<CR>

Display: Enter:

```
DT
  FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000  <CR>
  NUMBER OF SAMPLES                          = 000  20<CR>
```

Display:

```
DT
  FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000
  NUMBER OF SAMPLES                          = 000   20

    INDEX CYCLE QUAL  EXTQUALS      ADDR DATA  RVRS ASSEMBLY

    0000        IAQ   11111111      0000 1000  NOP
    0001        IAQ   11111111      0002 1000  NOP
     .           .       .           .    .     .
     .           .       .           .    .     .
     .           .       .           .    .     .

    001E        IAQ   11111111      003C 1000  NOP
    001F        IAQ   11111111      003E 1000  NOP

?
```

Situation 2: Display the last 10 trace samples:

Enter: DT<CR>

Display: Enter:

```
DT
  FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000   7FF<CR>
  NUMBER OF SAMPLES                          = 020   A<CR>
```

Display:

```
DT
  FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000   7FF
  NUMBER OF SAMPLES                          = 020   A

    INDEX CYCLE QUAL  EXTQUALS      ADDR DATA  RVRS ASSEMBLY

    03F6        IAQ   11111111      07EC 1000  NOP
    03F7        IAQ   11111111      07EE 1000  NOP
     .           .       .           .    .     .
     .           .       .           .    .     .
     .           .       .           .    .     .

    03FE        IAQ   11111111      07FC 1000  NOP
    03FF        IAQ   11111111      07FE 1000  NOP

?
```

Situation 3: The DT command is useful for repetitive execution. The following example always shows the last 10 trace samples taken (provided that the user responsed to the DT parameter prompts as shown in Situation 2, above, before this execution of the DT command).

Enter: MR.RUN DT.<CR>

Display:

```
MR
 WP = 4000
 PC = 0080
 ST = 0000
RUN
   RUNNING
   SBP
   WP=4000  PC=0222  ST=0000

DT
 FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 7FF
 NUMBER OF SAMPLES                          = 00A

   INDEX CYCLE QUAL  EXTQUALS      ADDR DATA  RVRS ASSEMBLY

   00C8        IAQ   11111111      0210 1000  NOP
   00C9        IAQ   11111111      0212 1000  NOP
    .           .       .            .    .     .
    .           .       .            .    .     .
    .           .       .            .    .     .

   00D0        IAQ   11111111      0220 1000  NOP
   00D1        IAQ   11111111      0222 1000  NOP
```

3.19 COMMAND: DTS - Display Trace Status

PURPOSE: Displays current values of the trace counter, event counter, and delay counter. This is useful to determine the proximity of execution to a hardware breakpoint, as well as the number of trace samples taken.

NO PARAMETERS.

EXAMPLES:

Situation 1: A typical display resulting from the execution of the DTS command is shown below; the display is useful if, for instance, the user wants to know the number of traces taken after a run.

Enter: DTS<CR>

Display:

```
DTS
    TRACE CNT     EVENTS LEFT    DELAYS LEFT
      03D2           0000           0000

?
```

In the above example, >3D2 trace samples have been stored.

Situation 2: Assume that the BP (Breakpoint) command has been set up with an EVENT COUNT of 127 and a DELAY COUNT of 25. A RUN is performed and the expected hardware breakpoint is not seen. The user stops the RUN and executes DTS. The display might appear as follows:

Enter: DTS<CR>

Display:

```
DTS
    TRACE CNT     EVENTS LEFT    DELAYS LEFT
      0014           0127           0025

?
```

Since none of the specified events have expired, the user should check the BPM or BPIO commands to see if his event definition is wrong.

3.20 COMMAND: DV - Display Value

PURPOSE: Provides a means for hexadecimal addition and subtraction. This is very useful when linked program modules are used and offsets need to be calculated.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
VALUE 1	0 - FFFFFF	0
VALUE 2	0 - FFFFFF	0

PARAMETER DESCRIPTIONS:

VALUE 1

An unsigned hexadecimal number up to 24 bits long.

VALUE 2

An unsigned hexadecimal number up to 24 bits long.

OUTPUT:

```
SUM   >000000          DIFFERENCE   >000000
```

The SUM is an unsigned 24-bit hexadecimal number resulting from the hexadecimal addition of VALUE 1 and VALUE 2. The DIFFERENCE is an unsigned hexadecimal number resulting from the larger value minus the smaller value. Both sum and difference are always represented as positive hexadecimal numbers.

EXAMPLES:

Situation 1: The offset of location >4062 in a module which begins at location >3C24 can be determined as shown below.

Enter: DV<CR>

```
Display:                                     Enter:

DV
  VALUE = 000000                             4062<CR>
  VALUE = 000000                             3C24<CR>
    SUM  >7C86       DIFFERENCE  >043E

?
```

NOTE

The DIFFERENCE displayed in the above example is the offset location.

Situation 2: The actual location of an instruction whose offset is >84 in a module beginning at >3C24 can be determined as follows.

Enter: DV<CR>

```
Display:                                          Enter:

DV
  VALUE = 004062                                  84<CR>
  VALUE = 003C24                                  <CR>
    SUM  >3CA8        DIFFERENCE  >3BA0

?
```

NOTE

The SUM displayed in the above example is the actual location of the instruction.

3.21 COMMAND: DW - Display Workspace

PURPOSE: Displays contents of all workspace registers in a compact format. This is more convenient than the register variables if several registers require simultaneous monitoring.

NO PARAMETERS

EXAMPLE:

Enter: DW<CR>

Display:

```
DW
     R0  = 48C2  R4  = 816F  R8  = 4382  R12 = 0F0F
     R1  = C321  R5  = 3030  R9  = 06D1  R13 = 9322
     R2  = A6C2  R6  = 9D92  R10 = 8A33  R14 = 7D7D
     R3  = 04A6  R7  = 318C  R11 = 436D  R15 = 218C
```

3.22 COMMAND: FILL - Fill Memory

PURPOSE: Fills a range of memory locations with a specified value.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
START ADDRESS	0 - FFFF	0
END ADDRESS	0 - FFFF	0
DATA	0 - FFFF	0

START ADDRESS
Specifies the beginning of the address range to be filled with data.

END ADDRESS
Specifies the end of the address range.

DATA
Specifies the value to be placed in memory.

EXAMPLE:

Situation: The target memory (addresses 0000 through FFFF) is filled with the data "0000" as shown below.

Enter: FILL<CR>

```
Display:                              Enter:

FILL
  START ADDRESS          = 0000      <CR>
  END   ADDRESS          = 0000      FFFF<CR>
  DATA                   = 0000      <CR>

?
```

3.23 COMMAND: FIND - Find Data

PURPOSE: Searches memory for specified data value and displays all locations containing that value.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
START ADDRESS	0 - FFFF	0
END ADDRESS	0 - FFFF	0
DATA	0 - FFFF	0
DATA MASK (ONES ENABLE)	0 - FFFF	0

PARAMETER DESCRIPTIONS

START ADDRESS
Specifies the address where the search for data will begin.

END ADDRESS
Specifies the end of the address range.

DATA
Specifies the value to be found in memory.

DATA MASK (ONES ENABLE)
Masks data so that any 0 bit in the DATA MASK is a don't care bit and the bits in the data value being compared are ignored. A 1 bit means that the data bit must match the compared data bit.

EXAMPLES:

Situation 1: This example finds all locations containing the value >4006.

Enter: FIND<CR>

```
Display:                                  Enter:

FIND
  START ADDRESS           = 0000         <CR>
  END   ADDRESS           = 0000         FFFF<CR>
  DATA                    = 0000         4006<CR>
  DATA MASK (ONES ENABLE) = FFFF         <CR>
    0048=4006
    1022=4006
    4068=4006
    9068=4006

?
```

NOTE

Execution of the FIND command can be aborted by pressing <ESC>.

Situation 2: The following example of the FIND command locates all bytes stored between locations >2000 and >2400 which contain the data >41 (high- or low-byte). This example uses two find operations and two manipulations of the data mask in order to limit the search to bytes rather than words.

Enter: FIND<CR>

```
Display:                                  Enter:

FIND
  START ADDRESS           = 0000         2000<CR>
  END   ADDRESS           = FFFF         2400<CR>
  DATA                    = 4006         41<CR>
  DATA MASK (ONES ENABLE) = FFFF         00FF<CR>
    2012=4C41
    2018=3741
    2184=0E41
    2348=8241
    23C4=4141

?
```

The preceding execution of FIND enables the low byte, finding and displaying the locations and data containing >41 in the low byte. The following execution enables the high byte, finding and displaying the locations and data containing >41 in the high byte. Note that location >23C4 appears in both executions, since both the high byte and the low byte contain >41.

Enter: FIND<CR>

```
Display:                                      Enter:

FIND
  START ADDRESS              = 2000           <CR>
  END   ADDRESS              = 2400           <CR>
  DATA                       = 0041           4100<CR>
  DATA MASK (ONES ENABLE)    = 00FF           FF00<CR>
    2002=4139
    23C4=4141

?
```

3.24 COMMAND: GHALT - Group Halt

PURPOSE: Interrupts RUN of all emulators in a multi-processing chain except the INDEPENDENTS that are not configured as START SYNCHRONOUS.

NOTE: This command is used in multi-processing operations.

NO PARAMETERS

EXAMPLES:

Situation: Four emulators are operating in a multi-processing chain; all are configured with the AUTOPOLLING feature turned ON; and all are set to start synchronously. Unit #4 is configured as the master and units #1 through #3 are slaves. Processing is started by calling the master unit (#4) into the foreground and issuing the GRUN command. Processing in the entire group is halted by calling the master unit (#4) into the foreground and issuing the GHALT command. The processor status of the individual emulators in the chain can be displayed by entering /n (where n is the number of the emulator). The halt status of the emulators can be displayed by entering #n (where n is the number of the emulator whose status is to be displayed).

Enter: #4

Display:

?

Enter: GRUN<CR>

Display:

?
AUTOPOLLING
???

Enter: GHALT<CR>

Enter: /1

Display:

WAITING TO OUTPUT

Enter: /2

Display:

```
WAITING TO OUTPUT
```

Enter: /3

Display:

```
WAITING TO OUTPUT
```

Enter: /4

Display:

```
WAITING TO OUTPUT
```

Enter: #4

Display:

```
Z-MID
WP=0000  PC=F000  ST=2000
```

3.25 COMMAND: GRUN - Group RUN

PURPOSE: Initiates the RUN condition in all emulators that have been configured for a synchronized start by the IMD command.

NOTE

This command is used in multi-processing operations.

NO PARAMETERS

EXAMPLE:

Situation: Four emulators are operating in a multi-processing chain; all are configured to start synchronously and have the AUTOPOLLING feature enabled. Unit #4 is configured as the master, and units #1 through #3 are slaves. Processing is started by calling the master unit into the foreground and issuing the GRUN command.

Enter: #4

Enter: GRUN<CR>

Display:

AUTOPOLLING
???

3.26 COMMAND: HELP - List Commands

PURPOSE: Displays a list of all commands, organized by function.

NO PARAMETERS

EXAMPLES:

Enter: HELP<CR>

Display:

```
                    TMS-9995    XDS    VERSION    1.3.0

  COMMANDS:
   INIT     IM      DR      RUN     BP      TR      HOST    IMP
   IPORT    DM      MR      CRUN    BPM     TRM     IHC     IMD
   IPRM     MM      DIO     SS      BPIO    TRIO    UL      ID
   ICC              MIO     SRR             TRIX    DL      BGND
   RCC                                      SOR
   RESTART

   MAP      FILL    XA      DPS     SSB     IT      LOG     GRUN
            FIND    XRA     DHS     DSB     DT      SNAP    TRUN
            DW              DTS     CSB             HELP    GHALT
                                    CASB            DV      THALT
  VARIABLES:
   WP       R       LGT     C       INTM
   PC               AGT     OV
   ST               EQ      OP

?
```

The commands are arranged in groups according to function:

1) Initialization - INIT, IPORT, IPRM, ICC, RCC, RESTART, and MAP

2) Memory - IM, DM, MM, FILL, FIND, and DW

3) Register - DR and MR

4) I/O - DIO and MIO

5) Assembler - XA and XRA

6) Run - RUN, CRUN, SS, and SRR

7) Run Status - DPS, DHS, and DTS

8) Breakpoint - BP, BPM, BPIO, SSB, DSB, CSB, and CASB

9) Trace - TR, TRM, TRIO, TRIX, SOR, IT, and DT

10) Host - HOST, IHC, UL, and DL

11) Miscellaneous - LOG, SNAP, HELP, and DV

12) Multi-processing- IMP, IMD, ID, BGND, GRUN, TRUN, GHALT, and THALT

3.27 COMMAND: HOST - Terminal Mode

PURPOSE: Places emulator into the terminal mode for communicating with a host computer system.

NO PARAMETERS

EXAMPLE:

Enter: HOST<CR>

Display:

```
HOST
  CONTROL-E EXITS
```

3.28 COMMAND: ICC - Initialize Cursor Control

PURPOSE: Maps the user's CRT cursor control keys into the emulator control functions. Only a single ASCII control character is accepted for each cursor control function. ICC does not support complex character strings (i.e., ANSI standard) used by some terminals for cursor control.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
DEPRESS KEY FOR CURSOR UP	UP-ARROW	+
DEPRESS KEY FOR CURSOR DOWN	DOWN-ARROW	-
DEPRESS KEY FOR CURSOR LEFT	LEFT-ARROW	<
DEPRESS KEY FOR CURSOR RIGHT	RIGHT-ARROW	>

NOTE

If the terminal uses a specific control character to move the cursor instead of ARROW keys, this control character may be used instead of the ARROW keys noted above. However, ONLY SINGLE CONTROL CHARACTERS THAT MOVE THE CURSOR MAY BE USED. If the terminal does not respond properly to the control characters entered via ICC, enter the RCC command and use the power-up values for cursor movement.

EXAMPLE:

Enter: ICC<CR>

```
Display:                                  Enter:

ICC
  DEPRESS KEY FOR CURSOR UP    = OK  <CURSOR UP key>
  DEPRESS KEY FOR CURSOR DOWN  = OK  <CURSOR DOWN key>
  DEPRESS KEY FOR CURSOR LEFT  = OK  <CURSOR LEFT key>
  DEPRESS KEY FOR CURSOR RIGHT = OK  <CURSOR RIGHT key>

?
```

The cursor control functions are now mapped into the cursor control keys on the terminal keyboard. Using the RCC command will exit the cursor control mode so that the original power-up keys are used to move the cursor.

3.29 COMMAND: ID - Identification

PURPOSE: Identifies the emulator by displaying the software logo. If in multi-processing mode, an identification number denoting the emulator's position in the chain is also displayed.

NO PARAMETERS.

EXAMPLES:

Situation 1: Emulator not in multi-processing mode.

Enter: ID<CR>

Display:

```
ID
     TMS9995  XDS VERSION 1.3.0
```

Situation 2: Emulator in multi-processing mode.

Enter: ID<CR>

Display:

```
ID
          #3
          TMS9995  XDS VERSION 1.3.0
```

The position of the emulator in the chain (#3 in the example above) is displayed.

3.30 COMMAND: IHC - Initialize Host Control

PURPOSE: Defines special control-key sequences that will inform the emulator's monitor that a download has started or terminated; that an upload has started or terminated; or that a control character is to be passed through to the host with no action taken. The last feature allows the user to pass a control character through to the host with no emulator action on the character. This command is a prerequisite to the DL and UL commands when selecting NONE or TEK. If VAX or ASR is selected the control characters are predefined, and the user need not execute IHC.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
DEPRESS KEY FOR DOWNLOAD START	CTRL-KEY	CTRL-V
DEPRESS KEY FOR DOWNLOAD END	CTRL-KEY	CTRL-W
DEPRESS KEY FOR UPLOAD START	CTRL-KEY	CTRL-A
DEPRESS KEY FOR UPLOAD END	CTRL-KEY	CTRL-Z
DEPRESS KEY FOR PASS THROUGH CHARACTER	CTRL-KEY	CTRL-P

EXAMPLE: The IHC command must be executed before initiating an upload, as shown below. The UL (Upload) command is executed after the IHC to transfer data from emulator memory locations 0000 through B000 to the host computer in TI data format, using no protocol.

Enter: IHC<CR>

```
Display:                                   Enter:

IHC
  DEPRESS KEY FOR DOWNLOAD START    = OK  <download start character>
  DEPRESS KEY FOR DOWNLOAD COMPLETE = OK  <download complete character>
  DEPRESS KEY FOR UPLOAD START      = OK  <upload start character>
  DEPRESS KEY FOR UPLOAD COMPLETE   = OK  <upload complete character>
  DEPRESS KEY FOR PASS THROUGH CHAR = OK  <pass through character>
```

Enter: UL<CR>

```
Display:                                                        Enter:

UL
  START ADDRESS                                      = 0000   <CR>
  END   ADDRESS                                      = B000   <CR>
  FORMAT [0=TI, 1=TICOMP, 2=TEK, 3=INTEL, 4=MR]      = 0      <CR>
  DATA   (0=WORD, 1=HI BYTE, 2=LO BYTE)              = 0      <CR>
  DESTINATION [0=HOST, 1=PROM, 2=USER]               = 0      <CR>
  PROTOCOL    [0=NONE, 1=TEK, 2=ASR, 3=VAX]          = 2      0<CR>

?
```

3.31 COMMAND: IM - Inspect Memory

PURPOSE: Allows the user to inspect the processor's memory space, moving back and forth and changing data at will.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
ADDRESS	0 - FFFF	0

PARAMETER DESCRIPTIONS:

ADDRESS
: Specifies the address location where memory inspection will begin.

CONTROL: Use <+>, <SPACE>, <DOWN-ARROW>, or<CR> to step forward to next address.

Use <-> or <UP-ARROW> to step backward to last address.

Use <Q><CR> or <ESC> to quit or abort IM and return to input mode.

NOTE

Cursor control characters are valid only after the ICC command is executed.

EXAMPLE:

Situation: As shown below, after IM is entered the current data at the ADDRESS is displayed. The user has the option of entering a new value into the displayed location, or any of the control characters mentioned above to move to a new location.

Enter: IM<CR>

```
Display:          Enter:

IM
  ADDRESS = 0000  80<CR>
```

```
Display:                          Enter:

IM
  ADDRESS = 0000  80
    0080=1000                     <CR>
   +0082=1000                     <CR>
   +0084=0000                     <CR>
   +0086=0200                     0<CR>
   +0088=1000                     200<CR>
   +008A=1000                     <->
   -0088=0200                     <->
   -0086=0000                     <->
   -0084=0000                     Q<CR>

?
```

3.32 COMMAND: IMD - Initialize Mode

PURPOSE: Initializes the mode of operation for emulators in multi-processing chain.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
PROCESSOR MODE [0=IND, 1=MASTER, 2=SLAVE]	0 = IND 1 = MASTER 2 = SLAVE	0
START SYNCHRONOUS [0=NO, 1=YES]	0 = NO 1 = YES	0

PARAMETER DESCRIPTIONS:

PROCESSOR MODE [0=IND, 1=MASTER, 2=SLAVE]
Defines the multi-processing operation mode for an individual emulator in the chain. The emulator may be specified as Independent, Master, or Slave. (See Section 6, MULTI-PROCESSING, for information regarding these operating modes.)

START SYNCHRONOUS [0=NO, 1=YES]
This parameter is used to establish a set of multi-processing emulators, which will start running simultaneously. Emulators with this parameter enabled will wait until all emulators so configured have been ordered to run before beginning their runs.

EXAMPLE:

Situation: Four emulators are connected for multi-processing emulation. After the IMP command is executed to initialize the multi-processing mode, IMD is used to configure the individual units to specific operating conditions. After IMP has been successfully executed, the user is in contact with unit #1.

Enter: IMD<CR>

```
Display:                                                Enter:

IMD
  PROCESSOR MODE [0=IND, 1=MASTER, 2=SLAVE] = 0   MASTER<CR>
  START SYNCHRONOUS [0=NO, 1=YES]           = 0   YES<CR>

?
```

Unit #2 is called into foreground, and the IMD command is executed for this emulator.

Enter: #2<CR>

Display:

```
?
```

Enter: IMD<CR>

Display:

```
IMD
  PROCESSOR MODE [0=IND, 1=MASTER, 2=SLAVE] = 0  SLAVE<CR>
  START SYNCHRONOUS [0=NO, 1=YES]           = 0  <CR>

?
```

Unit #3 is now called into foreground so that it may be set up.

Enter: #3<CR>

Display:

```
?
```

Enter: IMD<CR>

Display: Enter:

```
IMD
  PROCESSOR MODE [0=IND, 1=MASTER, 2=SLAVE] = 2  SLAVE<CR>
  START SYNCHRONOUS [0=NO, 1=YES]           = 0  <CR>

?
```

IMD need not be executed for unit #4 because the power-up values are acceptable for this unit. To see this, enter IMD!<CR>.

Enter: IMD!<CR>

Display:

```
IMD!
  PROCESSOR MODE [0=IND, 1=MASTER, 2=SLAVE] = 0
  START SYNCHRONOUS [0=NO, 1=YES]           = 0

?
```

3.33 COMMAND: IMP - Initialize Multi-Processor Mode

PURPOSE: Initializes the multi-processor mode of operation.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
AUTOPOLL [0=NO, 1=YES]	0 = NO 1 = YES	0

PARAMETER DESCRIPTIONS:

AUTOPOLL [0=NO, 1=YES]
This parameter initiates the autopolling sequence. The polling sequence will alternate between the keyboard and the emulator for signals that will interrupt of the program's execution.

EXAMPLES:

Situation 1: In this example, the multi-processing mode is initialized without the use of autopoll characters.

Enter: IMP<CR>

```
Display:                                    Enter:

IMP
  SEND AUTOPOLL CHAR? [0=NO, 1=YES] = 0   <CR>
    LAST EMU = 2

    TMS-9995   XDS   VERSION   1.3.0
```

Situation 2: In this example, the autopoll function is turned ON. At this point, the user may call up any emulator in the chain using #n (where n is the assigned identification number of the emulator being addressed). After an emulator is called into foreground, the user can control and issue monitor commands to it.

Enter: IMP<CR>

```
Display:                                    Enter:

IMP

  SEND AUTOPOLL CHAR? [0=NO, 1=YES] = 0   YES<CR>
    LAST EMU = 2

    TMS-9995   XDS   VERSION   1.3.0
?
```

3.34 COMMAND: INIT - Initialize Emulator

PURPOSE: Allows system configuration; controls global options such as clock source, memory options, and address configuration.

NOTE

After the INIT command is executed, with the target system selected as the clock source, Automatic First Wait State (AFWS) will be enabled. If the target clock frequency is below 16 MHz, AFWS can be removed by a normal target reset. If the target clock is equal to or above 16 MHz, AFWS will be forced.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
EMU: CLOCK SOURCE [0=INTERNAL, 1=TARGET]	0=INTERNAL 1=TARGET	0
EMULATOR RAM? [0=NO, 1=YES]	0 = NO 1 = YES	0
EXMEM: EXPANSION PAGE [0="A", 1="B"]	0 = "A" 1 = "B"	0
BP: NUMBER OF EXTENDED ADDRESS BITS (0-8)	0 - 8	0

PARAMETER DESCRIPTIONS:

EMU: CLOCK SOURCE [0=INTERNAL, 1=TARGET]

This parameter provides a choice for the emulator's frequency generator source. The power-up value is the 24-KHz clock on the emulator board. The TARGET system may also provide the frequency generator.

VALUES: 0 = Clock located on emulator board (24 KHz)
1 = External crystal located on target system

EMULATOR RAM? [0 = NO, 1 = YES]

This parameter allows the user to substitute the 1K or 7K emulator RAM for the target memory from addresses 0000 to 1BFF.

VALUES: 0 = NO Emulator RAM is not substituted for target memory.
1 = YES Emulator memory is "mapped" in.

PARAMETER DESCRIPTIONS (CONTINUED):

EXMEM: EXPANSION PAGE [0 = "A", 1 = "B"]

Two independent 64K byte pages of memory expansion are provided for the user. Only one of these is used at a time. Independent "maps" are associated with each page. Thus, one page can be used to map in certain areas and load them with data; then the other page can be selected, mapped, and loaded, without changing the information associated with the first page. The EXPANSION PAGE parameter selects one of the two pages.

VALUES: 0 = Selects page "A" of expansion memory.
1 = Selects page "B" of expansion memory.

NOTE

Selection of page "A" or page "B" (values 0 or 1, above) allows two users to swap software sessions and retain their programs independently of one another.

BP: NUMBER OF EXTENDED ADDRESS BITS (0 - 8)

This parameter configures the signals on the EXTENDED TRACE CABLE of the breakpoint/trace board. The power-up value of 0 indicates that the eight data probes are configured for data only. The address bits designated through this parameter reduce the number of data lines by the number selected. If the user selects four EXTENDED ADDRESS BITS, the number of data bits is reduced by four. In this case there are four address bits and four data bits.

EXAMPLES:

Situation 1: The following example biases-in 1K emulator RAM and configures the system to use the internal clock.

Enter: INIT<CR>

```
Display:                                                Enter:

INIT
    EMU: CLOCK SOURCE [0=INTERNAL, 1=TARGET]        = 0  <CR>
         EMULATOR RAM? [0=NO, 1=YES]                = 0  1<CR>
  EXMEM: EXPANSION PAGE [0="A", 1="B"]              = 0  <CR>
     BP: NUMBER OF EXTENDED ADDRESS BITS (0 - 8)    = 0  <CR>

?
```

EXAMPLES (CONTINUED):

Situation 2: The following example reconfigures the system to use the target system clock source and biases-out the emulator RAM.

Enter: INIT<CR>

Display: Enter:

```
INIT
    EMU: CLOCK SOURCE [0=INTERNAL, 1=TARGET]          = 0  1<CR>
         EMULATOR RAM? [0=NO, 1=YES]                  = 1  0<CR>
  EXMEM: EXPANSION PAGE [0="A", 1="B"]                = 0  <CR>
     BP: NUMBER OF EXTENDED ADDRESS BITS (0 - 8)      = 0  <CR>

?
```

NOTE

If a target clock is not present, the emulator will die and expansion memory will be lost. The RESET button on the emulator front panel must be pressed to change the clock source back to internal.

3.35 COMMAND: IPORT - Initialize EIA Port

PURPOSE: Configures the EIA RS-232-C Port for baud rate, parity, number of stop bits in the data signal, and the number of bits per character. These must conform to the requirements of the external device to which the port is connected.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
PORT [0=HOST ("D"), 1=LOG/PROM ("C")]	0 = HOST 1 = LOG	0
BAUD [19.2K, 9.6K, 4.8K, 2.4K, 1.2K, 600, 300, 110]	0 = 19.2K 1 = 9.6K 2 = 4.8K 3 = 2.4K 4 = 1.2K 5 = 600 6 = 300 7 = 110	0*
PARITY[0=OFF, 1=ODD, 2=EVEN]	0 = OFF 1 = ODD 2 = EVEN	0
STOP BITS [0=2, 1=1]	0 = 2 1 = 1	0
BITS/CHAR [0=7, 1=8]	0 = 7 1 = 8	0

* A zero is displayed as the power-up value, but the actual baud rate defaults to that of the terminal at Port A. This is established by the autobaud sequence on power-up.

PARAMETER DESCRIPTIONS:

PORT [0=HOST ("D"), 1=LOG/PROM ("C")]

This parameter allows the user to select the I/O Port on the XDS chassis to be configured. The HOST Port is Port D, and the LOG Port is Port C on the chassis. The LOG Port may be connected to a PROM programmer or a logging device.

PARAMETER DESCRIPTIONS (CONTINUED):

BAUD [19.2K, 9.6K, 4.8K, 2.4K, 1.2K, 600, 300, 110]
This parameter sets the rate for transmitting and receiving data.

VALUES:
0 = 19,200 bps
1 = 9,600 bps
2 = 4,800 bps
3 = 2,400 bps
4 = 1,200 bps
5 = 600 bps
6 = 300 bps
7 = 110 bps

PARITY [0=OFF, 1=ODD, 2=EVEN]
The PARITY parameter sets the parity bit to the appropriate value. If parity is turned OFF, the data transmitted will not have a parity bit added to the data stream. The monitor ignores the parity bit in data received.

STOP BITS [0=2, 1=1]
This parameter configures the number of stop bits in the data signal.

BITS/CHARACTER[0=7, 1=8]
This parameter configures the number of data bits in the data signal. The data field may consist of seven or eight bits.

NOTE

Transmission of data in TI-compressed format requires eight bits/character.

EXAMPLE:

Situation: The following example initializes EIA Port D with a baud rate of 9.6K bps, parity ignored, 2 stop bits, and 7 bits per character.

Enter: IPORT<CR>

Display: Enter:

```
IPORT
  PORT [0=HOST("D"), 1=LOG/PROM("C")]                     = 0  <CR>
  BAUD [19.2K, 9.6K, 4.8K, 2.4K, 1.2K, 600, 300, 110] = 0  9.6K<CR>
  PARITY [0=OFF, 1=ODD, 2=EVEN]                          = 0  <CR>
  STOP BITS (0=2, 1=1)                                   = 0  <CR>
  BITS/CHAR (0=7, 1=8)                                   = 0  <CR>
?
```

3.36 COMMAND: IPRM - Initialize Parameters

PURPOSE: Initializes all parameters to their power-up values. This command does not reinitialize emulator memory and the user's downloaded program remains intact after execution of IPRM.

NO PARAMETERS.

EXAMPLE:

Enter: IPRM<CR>

All parameters revert to their power-up values.

3.37 COMMAND: IT - Inspect Trace

PURPOSE: Allows the user to scroll back and forth through the trace samples. This command is more convenient than DT (Display Trace) for close examination of long traces.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
FIRST SAMPLE (0-7FE, 0=OLDEST 7FF=NEWEST)	0 - 7FF	0

PARAMETER DESCRIPTIONS:

FIRST SAMPLE (0 - 7FE, 0=OLDEST 7FF=NEWEST)
The value specifies which sample is first in the display. The first display includes as many samples as one screen will accommodate.

The IT command displays one page (19 lines) of trace samples starting at the sample specified in the FIRST SAMPLE parameter, or the last (newest) page of samples (if 7FF is entered); then waits for user input. The user may view previous or following samples by scrolling in the desired direction in the trace sample display; direction is controlled by entering <+> or <-> and <CR> or <SP>. The scroll number (jump size) defaults to 19, and can be changed by entering the direction indicator <+> or <->, a new jump value, and <CR> or <SP>. The command "remembers" the new scroll direction and jump number; so each time <CR> or <SP> is entered, the display scrolls in the same direction and jumps by the same amount. The following list defines user control of the IT command:

NOTE

Scrolling backward from the beginning of the trace buffer will simply cause the display to stay at the beginning of the buffer. Similarly, scrolling forward from the end of the buffer will cause the display to remain at the end of the buffer.

CONTROL: <CR> or <SP> — Displays one page of samples in the scroll direction and jump number from the present display.

CONTROL (CONTINUED):

<+><CR>, <+><SP>, <-><CR>, or <-><SP>	Changes scroll direction, jumps the scroll number in the new direction, and displays a new page of trace samples.
<+>nnn<CR> or <->nnn<CR>	Sets new scroll direction and scroll number; jumps by these; and displays a new page of trace samples.
Q	Quits IT command and continues command string.
<ESC>	Quits command and monitor returns to input mode.

NOTE

"nnn" is a hexadecimal number in the range 0 - FFF (that is, 0 - 4096 decimal). Values greater than >7FF (2047) will default to >7FF when processed.

DISPLAY FORMAT:

The Inspect Trace output is formatted as follows:

INDEX	CYCLE	QUAL	EXTQUALS	ADDR	DATA	RVRS ASSEMBLY
000A	EVNT	IAQ	11111111	0018	38C2	MPY R2,R3

INDEX - Index of the trace.

CYCLE - EVNT shows that the conditions of an event have been satisfied.

*EVT means that this event caused the event count to go to zero. Depending on the delay count, this may not be the last event in the trace.

QUAL - (Qualifier) Identifies the type of trace cycle executed.

EXT QUALS - (External Qualifiers) The external data bits are displayed in binary. Any bits which are defined as extended address bits are not shown here.

DISPLAY FORMAT (CONTINUED):

ADDR - Value on address bus.

DATA - Value on data bus. If the trace involves memory cycles, the data column will contain a hexadecimal representation of the data contained in the particular memory location. If the trace involves I/O (CRU) cycles, the data column will contain two single-bit data columns: the bit on the left is CRUOUT data, the bit on the right is CRUIN data.

RVRS ASSEMBLY - (Reverse Assembly) On an IAQ line, the instruction is reverse assembled when no extra address bits have been declared. In this case, the data shown is as fetched from memory at display time and may not correspond to the actual data traced.

EXAMPLE:

Situation: The following example shows a graphic representation of the trace memory.

* Part A shows the portion of trace memory containing valid trace samples in the shaded area.

* Part B shows the first display after responding to the FIRST SAMPLE prompt.

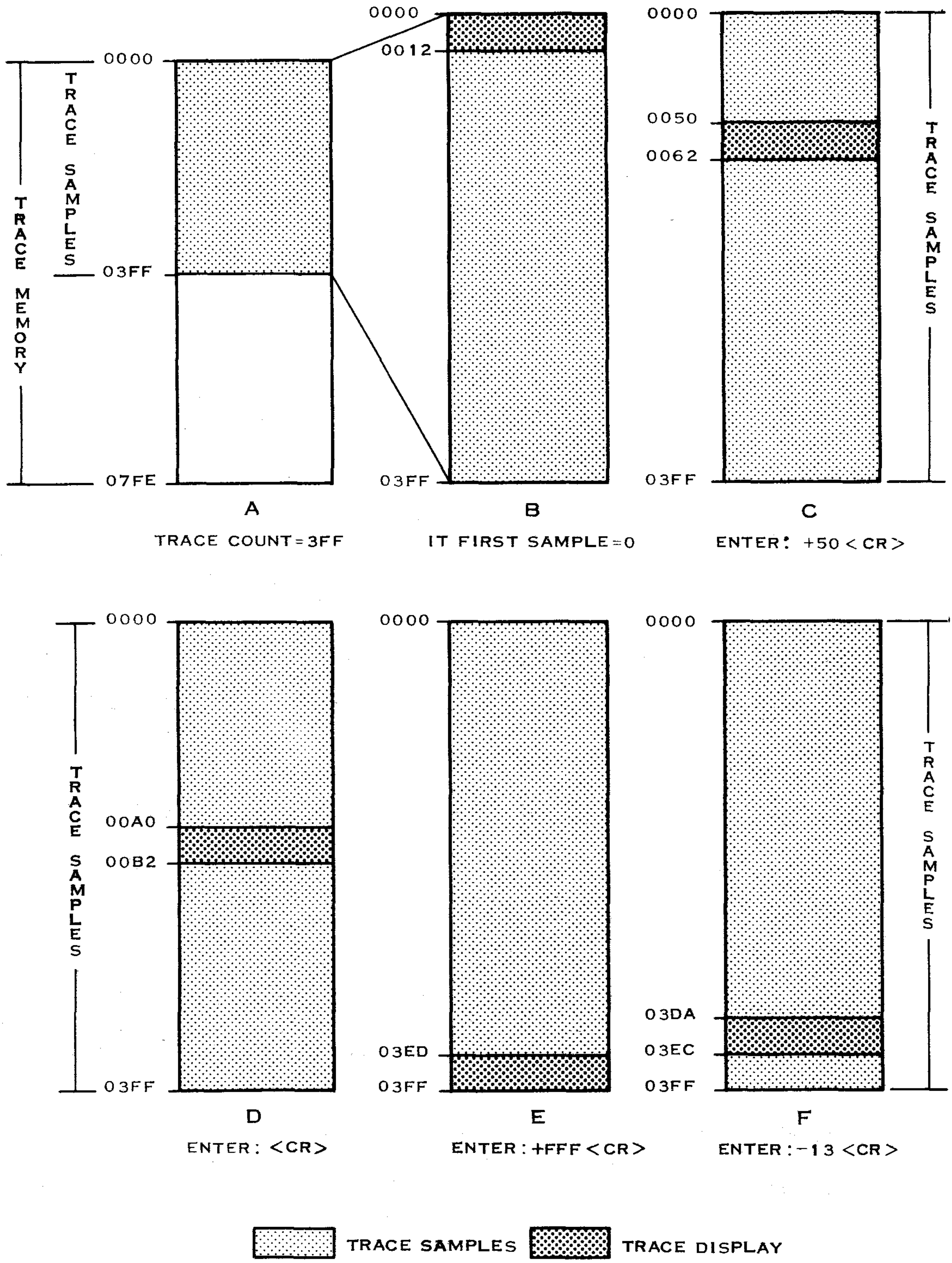
* Part C shows the display after entering +50<CR>.

* Part D shows the results of entering another <CR> (the displayed portion jumps to samples A0 through B2).

* The results of entering +FFF<CR> are shown in Part E (display jumps to the end of the valid trace samples in the trace memory).

* If -13<CR> is entered, the display jumps backwards to samples 3DA through 3EC, as shown in Part F.

* Entering Q<CR> quits the IT command and continues the command string in the input buffer.



TRACE MEMORY
TRACE SAMPLES
0000
03FF
07FE
A
TRACE COUNT=3FF
0000
0012
03FF
B
IT FIRST SAMPLE=0
0000
0050
0062
03FF
C
ENTER: +50 <CR>
0000
00A0
00B2
03FF
D
ENTER: <CR>
0000
03ED
03FF
E
ENTER: +FFF <CR>
0000
03DA
03EC
03FF
F
ENTER: -13 <CR>
TRACE SAMPLES
TRACE DISPLAY

3.38 COMMAND: LOG - Logging Device

PURPOSE: Enables or disables the logging function. The logging device is usually a printer.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
LOG DEVICE [0=OFF, 1=ON]	0 = OFF 1 = ON	0
LINE FEED WITH BACKSPACE [0=NO, 1=YES]	0 = OFF 1 = ON	0

PARAMETER DESCRIPTIONS:

LOG DEVICE [0=OFF, 1=ON]
This parameter turns the logging feature connected to Port C ON or OFF. If the logging device goes off-line, after approximately 5 seconds, the emulator turns this function OFF automatically.

LINE FEED WITH BACKSPACE [0=OFF, 1=ON]
When this feature is enabled, the emulator issues a linefeed with each backspace or series of backspaces.

COMMANDS AND PARAMETERS THAT AFFECT LOG PARAMETERS

COMMAND	PARAMETER	PARAMETER AFFECTED
DL (DOWNLOAD)	DESTINATION	LOG DEVICE
UL (UPLOAD)	DESTINATION	LOG DEVICE

EXAMPLES:

Situation 1: The user turns the log device (printer) ON and specifies that a backspace will NOT initiate a line feed. He executes the LOG command a second time to turn the log device OFF.

Enter: LOG<CR>

```
Display:                                               Enter:

LOG
  LOG DEVICE [0=OFF, 1=ON]                      = 0   1<CR>
  LINE FEED WITH BACKSPACE? [0=NO, 1=YES] = 0   <CR>
```

Enter: LOG<CR>

```
Display:                                          Enter:

LOG
  LOG DEVICE [0=OFF, 1=ON]                    = 0  <CR>
  LINE FEED WITH BACKSPACE? [0=NO, 1=YES]     = 0  <CR>
```

Situation 2: The following example illustrates what happens if a problem arises (such as the printer running out of paper) after the user turns the log device ON. This results in an error message five seconds after the printer goes off line. After this timeout period the LOG DEVICE parameter reverts to the OFF state, which can be seen by entering the command for review using the "!" terminator.

Enter: LOG<CR>

```
Display:                                          Enter:

LOG
  LOG DEVICE [0=OFF, 1=ON]                    = 0  1<CR>
  LINE FEED WITH BACKSPACE? [0=NO, 1=YES]     = 0  <CR>

ERROR 1112 LOG DEVICE OFF-LINE
```

Enter: LOG!<CR>

```
Display:

LOG
  LOG DEVICE [0=OFF, 1=ON]                    = 0
  LINE FEED WITH BACKSPACE? [0=NO, 1=YES]     = 0

?
```

3.39 COMMAND: MAP - Map Expansion Memory

PURPOSE: Allows the user to set up the address map for expansion memory or to unmap previously mapped memory.

NOTE: The expansion memory may be mapped into the processor's memory space as either RAM or ROM in 1K-byte blocks. If the base address specified is not on a 1K boundary, it is lowered to the nearest boundary. This command also checks to see if the 7K of emulator RAM has been mapped so as to prevent the user from mapping emulator addresses 0000 through 1BFF. (Refer to Section 4 for a more detailed description of the MAP command.)

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
BASE ADDRESS (1K BOUNDRY)	0000 - FC00	0
NUMBER OF 1KB BLOCKS IN HEX (0 - 40)	0 - 40	0
[0=UNMAP, 1=ROM, 2=RAM, 3=COPY]	0 = UNMAP 1 = ROM 2 = RAM 3 = COPY	0
NUMBER OF EXTRA WAIT STATES (0 - 3)	0 - 3	0

PARAMETER DESCRIPTIONS:

BASE ADDRESS (1K BOUNDARY)
Defines the start of a block

NUMBER OF 1KB BLOCKS IN HEX (0 - 40)
Defines the size of the block

[0=UNMAP, 1=ROM, 2=RAM, 3=COPY]
0 = Unmaps the block

1 = Maps the block as ROM

2 = Maps the block as RAM

3 = Maps the block as ROM and fills it with the contents of the target memory space

PARAMETER DESCRIPTIONS (CONTINUED):

NUMBER OF EXTRA WAIT STATES (0-3)

The expansion memory always runs with one "increment" wait state. If more are required, then the additional number of wait states may be selected by entering the appropriate number. The number of wait states only applies to the sections of memory biased-in by this command. The number of wait states required is determined by the customer's target system design and by the software resident in the target system. Since the memory expansion card is implemented with dynamic RAM, the memory cycles to the expansion memory may be delayed by one or two clocks to complete a refresh cycle. The expansion board appears to be implemented with static memory, and the refresh cycles are transparent if the user specifies two or more additional wait states.

EXAMPLE: The following example maps in two 1K byte blocks of RAM at address 4000.

Enter: MAP<CR>

```
Display:                                             Enter:

MAP
  BASE ADDRESS (1K BOUNDRY)            = 0000    4000<CR>
  NUMBER OF 1KB BLOCKS IN HEX (0-40)   = 0000    2<CR>
  [0=UNMAP, 1=ROM, 2=RAM, 3=COPY]      = 0000    2<CR>
  NUMBER OF EXTRA WAIT STATES (0-3)    = 0000    <CR>
```

3.40 COMMAND: MESG - Exchange Diagnostic Message

PURPOSE: Allows a Texas Instruments service technician to send or receive a text message while in the diagnostic mode. The technician can send the message if the ECHO ONLY is NOT enabled when initializing the diagnostic mode by using the DIAG command.

NO PARAMETERS.

NOTE

This command interacts with the ECHO ONLY parameter of the DIAG command which determines who originates the message.

EXAMPLES:

Situation 1: Diagnostic Mode (Echo enabled) on user's terminal.

User's terminal	Service Technician's Terminal
Display: ?	
Enter: DIAG (0)<CR>	
Display: ?	
Enter: MESG<CR> I AM HAVING PROBLEMS<CR> WITH THE MM COMMAND<CR><CR>	I AM HAVING PROBLEMS WITH THE MM COMMAND
	Enter: TRY ENTERING MM<CR><CR>
Display: TRY ENTERING MM	

The double carriage return transfers control of the message function to the other party. In this example, with Echo enabled, the user maintains control of the monitor and only he can issue commands to the emulator.

EXAMPLES (CONTINUED):

Situation 2: Echo Mode Disabled

Service Technician's Terminal	User's Terminal
Display: ?	
Enter: MESG<CR> I AM GOING TO DISPLAY<CR> MEMORY<CR> WITH THE MM COMMAND<CR><CR>	I AM GOING TO DISPLAY MEMORY WITH THE MM COMMAND Enter: GO AHEAD, I AM READY<CR><CR>
Display: GO AHEAD, I AM READY	

The service technician may now enter the MM command, and the display will appear on both terminals.

3.41 COMMAND: MIO - Modify I/O (CRU)

PURPOSE: Allows the user to write data to CRU bits.

PARAMETER	PARAMETER VALUE	POWER-UP VALUE
BASE ADDRESS	0 - FFFF	0
NUMBER OF BITS (0=16, 1 - F)	0 = 16 1 = 1 2 = 2 . . . . . . F = 15	0
DATA (LSB WRITTEN FIRST)		0

PARAMETER DESCRIPTIONS:

BASE ADDRESS
Specifies base CRU (Communication Register Unit) address where modification of the CRU bits begins.

NUMBER OF BITS (0=16, 1 - F)
Specifies number of bits to be read from CRU. Set to 0 for field width of 16 bits. Set to 1 through F for field width of 1 through 15 bits (respectively).

DATA (LSB WRITTEN FIRST)
The value entered into this parameter is sent to the addresses specified.

EXAMPLE:

Situation: Eight data bits are written to CRU addresses 1000 through 100E as follows:

Enter: MIO<CR>

```
Display:                                 Enter:

MIO
  BASE ADDRESS                  = 0000   1000<CR>
  NUMBER OF BITS (0=16, 1 - F)  = 0      8<CR>
  DATA (LSB WRITTEN FIRST)      = 00FF   0048<CR>
```

The data is written to the CRU starting with the LSB and continuing through the number of bits specified in the NUMBER OF BITS parameter. Data transfer is as shown in the diagram on the following page.

```
       >4                >8
  0  1  0  0        1  0  0  0
                                            -------
  |  |  |  |        |  |  |  |
  |  |  |  |        |  |  |   -------->|  0  |  1000
  |  |  |  |        |  |  |            |-----|          C
  |  |  |  |        |  |   ---------->|  0  |          R
  |  |  |  |        |  |               |-----|          U
  |  |  |  |        |   ------------->|  0  |
  |  |  |  |        |                  |-----|          A
  |  |  |  |         --------------->|  1  |          D
  |  |  |  |                           |-----|          D
  |  |  |   --------------------------->|  0  |          R
  |  |  |                              |-----|          E
  |  |   ----------------------------->|  0  |          S
  |  |                                 |-----|          S
  |   -------------------------------->|  1  |          E
  |                                    |-----|          S
   ----------------------------------->|  0  |  100E
                                        -------
```

3.42 COMMAND: MM - Modify Memory

PURPOSE: Allows user to modify the contents of memory at a specified address.

PARAMETER	PARAMETER VALUE	POWER-UP VALUE
ADDRESS	0000 - FFFF	0
DATA	0000 - FFFF	0

PARAMETER DESCRIPTIONS:

ADDRESS

Specifies a single emulator memory address that is modified with the entry of a data value.

DATA

The value entered into this parameter is sent to the emulator memory address specified in the ADDRESS parameter.

EXAMPLE:

Situation: The hexadecimal value FFFF is written to address >0014.

Enter: MM<CR>

```
Display:            Enter:

MM
  ADDRESS = 0000    14<CR>
  DATA    = 7FFF    FFFF<CR>
```

3.43 COMMAND: MR - Modify Registers

PURPOSE: Allows the user to change the contents of the workspace pointer, program counter, and status registers. The command does not prompt with the current values contained in the registers; the values shown are those last entered in parameter storage during execution of the MR command. To obtain the current values contained in the registers, the display registers command (DR) should be used.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
WP (Workspace Pointer)	0 - FFFF	0
PC (Program Counter)	0 - FFFF	0
ST (Status Register)	0 - FFFF	0

EXAMPLES:

Situation 1: Using the MR command to initialize the hardware registers to new default values of 4000 for the workspace pointer, 80 for the program counter, and 0000 for the status register.

Enter: MR<CR>

```
Display:        Enter:

MR
  WP = 0000     4000<CR>
  PC = 0014     80<CR>
  ST = 0000     <CR>
```

Situation 2: Using MR to establish program loop.

Enter: MR;RUN*<CR>

Display:

```
RUN
   RUNNING
   [Halt Status Display]
   WP=____  PC=____  ST=____
```

This procedure establishes a program loop; each execution starts with the register values that were assigned during the last MR command execution (as in Situation 1, above). The semicolon terminator suppresses the register display.

3.44 COMMAND: QDIAG - Quit Diagnostic Mode

PURPOSE: To inform the emulator that a diagnostic session is completed and to terminate the diagnostic operation. This command is sent by the Texas Instruments service technician if ECHO ONLY is disabled and by the user if ECHO ONLY is enabled.

NO PARAMETERS.

EXAMPLES:

Situation 1: ECHO ONLY parameter of DIAG command disabled.

TI Service Technician Enters: QDIAG<CR>

Display:

QDIAG
USER VDT IS NOW A COMMAND PORT

Situation 2: ECHO ONLY parameter of DIAG command is enabled.

User Enters: QDIAG<CR>

Display:

QDIAG

3.45 COMMAND: RCC - Reset Cursor Control

PURPOSE: Resets the cursor control functions to the original power-up values. This command is used to exit the cursor control functions defined by the ICC command.

NO PARAMETERS.

EXAMPLE:

Enter: RCC<CR>

Display:

RCC

Comment: The cursor control keys are no longer defined. The movement of the cursor is now controlled using the following keys:

- `-` To recall the preceding prompt.
- `+` To step forward to the next prompt.
- `<` To move the cursor left.
- `>` To move the cursor right.

3.46 COMMAND: RESTART - Restart Emulator

PURPOSE: Resets the emulator to the power-up, cold-start condition. This acts as a power cycle and eliminates the need to power-down to reset the emulator, thus prolonging the life of electronic components.

NOTE

When the RESTART command is executed, the system performs the AUTOBAUD routine and initializes the expansion memory to 0000.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
ARE YOU SURE [0=NO, 1=YES]	0 = NO 1 = YES	0

EXAMPLES:

Enter: RESTART<CR>

Display: Enter:

RESTART
ARE YOU SURE? [0=NO, 1=YES] = 0 YES<CR>

The emulator is now set to power-up condition. All parameter values and registers are set to this state.

Enter: <CR><CR>

After the user enters two carriage returns, the emulator performs the AUTOBAUD sequence and displays the HELP menu.

EXAMPLE (CONTINUED):

Display:

```
                          TEXAS INSTRUMENTS

                     TMS-9995   XDS   VERSION   1.3.0

   COMMANDS:
    INIT     IM       DR       RUN      BP       TR       HOST     IMP
    IPORT    DM       MR       CRUN     BPM      TRM      IHC      IMD
    IPRM     MM       DIO      SS       BPIO     TRIO     UL       ID
    ICC               MIO      SRR               TRIX     DL       BGND
    RCC                                          SOR
    RESTART

    MAP      FILL     XA       DPS      SSB      IT       LOG      GRUN
             FIND     XRA      DHS      DSB      DT       SNAP     TRUN
             DW                DTS      CSB               HELP     GHALT
                                        CASB              DV       THALT
   VARIABLES:
    WP       R        LGT      C        INTM
    PC                AGT      OV
    ST                EQ       OP

?
```

3.47 COMMAND: RUN - Execute Program

PURPOSE: To execute program currently residing in memory. All required register settings must be entered before issuing the RUN command.

NO PARAMETERS.

EXAMPLES:

Situation 1: The following example shows the output format for the RUN command.

```
RUN
   RUNNING
   [The reason for program halt is displayed]
   [REGISTER DISPLAY]
?
```

Situation 2: If any key is pressed while the program is executing, execution is halted and the display is as shown below:

```
Enter:  RUN<CR>
```

Display:

```
RUN
   RUNNING
```

```
Enter:  <any keystroke>
```

Display:

```
RUN
   RUNNING
   KEY
   WP=____ PC=____ ST=____
?
```

EXAMPLES (CONTINUED):

Situation 3: The following example shows the results of the emulator encountering a hardware breakpoint during the execution of a program. The "HBP" indicates that a hardware breakpoint caused the program to halt execution. In this case the last event is also displayed.

Enter: RUN<CR>

Display:

```
RUN
   RUNNING
   HBP
   INDEX CYCLE QUAL  EXTQUALS   ADDR DATA   RVRS ASSEMBLY
         *EVT  IAQ   11111111   0018 38C1   MPY  R2,R3
?
```

3.48 COMMAND: SNAP - Fixed Display

PURPOSE: To provide a 'snapshot' fixed display rather than a scrolling display format. SNAP is used only as the first command in a procedure. The output from the remaining commands scrolls up the screen and stops after the last command. On subsequent executions of the procedure, the commands' output again starts at the top of the procedure display, thus updating the screen.

NOTE

If the displayed output has more than 22 lines, the first lines will scroll off the top of the screen. Only the last 22 lines will be updated. Update information pertaining to the lines that scrolled off the screen will not be displayed.

FORMAT: SNAP,CMD1,CMD2,...,CMDn

NO PARAMETERS.

EXAMPLE:

Situation: The following example shows the results of executing a hypothetical procedure involving software breakpoints and single-step execution, with displays of halt status (DHS), memory (DM), and register R1. The SNAP command at the beginning of the procedure returns the cursor to the top of the display, which is updated with each successive execution of the procedure. The program used is listed following the example.

Enter: SNAP,SSB(26),SS,RUN,DHS,DM(100,11E) R1 *<CR>

EXAMPLE (CONTINUED):

Display:

```
SNAP,SSB(26),SS,RUN,DHS,DM(100,11E) R1 *
  SSB
  SS
     WP=0050  PC=001A  ST=D000   NEXT:  0018 38C2  MPY  R2,R3
  RUN
     RUNNING
  DHS
     SBP
     WP=0050  PC=0026  ST=D000
  DM
     0100=0000 0000 0000 0000 0000 0000 0000 0000 ................
     0110=0000 0000 0000 0000 0000 0000 0000 2468 ................
     R1=0001
```

At this point the display remains fixed with applicable portions being updated with each execution of the procedure. In this case the updated portions are: the program counter, status register, and next reverse assembly instruction displays (of the SS command); the status register display (of the DHS command); addresses 0100 and 011E (of the DM command); and the value of R1.

NOTE

Note that extra characters appear at the right of the memory display. These are the ASCII representations of the data stored in memory. If the ASCII representation is a nonprinting character, it is represented by a period.

The program upon which the procedures were performed is contained in locations 000A through 0028 (listed below).

ADDRESS	DATA	INSTRUCTION
000A	04C4	CLR R4
000C	0201	LI R1,10
000E	000A	
0010	0202	LI R2,>1234
0012	1234	
0014	0203	LI R3,>10
0016	0010	
0018	38C2	MPY R2,R3
001A	C803	MOV R3,@>100
001C	0100	
001E	C804	MOV R4,@>102
0020	011E	
0022	C0C1	MOV R1,R3
0024	0601	DEC R1
0026	16F8	JNE 0018
0028	10F0	JMP 000A

3.49 COMMAND: SOR - Set Opcode Range

PURPOSE: Allows a range of opcodes to be traced by the TRIX (Trace on Instruction Acquisition Extended) command.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
LOWER OPCODE BOUND (LS NIBBLE IGNORED)	0 - FFFF	0
UPPER OPCODE BOUND (LS NIBBLE IGNORED)	0 - FFFF	0
(0=DELETE RANGE, 1=ADD RANGE)	0 or 1	0

PARAMETER DESCRIPTIONS:

LOWER OPCODE BOUND (LS NIBBLE IGNORED)
The hexadecimal representation of the TMS9995 assembly language opcodes that are included as valid trace samples. The user can define a range of these values, which define the lower boundary of the range. The last four bits of the hexadecimal code word are treated as don't care bits and are ignored.

UPPER OPCODE BOUND (LS NIBBLE IGNORED)
This value defines the upper boundary of a range of opcode values to be traced. The last four bits of the hexadecimal code word are treated as don't care bits and are ignored.

(0=DELETE RANGE, 1=ADD RANGE)
This parameter allows the user to define one or more opcode ranges as valid trace samples. If the 1=ADD RANGE is selected, the range of values defined by the LOWER and UPPER OPCODE BOUNDS is added to the previously defined ranges. If the 0=DELETE RANGE option is selected, the specified opcode range is deleted as a valid trace sample.

EXAMPLE:

Situation: The following example illustrates how specific opcodes (or opcode ranges) can be qualified for tracing using the SOR and TRIX (Trace on Instruction Acquisition Extended) commands. Note that two ranges, >0400 through >0470 (Branch and Branch-and-Link through WP vector - B and BLWP) and >0680 through >06B0 (Branch-and-Link, BL) have been set using the SOR command. The DT (Display Trace) command is executed to show the trace samples that were stored during the execution of a hypothetical program.

EXAMPLE (CONTINUED):

NOTE

The TR (Trace) and TRIX commands must be executed to turn on the TRIX tracing option (in the TR command) and to enable the opcode qualifiers (in the TRIX command).

Enter: SOR<CR>

```
Display:                                                Enter:

SOR
  LOWER OPCODE BOUND (LS NIBBLE IGNORED) = 0000   400<CR>
  UPPER OPCODE BOUND (LS NIBBLE IGNORED) = 0000   470<CR>
  ( 0=DELETE RANGE, 1=ADD RANGE )        = 0      1<CR>
```

Enter: SOR<CR>

```
Display:                                                Enter:

SOR
  LOWER OPCODE BOUND (LS NIBBLE IGNORED) = 0400   680<CR>
  UPPER OPCODE BOUND (LS NIBBLE IGNORED) = 0470   6B0<CR>
  ( 0=DELETE RANGE, 1=ADD RANGE )        = 1      <CR>
```

Enter: DT<CR>

Display:

```
DT
    INDEX CYCLE QUAL  EXTQUALS      ADDR DATA  RVRS ASSEMBLY

    0000        IAQ   11111111      0002 0460  B    @>0006
    0001        IAQ   11111111      0008 06A0  BL   @>000C
    0002        IAQ   11111111      000E 0420  BLWP @>0012

?
```

3.50 COMMAND: SRR - Software Reset and Run

PURPOSE: Simulates a Target reset. The emulator fetches the reset trap vector from addresses 0000 and 0002 and begins execution as if a reset had occurred.

NO PARAMETERS.

EXAMPLE:

Situation: The DM (Display Memory) command shows the values at locations 0000 and 0002 (the reset trap vector). The RUN (executed by SRR) stops, and the display shows that the trace conditions have been met by displaying "TMF". The workspace pointer, program counter, and status register values are also displayed. The DT (Display Trace) command is executed to show the trace samples that were taken.

Enter: DM<CR>

```
Display:                Enter:

DM

   START ADDRESS = 0000   <CR>
   END   ADDRESS = 0000   2<CR>
    0000=4000 0080                                  @. ..
```

Enter: SRR<CR>

```
Display:

SRR
    RUNNING
    TMF
    WP=4000  PC=008E  ST=0000
```

Enter: DT<CR>

```
Display:                                               Enter:

DT
  FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 7FF    0<CR>
  NUMBER OF SAMPLES                          = 00A    20<CR>
```

EXAMPLE (CONTINUED):

Display:

```
DT
 FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 7FF  0
 NUMBER OF SAMPLES                          = 00A  20

   INDEX CYCLE QUAL  EXTQUALS      ADDR DATA  RVRS ASSEMBLY

   0000        IAQ   11111111      0080 1000  NOP
   0001        IAQ   11111111      0082 1000  NOP
   0002        IAQ   11111111      0084 1000  NOP
   0003        IAQ   11111111      0086 1000  NOP
   0004        IAQ   11111111      0088 1000  NOP
   0005        IAQ   11111111      008A 1000  NOP
   0006        IAQ   11111111      008C 1000  NOP
   0007        IAQ   11111111      008E 1000  NOP

?
```

3.51 COMMAND: SS - Single Step

PURPOSE: To execute one instruction of assembly program. The program counter and status register contents are displayed after the instruction is executed.

NOTE

If the user changes the value of the event count or delay count parameters of the BP (Hardware Breakpoint) command or the trace count parameter of the TR (Trace) command, any previous trace samples will be flushed from the trace memory on the next execution of the SS or CRUN (Continue RUN) command.

NO PARAMETERS.

EXAMPLES:

Situation 1: The result of executing one instruction of a program using the SS command is shown below.

Enter: SS<CR>

Display:

```
SS
    WP=4000  PC=00A8  ST=0000    NEXT:  00A8 1000  NOP

?
```

Situation 2: The following example shows the result of terminating the SS command with an asterisk (repeat terminator).

NOTE

When executing the SS command with the "*" terminator, the display can be paused by pressing any key; execution can be resumed by pressing any key. Press <ESC> to quit the SS* command.

EXAMPLES (CONTINUED):

Enter: SS*<CR>

Display:

```
SS*
     WP=4000  PC=00AA  ST=0000    NEXT:  00AA 1000  NOP
     WP=4000  PC=00AC  ST=0000    NEXT:  00AC 1000  NOP
     WP=4000  PC=00AE  ST=0000    NEXT:  00AE 1000  NOP
     WP=4000  PC=00B0  ST=0000    NEXT:  00B0 1000  NOP
     WP=4000  PC=00B2  ST=0000    NEXT:  00B2 1000  NOP
     WP=4000  PC=00B4  ST=0000    NEXT:  00B4 1000  NOP
     WP=4000  PC=00B6  ST=0000    NEXT:  00B6 1000  NOP
     WP=4000  PC=00B8  ST=0000    NEXT:  00B8 1000  NOP
     WP=4000  PC=00BA  ST=0000    NEXT:  00BA 1000  NOP
     WP=4000  PC=00BC  ST=0000    NEXT:  00BC 1000  NOP
     WP=4000  PC=00BE  ST=0000    NEXT:  00BE 1000  NOP
     WP=4000  PC=00C0  ST=0000    NEXT:  00C0 1000  NOP
     WP=4000  PC=00C2  ST=0000    NEXT:  00C2 1000  NOP
     WP=4000  PC=00C4  ST=0000    NEXT:  00C4 1000  NOP
     WP=4000  PC=00C6  ST=0000    NEXT:  00C6 1000  NOP
     WP=4000  PC=00C8  ST=

?
```

3.52 COMMAND: SSB - Set Software Breakpoint

PURPOSE: Sets software breakpoints that halt execution when they are encountered. The software breakpoints should be set on addresses that contain instructions. If, at any time, an attempt is made to execute that instruction, the processor will be halted before its execution. The software breakpoint will automatically be cleared at that time. A maximum of 10 software breakpoints can be defined and may only be set in modifiable areas in RAM.

NOTE

* It is possible to set software breakpoints on non-IAQ addresses; however, they will not function as breakpoints. Instead, they will cause the emulator to pick up erroneous immediate data, indexes, etc. The user should be extremely careful to ensure that the BREAKPOINT ADDRESS is an IAQ address.

* All existing software breakpoints should be cleared, using the CASB (Clear All Software Breakpoints) command, prior to performing a download because all software breakpoints existing at the time of the download are still valid after the download is performed.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
BREAKPOINT ADDRESS	0 - FFFF	0

PARAMETER DESCRIPTIONS:

BREAKPOINT ADDRESS

This parameter specifies a location in memory, that will be defined as a software breakpoint. If the executing program reaches the instruction contained in this address, the processor will be halted. An error is generated if the user attempts to set the breakpoint in an unmodifiable address (e.g., ROM).

INTERACTION WITH OTHER COMMANDS AND PARAMETERS

PARAMETER	COMMAND AFFECTED	PARAMETER AFFECTED
BREAKPOINT ADDRESS	CSB	BREAKPOINT ADDRESS

EXAMPLE:

Situation: The following example sets software breakpoints at addresses F084, F0FA, F0BC, F010, and F000. The DSB (Display Software Breakpoint) command is included in this example to illustrate the setting of the breakpoints. Note also that the parenthesized mode of entry is used to set some of the breakpoints.

Enter: SSB<CR>

Display: Enter:

```
SSB
  BREAKPOINT ADDRESS = 0000 F084<CR>
```

Enter: DSB<CR>

Display:

```
DSB
    F084
```

Enter: SSB(F0FA)SSB(F0BC)SSB(F010)SSB(F000)<CR>

Display:

```
SSB(F0FA)SSB(F0BC)SSB(F010)SSB(F000)
  SSB
  SSB
  SSB
  SSB
```

Enter: DSB<CR>

Display:

```
DSB
    F084   F0FA   F0BC   F010   F000
```

3.53 COMMAND: THALT - Total Halt

PURPOSE: Interrupts RUN or Background operations of all emulators in a multi-processing chain.

NO PARAMETERS.

EXAMPLE:

Situation: Four emulators are operating in a multi-processing chain. Unit #1 is configured as master; units #2 and #3 are slaves; and the group is set to start synchronously. Unit #4 is operating as an independent and is not set to start synchronously. Units #1 and #4 are running. The entire multi-processing chain can be stopped by issuing the THALT command. The processor status of the individual emulators can be displayed by entering /n (where n is the number of the unit in the chain); halt status of any of the individual units can be displayed by entering #n (where n is the number of the unit in the chain).

Enter: THALT<CR>

Enter: /1

Display:

READY TO OUTPUT

Enter: /2

Display:

READY TO OUTPUT

Enter: /3

Display:

READY TO INPUT

Enter: /4

EXAMPLE (CONTINUED):

Display:

```
    READY TO INPUT
```

Enter: #1

Display:

```
    Z-MID
    WP=0000  PC=F000  ST=2000
```

Enter: #4

Display:

```
    TMF
    WP=F300  PC=0040  ST=2000
```

All emulators operating in multi-processing mode are interrupted, and one of the emulators assumes control.

3.54 COMMAND: TR - Trace

PURPOSE: Defines which cycles in the user program to trace during execution.

NOTE

The breakpoint-trace board must be installed to use this command.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
TRACE COUNT (1-7FF, 0=INFINITE)	0 - 7FF	0
ADDRESS MASK (ONES ENABLE)		
NUMBER OF EXTENDED ADDRESS BITS = 0	0 - FFFF	FFFF
NUMBER OF EXTENDED ADDRESS BITS = 8	0 - FFFFFF	FFFFFF
TRACE MODE [0=TRM & TRIO, 1=TRIX]	0 = TRM & TRIO 1 = TRIX	0

PARAMETER DESCRIPTIONS:

TRACE COUNT (1 - 7FF, 0=INFINITE)

This parameter specifies the number of samples to be counted in a trace. Each valid sample causes the count to decrement by one. When the count reaches zero, program execution is interrupted and the TMF code is displayed. TMF indicates that the reason for the interrupt was the expiration of the trace memory count. An entry of 0 means to wrap-around (don't stop). If the trace count is changed, any previous trace samples will be flushed from the trace memory when the next SS (Single Step Execution) or CRUN (Continue RUN) command is executed.

ADDRESS MASK (ONES ENABLE)

This hexadecimal bit mask allows all or part of the ADDRESS parameters in the TRM, TRIO, and TRIX commands to be ignored during comparison. For every bit value of 0 the corresponding address bit is ignored. A value of all 0's causes the entire address value to be ignored; a value of all F's causes the entire address value to be compared. (Refer to Appendix B for details on masking.)

TRACE MODE [0=TRM & TRIO, 1=TRIX]

This parameter specifies the tracing mode to be used.

VALUES:
- 0 = Trace on memory or I/O cycle.
- 1 = Trace some or all of the extra cycles associated with instructions (IAQ's).

INTERACTION WITH OTHER COMMANDS AND PARAMETERS

PARAMETER	COMMAND AFFECTED	PARAMETER AFFECTED
ADDRESS MASK	TRM	TRACE ADDRESS #1
ADDRESS MASK	TRM	TRACE ADDRESS #2
ADDRESS MASK	TRIX	TRACE ADDRESS #1
ADDRESS MASK	TRIX	TRACE ADDRESS #2
TRACE MODE	TRM	All parameters
TRACE MODE	TRIO	All parameters
TRACE MODE	TRIX	All parameters

EXAMPLE:

Situation: The following execution of the TR command sets the TRACE COUNT (number of samples taken) to 10, keeps the ADDRESS MASK fully enabled (allowing only those addresses which match the compare value to qualify for tracing), and specifies the TRM & TRIO TRACE MODE.

Enter: TR<CR>

```
Display:                                               Enter:

TR
  TRACE COUNT (1 - 7FF, 0=INFINITE)        = 000    10<CR>
  ADDRESS MASK (ONES ENABLE)               = FFFF   <CR>
  TRACE MODE [0=TRM & TRIO, 1=TRIX]        = 0      <CR>
```

3.55 COMMAND: TRIO - Trace, Input/Output

PURPOSE: To set up the tracing of I/O (CRU) cycles.

NOTE: The breakpoint-trace board must be installed to use this command. This command is used in conjunction with the TR command.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
QUALIFIER [OFF, IOA]	0 = OFF 1 = IOA	0
TRACE ADDRESS #1	0 - FFFF	0
TRACE ADDRESS #2	0 - FFFF	0
RANGE INDICATOR [0=NO, 1=YES]	0 = NO 1 = YES	0
EXTENDED DATA COMPARE BYTE NUMBER OF EXTENDED ADDRESS BITS = 0 NUMBER OF EXTENDED ADDRESS BITS = 8	 FF 0	0
EXTENDED DATA COMPARE MASK NUMBER OF EXTENDED ADDRESS BITS = 0 NUMBER OF EXTENDED ADDRESS BITS = 8	 FF 0	0

PARAMETER DESCRIPTIONS:

QUALIFIER [OFF, IOA]
This parameter defines the primary qualifying criteria for I/O traces. The OFF value disqualifies all I/O accesses as trace samples; the IOA value allows all I/O accesses to qualify as trace samples.

TRACE ADDRESS #1
Allows a specific I/O address to qualify as a trace sample.

TRACE ADDRESS #2
Allows a second I/O address to qualify as a trace sample. To select a single address , set the RANGE INDICATOR to OFF and TRACE ADDRESS #1 to the same value as TRACE ADDRESS #2.

PARAMETER DESCRIPTIONS (CONTINUED):

RANGE INDICATOR [0=NO, 1=YES]

NO - Each address entered above qualifies as an individual trace sample.

YES - The addresses act as boundaries for a range of addresses as follows:

Case 1 (ADDRESS 1 < ADDRESS 2)
All addresses from ADDRESS 1 to ADDRESS 2 (inclusive) qualify as trace samples. The addresses outside the range do not qualify.

Case 2 (ADDRESS 1 > ADDRESS 2)
No addresses between ADDRESS 1 and ADDRESS 2 (inclusive) qualify as trace samples. The addresses outside the range do qualify.

EXT DATA COMPARE BYTE

External probe data must match this value (based on the mask below) for an access to qualify as a trace sample.

EXT DATA COMPARE MASK (ONES ENABLE)

This hexadecimal bit mask allows all or part of the EXT DATA COMPARE BYTE to be ignored during comparison. For every bit value of 0 the corresponding data bit is ignored. A value of all 0's causes the entire probe value to be ignored; a value of all F's causes the entire probe value to be compared. (Refer to Appendix B for details on masking.)

COMMANDS AND PARAMETERS THAT AFFECT TRIO PARAMETERS

INTERACTING COMMAND	INTERACTING PARAMETER	AFFECTED PARAMETER
INIT	NUMBER OF EXT ADDRESS BITS	EXT DATA COMPARE BYTE
INIT	NUMBER OF EXT ADDRESS BITS	EXT DATA MASK
TR	ADDRESS MASK	TRACE ADDRESS #1
TR	ADDRESS MASK	TRACE ADDRESS #2

EXAMPLE: The following execution of the TRIO command sets the TRIO QUALIFIER to ON (all I/O cycles to be traced) and sets the TRACE ADDRESS range to include all addresses between 0000 and FFFF. The DT (Display Trace) command displays the traces performed on the I/O cycles.

Situation: The TR (Trace) command was previously executed to set the trace conditions to a TRACE COUNT of 14 (>14 cycles traced) and to set the TRACE MODE to trace on memory and/or I/O cycles (TRM & TRIO). (Memory cycles are not traced in this example because the TRM command is initially OFF - QUALIFIER=0.) If the TRM command is used, memory cycles are also traced.

EXAMPLE (CONTINUED):
Enter: TRIO<CR>

```
Display:                                              Enter:

TRIO
  QUALIFIER [ OFF, IOA ]                   = 0        IOA<CR>
  TRACE ADDRESS  #1                        = 0000     <CR>
  TRACE ADDRESS  #2                        = 0000     FFFF<CR>
  RANGE INDICATOR [0=NO, 1=YES]            = 0        YES<CR>
  EXT DATA COMPARE BYTE                    = 00       <CR>
  EXT DATA COMPARE MASK (ONES ENABLE)      = 00       <CR>
```

Enter: DT<CR>

```
Display:                                                            Enter:

DT
  FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000                 <CR>
  NUMBER OF SAMPLES                          = 000                 7FE<CR>
    INDEX CYCLE QUAL  EXTQUALS      ADDR DATA  RVRS ASSEMBLY

    0000        I/O   11111111      0000 0  1
    0001        I/O   11111111      0002 1  1
    0002        I/O   11111111      0002 0  1
    0003        I/O   11111111      0004 0  1
    0004        I/O   11111111      0006 0  1
    0005        I/O   11111111      0008 0  1
    0006        I/O   11111111      000A 0  1
    0007        I/O   11111111      000C 0  1
    0008        I/O   11111111      000E 0  1
    0009        I/O   11111111      0010 0  1
    000A        I/O   11111111      0002 0  1
    000B        I/O   11111111      0004 1  1
    000C        I/O   11111111      0004 0  1
    000D        I/O   11111111      0006 0  1
    000E        I/O   11111111      0008 0  1
    000F        I/O   11111111      000A 0  1
    0010        I/O   11111111      000C 0  1
    0011        I/O   11111111      000E 0  1
    0012        I/O   11111111      0010 0  1
    0013        I/O   11111111      0012 0  1
```

NOTE

The DATA column of the DT display contains two single-bit columns of data. These are CRUOUT data (on the left) and CRUIN data (on the right). When I/O cycles are traced, no indication is given as to whether the cycles are read or write.

3.56 COMMAND: TRIX - Trace on Instruction Acquisition Extended

PURPOSE: Qualifies trace based only on IAQ cycle, but allows tracing of other cycles associated with the traced IAQ. This allows all or some of the cycles associated with a particular group of instructions to be traced.

NOTE: The breakpoint-trace board must be installed to use this command.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
OPCODE QUALIFIERS? [0=NO, 1=YES]	0 = NO 1 = YES	0
MEMORY QUALIFIER [OFF,MA, MR, MW,]	0 = OFF 1 = MA 2 = MR 3 = MW	0
I/O QUALIFIER [OFF, IOA]	0 = OFF 1 = IOA	0
TRACE ADDRESS #1		
NUMBER OF EXTENDED ADDRESS BITS = 0	0 - FFFF	0
NUMBER OF EXTENDED ADDRESS BITS = 8	0 - FFFFFF	0
TRACE ADDRESS #2		
NUMBER OF EXTENDED ADDRESS BITS = 0	0 - FFFF	0
NUMBER OF EXTENDED ADDRESS BITS = 8	0 - FFFFFF	0
RANGE INDICATOR [0=NO, 1=YES]	0 = NO 1 = YES	0
EXTENDED DATA COMPARE BYTE		0
NUMBER OF EXTENDED ADDRESS BITS = 0	FF	
NUMBER OF EXTENDED ADDRESS BITS = 8	00	
EXTENDED DATA COMPARE MASK		0
NUMBER OF EXTENDED ADDRESS BITS = 0	FF	
NUMBER OF EXTENDED ADDRESS BITS = 8	00	

PARAMETER DESCRIPTIONS:

OPCODE QUALIFIERS? [0=NO, 1=YES]
This parameter allows the opcode ranges set by the SOR command to determine which instructions are traced.

PARAMETER DESCRIPTIONS (CONTINUED):

MEMORY QUALIFIER [OFF, MA, MR, MW]

If the base instruction (IAQ) qualifies as a trace (based on opcode, address, and external data) then other memory cycles of the instruction are traced based on this prompt.

OFF = No memory cycles associated with the instruction are traced.

MA = All memory cycles associated with the instruction are traced.

MR = Any memory reads associated with the instruction are traced.

MW = Any memory writes associated with the instruction are traced.

I/O QUALIFIER [OFF, IOA]

If the base instruction (IAQ) qualifies as a trace (based on opcode, address, and external data) then I/O cycles of the instruction are traced based on this prompt.

OFF = No I/O cycles associated with the instruction are traced.

IOA = All I/O cycles associated with the instruction are traced.

TRACE ADDRESS #1

Allows a specific IAQ address to qualify as a trace sample.

TRACE ADDRESS #2

Allows a second IAQ address to qualify as a trace sample. To select a single address , set the RANGE INDICATOR to OFF and TRACE ADDRESS #1 to the same value as TRACE ADDRESS #2.

RANGE INDICATOR [0=NO, 1=YES]

NO - Each address entered above qualifies as an individual trace sample.

YES - The addresses act as boundaries for a range of addresses as follows:

Case 1 (ADDRESS 1 < ADDRESS 2)

All addresses from ADDRESS 1 to ADDRESS 2 (inclusive) qualify as trace samples. The addresses outside the range do not qualify.

Case 2 (ADDRESS 1 > ADDRESS 2)

No addresses between ADDRESS 1 and ADDRESS 2 (exclusive) qualify as trace samples. The addresses outside the range do qualify (ADDRESS 1 and ADDRESS 2 also qualify).

PARAMETER DESCRIPTIONS (CONTINUED):

EMU DATA COMPARE WORD
Processor data must match this value (based on the mask below) for the access to qualify as a trace samples.

EMU DATA COMPARE MASK (ONES ENABLE)
This hexadecimal bit mask allows all or part of the EMU DATA COMPARE WORD to be ignored during comparison. For every bit value of 0 the corresponding data bit is ignored. A value of all 0's causes the entire data value to be ignored; a value of all F's causes the entire data value to be compared. (Refer to Appendix B for details on masking.)

EXT DATA COMPARE BYTE
External probe data must match this value (based on the mask below) for the access to qualify as a trace sample.

EXT DATA COMPARE MASK (ONES ENABLE)
This hexadecimal bit mask allows all or part of the EXT DATA COMPARE BYTE to be ignored during comparison. For every bit value of 0 the corresponding data bit is ignored. A value of all 0's causes the entire probe value to be ignored; a value of all F's causes the entire probe value to be compared. (Refer to Appendix B for details on masking.)

EXAMPLE:

Situation: The following example illustrates how memory writes outside the address range can be traced. The TR (Trace) and TRIX commands are executed to establish the trace conditions for a hypothetical program contained in memory, which uses register R0 as the address register and adds locations FFF0 through FFFE to register R1. The TR (Trace) command is executed to set the trace count to 0 (infinite) and to set the trace mode to TRIX (Trace on Instruction Acquisition Extended). The TRIX command is then executed to set the trace conditions for memory writes from the instruction at address 0004. After the program is executed, the DT (Display Trace) command is executed to set the first sample to 0 (oldest sample in trace memory) and the number of samples to 7FE. The DT display shows the out-of-range memory writes that are traced.

Enter: TR<CR>

```
Display:                                              Enter:

TR
  TRACE COUNT (1 - 7FF, 0=INFINITE)          = 000    <CR>
  ADDRESS MASK (ONES ENABLE)                 = FFFF   <CR>
  TRACE MODE [0=TRM & TRIO, 1=TRIX]          = 0      1<CR>
```

Enter: TRIX<CR>

EXAMPLE (CONTINUED):

```
Display:                                            Enter:

TRIX
  OPCODE QUALIFIERS? [0=NO, 1=YES]          = 0     <CR>
  MEMORY QUALIFIER [ OFF, MA, MR, MW ] = 0          MW<CR>
  I/O QUALIFIER [ OFF, IOA ]                = 0     1<CR>
  TRACE ADDRESS  #1                         = 0000  0004<CR>
  TRACE ADDRESS  #2                         = 0000  0004<CR>
  RANGE INDICATOR [0=NO, 1=YES]             = 0     <CR>
  EXT DATA COMPARE BYTE                     = 00    <CR>
  EXT DATA COMPARE MASK (ONES ENABLE)       = 00    <CR>
```

Enter: DT<CR>

```
Display:                                                      Enter:

DT
  FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000            <CR>
  NUMBER OF SAMPLES                          = 000            7FE<CR>
    INDEX CYCLE QUAL  EXTQUALS      ADDR DATA  RVRS ASSEMBLY

    0000        IAQ   11111111      0004 A050  A    *R0,R1
    0001        MW    11111111      0022 FFFF
    0002        IAQ   11111111      0004 A050  A    *R0,R1
    0003        MW    11111111      0022 FFFE
    0004        IAQ   11111111      0004 A050  A    *R0,R1
    0005        MW    11111111      0022 FFFD
    0006        IAQ   11111111      0004 A050  A    *R0,R1
    0007        MW    11111111      0022 FFFC
    0008        IAQ   11111111      0004 A050  A    *R0,R1
    0009        MW    11111111      0022 FFFB
    000A        IAQ   11111111      0004 A050  A    *R0,R1
    000B        MW    11111111      0022 FFFB
    000C        IAQ   11111111      0004 A050  A    *R0,R1
    000D        MW    11111111      0022 FFFB
    000E        IAQ   11111111      0004 A050  A    *R0,R1
    000F        MW    11111111      0022 FFFB
```

3.57 COMMAND: TRM - Trace, Memory

PURPOSE: Allows memory cycles to be traced.

NOTE: The breakpoint-trace board must be installed to use this command. This command is used in conjunction with the TR command.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
QUALIFIER [OFF,MA, MR, MW, IAQ]	0 = OFF 1 = MA 2 = MR 3 = MW 4 = IAQ	0
TRACE ADDRESS #1		
NUMBER OF EXTENDED ADDRESS BITS = 0	0 - FFFF	0
NUMBER OF EXTENDED ADDRESS BITS = 8	0 - FFFFFF	0
TRACE ADDRESS #2		
NUMBER OF EXTENDED ADDRESS BITS = 0	0 - FFFF	0
NUMBER OF EXTENDED ADDRESS BITS = 8	0 - FFFFFF	0
RANGE INDICATOR [0=NO, 1=YES]	0 = NO 1 = YES	0
EMU DATA COMPARE WORD	0 - FFFF	0
EMU DATA COMPARE MASK (ONES ENABLE)	0 - FFFF	0
EXT DATA COMPARE BYTE		
NUMBER OF EXTENDED ADDRESS BITS = 0	0 - FF	0
NUMBER OF EXTENDED ADDRESS BITS = 8	0	
EXT DATA COMPARE MASK		
NUMBER OF EXTENDED ADDRESS BITS = 0	0 - FF	0
NUMBER OF EXTENDED ADDRESS BITS = 8	0	

PARAMETER DESCRIPTIONS:

QUALIFIER [OFF,MA, MR, MW, IAQ]

This parameter defines the primary qualifying criteria for traces.

OFF - Disqualifies all memory accesses as trace samples.

MA - Allows all memory accesses to qualify trace samples.

PARAMETER DESCRIPTIONS (CONTINUED):

MR - Allows only read operations to qualify trace samples.

MW - Allows only write operations to qualify trace samples.

IAQ - Allows only acquisitions to qualify trace samples.

TRACE ADDRESS #1
Allows a specific memory address to qualify a trace sample.

TRACE ADDRESS #2
Allows a second memory address to qualify a trace sample. To select a single address , set the RANGE INDICATOR to OFF and TRACE ADDRESS #1 to same value as TRACE ADDRESS #2.

RANGE INDICATOR [0=NO, 1=YES]
NO - Each address entered above qualifies as an individual trace sample.

YES - The addresses act as boundaries for a range of addresses as follows:
Case 1 (ADDRESS 1 < ADDRESS 2)
All addresses from ADDRESS 1 to ADDRESS 2 (inclusive) qualify trace samples. The addresses outside the range do not qualify.
Case 2 (ADDRESS 1 > ADDRESS 2)
No addresses between ADDRESS 1 and ADDRESS 2 (inclusive) qualify trace samples. The addresses outside the range do qualify.

EMU DATA COMPARE WORD
Processor data must match this value (based on the mask below) for an access to qualify a trace sample.

EMU DATA COMPARE MASK (ONES ENABLE)
This hexadecimal bit mask allows all or part of the EMU DATA COMPARE WORD to be ignored during comparison. For every bit value of 0 the corresponding data bit is ignored. A value of all 0's causes the entire data value to be ignored; a value of all F's causes the entire data value to be compared. (Refer to Appendix B for details on masking.)

EXT DATA COMPARE BYTE
External probe data must match this value (based on the mask below) for an access to qualify a trace sample.

EXT DATA COMPARE MASK (ONES ENABLE)
This hexadecimal bit mask allows all or part of the EXT DATA COMPARE BYTE to be ignored during comparison. For every bit value of 0, the corresponding data bit is ignored. A value of all 0's causes the entire probe value to be ignored; a value of all F's causes the entire probe value to be compared. (Refer to Appendix B for details on masking.)

COMMANDS AND PARAMETERS THAT AFFECT TRM PARAMETERS

INTERACTING COMMAND	INTERACTING PARAMETER	AFFECTED PARAMETER
INIT	NUMBER OF EXT ADDRESS BITS	TRACE ADDRESS EXT DATA COMPARE BYTE EXT DATA MASK
TR	ADDRESS MASK	TRACE ADDRESS

EXAMPLE:

Situation: The following example illustrates how memory accesses are traced. The program that was traced uses Register R0 as an address register and adds locations FFFC and FFFE to Register R1. The execution of the TR (Trace) command sets the TRACE COUNT to 0 (INFINITE) and enables the ADDRESS MASK. Zero (TRM & TRIO) is retained for TRACE MODE so that the memory cycles can be traced in the TRM command. The DT command is executed to display the trace (the FIRST SAMPLE is set to 0 - oldest sample in trace memory, and the NUMBER OF SAMPLES is set to 7FE).

Enter: TRM<CR>

```
Display:                                              Enter:

TRM
  QUALIFIER [ OFF, MA, MR, MW, IAQ ]       = 0        MA<CR>
  TRACE ADDRESS  #1                        = 0000     <CR>
  TRACE ADDRESS  #2                        = 0000     FFFF<CR>
  RANGE INDICATOR [0=NO, 1=YES]            = 0        YES<CR>
  EMU DATA COMPARE WORD                    = 0000     <CR>
  EMU DATA COMPARE MASK (ONES ENABLE)      = 0000     <CR>
  EXT DATA COMPARE BYTE                    = 00       <CR>
  EXT DATA COMPARE MASK (ONES ENABLE)      = 00       <CR>
```

Enter: TR<CR>

```
Display:                                                  Enter:

TR
  TRACE COUNT (1 - 7FF, 0=INFINITE)            = 000      <CR>
  ADDRESS MASK (ONES ENABLE)                   = FFFF     <CR>
  TRACE MODE [0=TRM & TRIO, 1=TRIX]            = 0        <CR>
```

Enter: DT<CR>

EXAMPLE (CONTINUED):

```
Display:                                                    Enter:

DT
  FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST] = 000    <CR>
  NUMBER OF SAMPLES                          = 000    7FE<CR>
```

```
Display:

DT
  FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST] = 000
  NUMBER OF SAMPLES                          = 000    7FE

    INDEX CYCLE QUAL  EXTQUALS      ADDR DATA  RVRS ASSEMBLY

    0000        IAQ   11111111      0000 0200  LI   R0,>FFFC
    0001        MR    11111111      0002 FFFC
    0002        IAQ   11111111      0004 A050  A    *R0,R1
    0003        MW    11111111      0040 FFFC
    0004        MR    11111111      0040 FFFC
    0005        MR    11111111      FFFC 0016
    0006        MR    11111111      0042 0021
    0007        IAQ   11111111      0006 05C0  INCT R0
    0008        MW    11111111      0042 0037
    0009        MR    11111111      0040 FFFC
    000A        IAQ   11111111      0008 16FD  JNE  >0004
    000B        MW    11111111      0040 FFFE
    000C        IAQ   11111111      0004 A050  A    *R0,R1
    000D        MR    11111111      0040 FFFE
    000E        MR    11111111      FFFE 0017
    000F        MR    11111111      0042 0037
    0010        IAQ   11111111      0006 05C0  INCT R0
    0011        MW    11111111      0042 004E
    0012        MR    11111111      0040 FFFE
    0013        IAQ   11111111      0008 16FD  JNE  >0004
    0014        MW    11111111      0040 0000
    0015        IAQ   11111111      000A 0000  DATA >0000

?
```

3.58 COMMAND: TRUN - Total RUN

PURPOSE: Directs all emulators in the multi-processor chain to begin running. Those not in the global group may delay the start of the RUN.

NO PARAMETERS.

EXAMPLES:

Situation 1: AUTOPOLL feature disabled.

Enter: TRUN<CR>

Display:

```
TRUN

BACKGROUND
??
```

Situation 2: AUTOPOLL feature enabled.

Enter: TRUN<CR>

Display:

```
TRUN

AUTOPOLLING
???
```

3.59 COMMAND: UL - Upload

PURPOSE: Sets parameters for uploading object code from the emulator to the host system.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
START ADDRESS	0 - FFFF	0
END ADDRESS	0 - FFFF	0
FORMAT [0=TI, 1=TICOM, 2=TEK, 3=INTEL, 4=MR]	0 = TI 1 = TICOM 2 = TEK 3 = INTEL 4 = MR	0
DATA (0=WORD, 1=HI BYTE, 2=LOW BYTE)	0=WORD 1=HI BYTE 2=LOW BYTE	0
DESTINATION [0=HOST, 1=PROM, 2=USER]	0=HOST 1=PROM 2=DATA	0
PROTOCOL [0=NONE, 1=TEK, 2=ASR, 3=VAX]	0=NONE 1=TEK 2=ASR 3=VAX	3

PARAMETER DESCRIPTIONS:

START ADDRESS
Specifies the beginning of the block of data to be uploaded.

END ADDRESS
Specifies the end of the block of data to be uploaded.

FORMAT [0= TI, 1= TICOM, 2=TEK, 3=INTEL, 4=MR]
Specifes the format for the object code.
VALUES: 0 = TI Normal ASCII
1 = TI Compressed Format
2 = Tektronix Hexadecimal Format
3 = Intel Intellec 8/MDS Format
4 = Motorola Exorcisor Format

DATA (0=WORD, 1=HI BYTE, 2=LOW BYTE)
Used in conjunction with object FORMATs other than TI or TI-compressed. Allows the user to upload one byte of the data word for split-word PROM applications.

PARAMETER DESCRIPTIONS (CONTINUED):

DESTINATION [0=HOST, 1=PROM, 2=USER]

Allows the user to define the destination of the uploaded data.

VALUES:
- 0 = Data is uploaded through Port D (host port).
- 1 = PROM data is uploaded through Port C into either a PROM programmer or some logging device (if desired).
- 2 = Data is uploaded through Port A. This feature is provided for those using intelligent terminals or personal computers as terminals.

PROTOCOL [0= NONE, 1= TEK, 2= ASR, 3= VAX]

Specifies the handshaking protocols used during data transfers. The selections for this parameter provide standard control-character codes for beginning downloads, ending downloads, beginning uploads, ending uploads, and pass-through characters.

VALUES:
- 0 = No handshake protocol used
  Specific file control-characters are defined using the IHC command. IHC must be done before executing this command.

- 1 = Tektronix handshakes are enabled.
  ASCII "0" - Record transferred without error.
  ASCII "7" - Error detected. Retransmit record.
  Specific control-characters are defined using IHC command. IHC must be done before executing this command.

- 2 = ASR terminal handshake is enabled.
  CTRL-R - Start download
  CTRL-S - End download
  CTRL-Q - Start upload
  CTRL-@ - End upload
  CTRL-P - Pass through next control character

- 3 = VAX or PDP-11 system handshake is enabled.
  CTRL-A - Start upload
  CTRL-W - End upload
  CTRL-V - Start download
  CTRL-Z - End download
  CTRL-P - Pass through character

EXAMPLE:

Situation: The following example uploads data from addresses 01A through 110 to host system in TI format. To perform this upload, enter HOST mode and send a 'START UPLOAD' character from the host or keyboard.

Enter: UL<CR>

```
Display:                                                  Enter:

UL
  START ADDRESS                                  = 0000   001A<CR>
  END ADDRESS                                    = 0000   0110<CR>
  FORMAT [0=TI, 1=TICOM, 2=TEK, 3=INTEL, 4=MR]   = 0      <CR>
  DATA (0=WORD, 1=HI BYTE, 2=LOW BYTE)           = 0      <CR>
  DESTINATION [0=HOST, 1=PROM, 2=USER]           = 0      <CR>
  PROTOCOL [0=NONE, 1=TEK, 2=ASR, 3=VAX]         = 3      <CR>
?
```

3.60 COMMAND: XA - Execute Assembler

PURPOSE: To enter and assemble a program from the keyboard without a host computer.

NOTE: Refer to Section 4 for a detailed explanation of the assembler.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
START ADDRESS	0 - FFFF	0

PARAMETER DESCRIPTIONS:

START ADDRESS
This specifies the beginning address for the program to be assembled.

EXAMPLE:

Situation: The following example enters an assembly language program. The INIT command is executed prior to the example in order to allocate memory (Emulator RAM) in which to enter the sample program. The XRA (Execute Reverse Assembler) command is executed after the example to list the instructions which were entered.

NOTE

The assembler accepts only two-character labels. If entry of more than two characters is attempted, the assembler ignores all but the last two. To enter an instruction without first entering a label, a space (<SP>) is entered; this jumps the cursor to the INST (instruction) entry field.

Enter: INIT<CR>

EXAMPLE (CONTINUED):

```
Display:                                                    Enter:

INIT
    EMU: CLOCK SOURCE [0=INTERNAL, 1=TARGET]          = 0   0<CR>
         EMULATOR RAM? [0=NO, 1=YES]                  = 0   1<CR>
  EXMEM: EXPANSION PAGE [0="A", 1="B"]                = 0   0<CR>
     BP: NUMBER OF EXTENDED ADDRESS BITS (0 - 8)      = 0   <CR>
```

NOTE

* Cursor movement is confined to the space bar while using the assembler.
* For emphasis, the spaces between fields have been expanded in the following display.

Enter: XA<CR>

```
Display:                  Enter:

XA
  START ADDRESS = 0000    A<CR>
    CONTROL-E EXITS

    ADDR      DATA      LB       INST
    000A      04C4      IN       CLR R4<CR>
    000C      0201      <SP>     LI R1,10<CR>
    000E      000A
    0010      0202      <SP>     LI R2,>1234<CR>
    0012      1234
    0014      0203      <SP>     LI R3,>10<CR>
    0016      0010
    0018      38C2      LP       MPY R2,R3<CR>
    001A      C803      <SP>     MOV R3,@>100<CR>
    001C      0100
    001E      C804      <SP>     MOV R4,@>102<CR>
    0020      0102
    0022      C0C1      <SP>     MOV R1,R3<CR>
    0024      0601      <SP>     DEC R1<CR>
    0026      16F8      <SP>     JNE LP<CR>
    0028      10F0      <SP>     JMP IN<CR>
    002A                <SP>     END<CR> 0000
```

The user entered <SP>END<CR> to exit the XA command. The value displayed in the END instruction indicates the number of errors in the program entry. An alternate method of exiting the XA command is to enter <CTRL>E, in which case the number of errors is displayed in the DATA column of the last line in the display. Backspace is not allowed; <SPACE> END or <CTRL>E are the only ways to quit the ASSEMBLER.

EXAMPLE (CONTINUED):

Enter: XRA<CR>

```
Display:                               Enter:

XRA
  START ADDRESS           = 0000      A<CR>
  NUMBER OF INSTRUCTIONS  = 0000      B<CR>
    000A 04C4  CLR  R4
    000C 0201  LI   R1,>000A
    000E 0202  LI   R2,>1234
    0014 0203  LI   R3,>0010
    0018 38C2  MPY  R2,R3
    001A C803  MOV  R3,@>0100
    001E C804  MOV  R4,@>0102
    0022 C0C1  MOV  R1,R3
    0024 0601  DEC  R1
    0026 16F8  JNE  >0018
    0028 10F0  JMP  >000A
```

3.61 COMMAND: XRA - Execute Reverse Assembler

PURPOSE: Displays a section of memory in assembly language. The monitor effectively recreates a source listing from the object code stored in the desired memory block by printing the memory address, memory data, the mnemonic code for the instruction, and operands.

PARAMETER	PARAMETER VALUES	POWER-UP VALUE
START ADDRESS	0 - FFFF	0
NUMBER OF INSTRUCTIONS	0 - FFFF	0

PARAMETER DESCRIPTIONS:

START ADDRESS
Specifies the beginning address of the program to be reverse assembled.

NUMBER OF INSTRUCTIONS
Specifies the number of the program's instructions to be displayed.

EXAMPLES:

Situation 1: The following example reverse assembles 11 instructions beginning at address 000A.

Enter: XRA<CR>

Display: Enter:

```
XRA
  START ADDRESS              = 0000 A<CR>
  NUMBER OF INSTRUCTIONS     = 0000 B<CR>
    000A 04C4  CLR  R4
    000C 0201  LI   R1,>000A
    0010 0202  LI   R2,>1234
    0014 0203  LI   R3,>0010
    0018 38C2  MPY  R2,R3
    001A C803  MOV  R3,@>0100
    001E C804  MOV  R4,@>0102
    0022 C0C1  MOV  R1,R3
    0024 0601  DEC  R1
    0026 16F8  JNE  >0018
    0028 10F0  JMP  >000A
?
```

EXAMPLES (CONTINUED):

Situation 2: Care should be taken to ensure that an opcode resides at the specified start address. When the reverse assembler expects an opcode at a memory location, and the value at that location has no valid opcode mnemonic, the reverse assembler will interpret the value as a "data" statement. On the other hand, if the user specifies a start address in the "middle" of a program (for instance on a data statement), and that address contains data that has a valid opcode mnemonic, the resulting display will be very confusing. Consider the sample displays below (The XA command is shown to illustrate the program that was reverse assembled). In the first display, the user specifies a start address on a valid opcode boundary. If the user randomly picks address 0012 as the start address, an entirely different program is reverse assembled, as shown in the second display.

```
XA
 START ADDRESS = 000A  10
   CONTROL-E  EXITS

   ADDR DATA LB INST
   0010 C803    MOV R3,@>200
   0012 0200
   0014 C820    MOV @>380,@>100
   0016 0380
   0018 0100
   001A         END   0000
```

Enter: XRA<CR>

Display: Enter:

```
XRA
 START ADDRESS             = 000A  10<CR>
 NUMBER OF INSTRUCTIONS    = 0010  2<CR>
   0010 C803  MOV  R3,@>0200
   0014 C820  MOV  @>0380,@>0100
?
```

EXAMPLES (CONTINUED):

In the preceding execution of the command, the instructions are correctly reverse assembled and agree with the program entered in the XA command, above. Notice that in the following execution, the reverse assembly starts at location 0012 instead of the correct opcode boundary at location 0010. The reverse assembly produces an entirely different program than was entered above. Location 0012 contains a data value which is mistaken for an opcode, since the START ADDRESS was not a valid opcode boundary.

Enter: XRA<CR>

```
Display:                           Enter:

XRA
  START ADDRESS            = 0010  12<CR>
  NUMBER OF INSTRUCTIONS   = 0002  3<CR>
    0012 0200  LI    R0,>C820
    0016 0380  RTWP
    0018 0100  DATA >0100
```

3.61 VARIABLES

PURPOSE: Variables are specialized instructions that allow the user to access registers that may not otherwise be easily accessible. This simplifies the process of displaying or changing the contents of registers.

VARIABLE NAMES

Variable	Register	Maximum Value
WP	Workspace Pointer	FFFF
PC	Program Counter	FFFF
ST	Status Register	FFFF

VARIABLES FOR STATUS REGISTER BITS

Variable Name	Register Name	Maximum Value	Bit Position(s) in Status Reg.
LGT	Logical Greater Than	1	0
AGT	Arithmetic Greater Than	1	1
EQ	Equal	1	2
C	Carry Out	1	3
OV	Overflow Flag	1	4
OP	Odd Parity	1	5
INTM	Interrupt Mask	F	12 - 15

EXAMPLES:

Example 1. The display function of a variable.

Enter: PC<CR>

Display:

```
PC
    PC=01AD
```

The monitor displays the current contents of the program counter.

Variables may also be used to set the contents of specific registers. To set the contents of the program counter, enter the PC variable followed by an equal sign <=>, and then the new value. When the <CR> is entered the program counter is set to the new value.

Example 2. The set function of a variable.

Enter: PC=22<CR>

Display:

```
PC=22
```

The program counter is now set to a value of 0022.

SECTION 4

MEMORY MAPPING AND ASSEMBLER

4.1 INTRODUCTION

This section provides a detailed discussion of the TMS9995 Emulator's design considerations and memory mapping feature and provides instructions on using the line-by-line symbolic assembler.

4.2 DESIGN CONSIDERATIONS

When you are designing memory controllers, you must strongly consider the following two items:

4.2.1 The presence of the WE- and DBIN- signals validate memory and CRU (Communication Register Unit) cycles. When the emulator is halted, the MEM- signal will be active but does not necessarily indicate a memory cycle. Only the falling edge of the WE- or DBIN- signals indicate the start of a valid memory cycle. Conversely, only the rising edge of WE- or DBIN- indicate the end of a memory cycle.

4.2.2 When emulator memory or expansion memory are mapped in, any target memory accesses to the same addresses as mapped emulator or expansion memory are ignored. This is accomplished by ignoring TARGET READY on accesses to the addresses corresponding to mapped ERAM (emulator RAM) or DRAM (expansion memory RAM).

4.3 MEMORY MAPPING

The TMS9995 Emulator allows the flexibility to substitute blocks of emulator memory for target system memory to facilitate development of target system software before prototype hardware exists. It also provides convenient development/debugging of system firmware. The emulator contains two 64K-byte pages of dynamic RAM (expansion memory) that can be mapped in place of target memory with 1K-byte resolution and can be write protected (ROM) if desired. In addition, it allows 1K-byte of static RAM (emulator RAM) to be substituted for the first 1K-byte block of memory. All memory accesses go to the target system, regardless of any mapping scheme. Thus, any data manipulations on mapped memory will also affect target memory; target read data is simply ignored.

4.3.1 Mapping Emulator RAM and Expansion Memory

When the TMS9995 Emulator is first powered-up, no emulation memory is substituted. The user has access to 259 bytes of on-chip RAM (addresses >F000 through >F0FA and >FFFA through >FFFF) as well as all of the target system memory. This can be modified by turning on emulator RAM and/or mapping in expansion memory as desired. The TMS9995 memory mapping scheme and the methods for manipulating memory are discussed below.

4.3.1.1 Emulator RAM

The emulator is supplied with 1K-byte of static RAM (emulator RAM) which can be selected in the INIT (Initialize Emulator) command. Two possible reasons for selecting emulator RAM to use in place of target or expansion memory are:

1) This memory has a one-state access and can be used to maximize the performance of the TMS9995 Emulator.

2) It is static RAM, rather than dynamic RAM, and will not lose data in the event that the system clock is lost (for instance, if the system was running on the target system clock and the target system was powered down, the static emulator RAM would not lose the data contained in memory).

The emulator RAM resides in address >0000 through >03FF and takes precedence over any other memory that may reside in those same locations. This means that any accesses to the first >400 locations in memory will access emulator RAM (when it is turned ON). Figure 4-X shows the relationship of emulator RAM to target memory and expansion memory.

4.3.1.2 Expansion Memory

Two 64K-byte pages of dynamic RAM expansion memory are provided with the emulator. Figure 4-1 shows the relationship of expansion memory to target memory and emulator RAM. Like emulator RAM, when expansion memory is mapped it takes precedence over target memory (target memory reads are ignored but data manipulations still occur). Emulator RAM takes precedence over expansion memory if emulator RAM is selected. If an attempt is made to map the first 1K-byte block of expansion memory after selecting emulator RAM a warning will occur; expansion memory will be mapped but the user will be warned that it overlaps emulator RAM.

Expansion memory can be mapped as ROM or RAM, or can be turned off, by making the appropriate selection in the MAP command. Another function of the MAP command allows target memory to be copied into the corresponding expansion memory locations. Each of these functions is described in paragraphs 4.3.1.2.2 through 4.3.1.2.5.

```
0000-- ----------        0000-- ----------     LEGEND:
       |........|               |\\\\\\\\|
       |........|        03FF-- |\\\\\\\\|      ....
       |........|               |........|      .... = TARGET
       |........|               |........|      ....   MEMORY
       |........|               |........|
       |........|               |........|
       |........|               |........|      ////
       |........|               |........|      //// = ON-CHIP
       |........|               |........|      ////   MEMORY
7FFF-- |........|        7FFF-- |........|
       |........|               |........|
       |........|               |........|      \\\\
       |........|               |........|      \\\\ = EMULATOR
       |........|               |........|      \\\\   MEMORY
F000-- |////////|        F000-- |////////|
       |////////|               |////////|
F0FA-- |////////|        F0FA-- |////////|
       |........|               |........|
       |........|               |........|
FFFA-- |////////|        FFFA-- |////////|
       |////////|               |////////|
FFFF-- ----------        FFFF-- ----------

   ON-CHIP MEMORY               EMULATOR RAM
                                     +
                               ON-CHIP MEMORY
```

```
0000-- ----------        0000-- ----------     LEGEND:
       |********|               |\\\\\\\\|
       |********|        03FF-- |\\\\\\\\|
       |********|               |********|      ....
       |********|               |********|      .... = TARGET
       |********|               |********|      ....   MEMORY
       |********|               |********|
       |********|               |********|
       |********|               |********|      ////
       |********|               |********|      //// = ON-CHIP
7FFF-- |********|        7FFF-- |********|      ////   MEMORY
       |........|               |********|
       |........|               |********|
       |........|               |********|      \\\\
F000-- |////////|        F000-- |////////|      \\\\ = EMULATOR
       |////////|               |////////|      \\\\   MEMORY
F0FA-- |////////|        F0FA-- |////////|
       |........|               |********|
       |........|               |********|      ****
FFFA-- |////////|        FFFA-- |////////|      **** = EXPANSION
       |////////|               |////////|      ****   MEMORY
FFFF-- ----------        FFFF-- ----------

    >20K-BYTES OF          EXMEM PAGE ("A" or "B")
EXMEM PAGE ("A" or "B")              +
          +                   ON-CHIP MEMORY
    ON-CHIP MEMORY                   +
                               EMULATOR RAM
```

FIGURE 4-1. MEMORY MAPPING

4.3.1.2.1 Expansion Memory Blocks

Before understanding the details of the types of memory defined in the MAP command, the address blocking method must be explained. The "BASE ADDRESS" and "NUMBER OF 1KB BLOCKS" parameters define the starting address and the amount of memory to be mapped. The base address must fall on a 1K-byte boundary (e.g., >0000, >0400, >0800, >0C00, etc.). If an address is specified that is not a boundary, the base address will be rounded to the next lower boundary. For example, if the user specifies >81C4 as the base address, it will be rounded to >8000 when the MAP command is executed.

The "NUMBER OF 1KB BLOCKS" parameter defines the amount of memory being mapped. The value assigned to this parameter must be hexadecimal. The maximum value is >40 (64, decimal), which allows the user to map all of one expansion memory page at once. Three things must be remembered about selecting the blocks being mapped:

1) If a base address of less than >400 is specified, and emulator RAM is selected, an error (WARNING - Overlap with EMU RAM) will occur.

2) Block boundaries occur on 1K-byte boundaries. If it is necessary to map memory on other than even boundaries (addresses >86A4 through >896B, for instance) then the blocks that span the range of addresses must be mapped (in the case of this example, the blocks that include addresses >8400 through >8BFF must be mapped).

3) If the number of blocks selected exceed the 64K-byte address space, the blocks that extend past the address range will not be mapped (wrap-around does not occur). For instance, if target memory is being copied to expansion memory and >20 (32, decimal) blocks are selected starting with base address >C400, only >F (15, decimal) blocks will be copied.

4.3.1.2.2 Unmapping Expansion Memory

The UNMAP response to the "[0=UNMAP, 1=ROM, 2=RAM, 3=COPY]" parameter of the MAP command removes the block of memory specified in the "BASE ADDRESS" and "NUMBER OF 1KB BLOCKS" parameters from the emulator's memory map and allows read and write accesses to be performed on target memory residing in the corresponding addresses. The contents of the block of memory remain intact after unmapping and can be recalled by remapping the block. Note that emulator RAM is not affected by the UNMAP response, to remove emulator RAM from the memory map, the INIT command must be executed with a NO response to the "EMULATOR RAM?" parameter.

4.3.1.2.3 Expansion Memory ROM

The expansion memory is made up of dynamic RAM (DRAM) but can be made to function as ROM if desired. This is accomplished by selecting the ROM response in the "[0=UNMAP, 1=ROM, 2=RAM, 3=COPY]" parameter of the MAP command. When expansion memory (or a block of expansion memory) is defined as ROM, the locations so specified can still be written to in the control mode of operation, but in the run mode they behave exactly as ROM. This allows the user to operate those portions of memory as ROM but lets him manually change the values of those locations as required.

4.3.1.2.4 Expansion memory RAM

When expansion memory is defined as RAM in the MAP command, it functions as typical dynamic RAM. The refresh cycles can be made transparent by selecting two extra wait states in the "NUMBER OF EXTRA WAIT STATES" parameter of the MAP command.

4.3.1.2.5 Copying Target Memory into Expansion Memory

The COPY selection of the "[0=UNMAP, 1=ROM, 2=RAM, 3=COPY]" parameter of the MAP command allows the user to copy the specified block of target memory into the corresponding block of expansion memory. After the copy is performed, the expansion memory block is set as ROM. To define the block as RAM, the MAP command must be executed again.

4.4 TMS9995 SYMBOLIC ASSEMBLER

The XDS TMS9995 Emulator's assembler is a symbolic, absolute, line-by-line, onboard assembler. You should refer to the TMS9995/99000 Assembly Language Programmer's Guide to become familiar with the TMS9995 assembly language.

The TMS9995 Emulator's assembler supports the following basic features:

* Symbolic addressing
* Forward referencing
* Simple expressions using the + and - operators with the following operands:
  - Hexadecimal data
  - Decimal data
  - Text
  - Symbols
* Responds to most common directives

4.4.1 Entering Source Code

An assembly language source program consists of statements that may contain assembler directives, machine instructions, operands, or comments. These source statements are partitioned into one to four ordered fields separated by one or more blanks. These fields (label, command mnemonic, operand, and comment) are discussed in the following paragraphs. Source statements that begin with an asterisk (*) in the first character position are comment statements and do not affect the assembly.

The syntax for source statements other than comments is:

[<label>] <SP> <mnemonic> <SP> [[<operand>] . . . [,operand]] [<comment>]

* The optional label is defined by the user.
* One or more blanks separate the label field from the command mnemonic. If no label is used, a <SP> must precede the mnemonic.
* The generic term mnemonic includes operation codes or assembler directives.
* One blank separates the mnemonic from the operand field.
* Multiple operands are separated by <u>one</u> blank or a comma.
* One or more blanks separate the operand(s) from the comment field. Comments are ignored by the assembler.

Source code may be entered into the assembler only by direct entry from the keyboard using the XA (Execute Assembler) XDS monitor command.

The XA command asks you for the program's starting location. When you are using the assembler, remember that there is no cursor control or backspace. If you make a mistake while entering code, you must press <ESC> and start again to enter code correctly at the same address. If you make a mistake, such as a syntax error, during entry and do not catch it, the assembler will sense the error and will display the same address again for reentry.

When source statements are entered using the XA command, the entry field is preceded by a display of the address and a blank field, which is filled after the source statement is entered. The entry line appears as follows before the data is entered:

```
F000      []
```

It appears as follows after the data is entered:

```
F000 C803 MOV R3,@>100
F002 0100
```

This source statement requires four bytes for implementation; therefore, two addresses are displayed (F000 - contains opcode for the instruction, F002 - contains the address where the contents of R3 are to be moved).

To exit the assembler, enter either the END directive or a CTRL-E. Entering <ESC> begins a new line of input at the same address.

4.4.2 The Source Statement Line

The source statement line is made up of the label field, command (mnemonic) field, operand field, and comment field. These data fields are described in the following paragraphs.

4.4.2.1 Label Field

The label field begins in the first character position of the source statement and extends to the first blank. The TMS9995 Emulator's assembler uses two-character labels. Symbols of any length may be used but only the last two characters are significant. The first character of the label must be alphabetic; the other character may be alphanumeric. A label is optional for machine instructions and for many assembler directives.

Labels cannot be redefined. The use of undefined symbols (forward referencing) is allowed for most operands. These references are backchained in target memory and resolved when the symbol is defined. Any operand can be forward referenced. However, only the first operand of the DATA directive can be forward referenced.

Any jump displacement can be forward referenced. Consecutive references must occur within range of 127 bytes.

NOTE

> Assemble only into RAM memory space, because of the backchaining operation. Assembly into ROM causes the assembler to attempt to resolve invalid chains, with possible disastrous results. Exiting the assembler with unresolved chains in memory may produce unexpected results.

4.4.2.2 Command Field

The command field begins one space after the label field (i.e., [label]<SP>[command]). If no label is used, one blank space must precede the command field. The command is terminated by one blank space.

4.4.2.3 Operand Field

The operand field begins after the blank that follows the command field (i.e., [<label>] <SP> [<command>] <SP> [<operand> <operand> <operand>]). The operand field can contain one or more constants or expressions separated by a comma <u>or</u> a space. The operand field is terminated by one or more blank spaces.

4.4.2.4 Comment Field

The comment field begins after the blank(s) that terminate(s) the operand field, or the command field of there are no operands (i.e., [<label>] <SP> [<command>] <SP> [<operand> <operand> <operand>] <SP> [<comment>]). A comment may also begin in the first column of the source statement line (normally the start of the label field) if the comment is preceded by an asterisk (*). The comment field may contain any valid ASCII character. Comments have no effect on assembly.

4.4.3 Constants

The assembler recognizes four types of constants, listed below, which are maintain internally as 16-bit quantities.

* Decimal integers
* Hexadecimal integers
* Text strings
* Assembly-time

Each of these types of constants are discussed in the following paragraphs.

4.4.3.1 Decimal Integer Constants

A decimal integer constant is a string of decimal digits in the range of -32,768 to +32,767. Positive decimal integer constants within the range of 32,768 to 65,535 are considered negative when interpreted as twos complement values.

The following are valid decimal integer constants:

```
  2000 - Constant equal to 2000 (>7D0)
-32768 - Constant equal to -32768 (>8000)
    25 - Constant equal to 25 (>19)
```

4.4.3.2 Hexadecimal Integer Constants

A hexadecimal integer constant is a string of up to four hexadecimal digits preceded by a 'greater-than' sign (>).

The following are valid hexadecimal integer constants:

```
>90   - Constant equal to >90 (144, decimal)
>F    - Constant equal to >F (15, decimal)
>37AC - Constant equal to >37AC (14252, decimal)
```

4.4.3.3 Text String Constants

Text string constants must begin and end with a single-quote ('). Text strings may be any length. The characters enclosed in the single-quotes are represented internally as eight-bit ASCII codes. A text string constant consisting of only two single-quotes (no character) is valid and assumes the value of >00. Double-quotes are not supported by the assembler. Control characters are valid text.

The following are valid string constants:

```
'AB' - Represented internally as >41 >42
'C'  - Represented internally as >43
'N'  - Represented internally as >4E
```

4.4.3.4 Assembly-Time Constants

An assembly-time constant is a symbol assigned a value by an EQU assembler directive. The value is determined at assembly time and is considered absolute or relocatable according to relocatability of the expression (not according to the relocatability of the location counter value). Absolute value symbols may be assigned values with expressions using any of the above constant types.

4.4.4 Symbols

Symbols are used in the label field and the operand field. Symbols are defined by the following characteristics:

* An alphanumeric string of characters of undetermined length
* The first character must be alphabetic
* Blanks are NOT allowed
* Symbols must be unique in the last two characters

Symbols used in the label field of the source statement become symbolic addresses. They are associated with specific locations in the program and must not be used in the label fields of other statements.

Symbols used in the operand field must be defined in the assembly, usually by appearing in the label field of a statement.

The following are examples of valid symbols:

B1	Assigned the value of the location of the label field of the statement in which it appears
AD	Assigned the value of the location of the label field of the statement in which it appears
OPERATION	ON is assigned the value of the location of the label field of the statement in which it appears

4.4.5 Expressions

Expressions can be used for any operand as follows:

* Defined symbols can be used anywhere in the expression
* Undefined symbols occur as single operands (where allowed) with no operators
* Operands in an expression are separated by a + or - operator
* Operands in an expression may be decimal integer or hexadecimal integer constants, or text strings:
  - Integer constants are assumed to decimal unless preceded by >
  - May be preceded by a minus (-)
  - Any number of digits may be entered, but only 16-bit results are retained with a warning of overflow
  - Binary constants are not supported at this time
* Text strings may be used in an expression
  - Text strings may be any length but only the last two characters are evaluated
  - Control characters may be included.
* The $ may be used to refer to the current value of the location counter
* Other than $ and R, there are no reserved or predefined symbols

* Any type of operand may be used in any order

* Expression terminators are:

  - <CR>
  - Blank
  - Comma

The following are valid expressions:

A+B	The symbol A plus the symbol B
A+4	The symbol A plus the decimal integer constant 4
$+>A4	The current location counter value plus the hexadecimal integer >A4

4.3.6 TMS9995 Emulator Assembler Directives

The following assembler directives are accepted by the TMS9995 Emulator:

* Program counter control

  - AORG Absolute Origin. Operand required; must be a well-defined expression (all symbols previously defined). Changes the location counter to the operand's value. Label is assigned the old value of the location counter.

  - BSS Block Starting with Symbol. Operand required; value of operand is added to location counter. Label is assigned the address of the start of the block.

* Data directives

  - DATA Initialize Word. Operands are expressions. Results of evaluated expressions are stored as 16-bit values. Multiple operands are separated by commas.

  - TEXT Initialize Text. Operand begins and ends with a single-quote. Character string is stored in memory in ASCII format. Control characters may be included. (A double-quote is stored as a single-quote when located in the character string.)

* Linkage directives

  - EQU Define assembly-time constant. Defines a label. Operand may be a R<n> or an expression. The operand value is assigned to the label.

* Miscellaneous

  - END Ends the assembly.

4.3.7 Examples of Assembler

The following example shows the function of the TMS9995 Assembler. It includes assembler directives and machine instructions, and illustrates the results of resolved and unresolved forward referencing. The program to be entered includes several comments, which appear two ways. The most common method of entering comments uses an asterisk (*) in the first column of the label field. An alternate method involves entering a space (<SP>) following the operand field or command field, if there are no operands. Neither type of comment is retained in memory after assembly; they will be present only on an assembly session log.

Enter: XA<CR>

Display:
```
XA
  START ADDRESS = 0000 []
```

Enter: F000<CR>

Comment: Location F000 will be the starting address for the program.

Display:
```
XA
  START ADDRESS = 0000  F000
    CONTROL-E  EXITS

    ADDR DATA LB INST
    F000      []
```

Enter: CT EQU 0 <SP> LOOP COUNTER WILL BE REGISTER R0 <CR>

Display:
```
XA
  START ADDRESS = 0000  F000
    CONTROL-E  EXITS

    ADDR DATA LB INST
    F000      CT EQU 0 LOOP COUNTER WILL BE REGISTER R0
    F000      []
```

Comment: Notice that the address counter did not increment when you entered the statement. Notice also, that a comment is included in this statement, set off from the operand by one space.

Enter: PT EQU 1 <SP> BLOCK POINTER WILL BE REGISTER R1 <CR>

Display:
```
XA
 START ADDRESS = 0000  F000
   CONTROL-E  EXITS

   ADDR DATA LB INST
   F000         CT EQU 0 LOOP COUNTER WILL BE REGISTER R0
   F000         PT EQU 1 BLOCK POINTER WILL BE REGISTER R1
   F000         []
```

Enter: * INITIALIZE <CR>

Display:
```
XA
 START ADDRESS = 0000  F000
   CONTROL-E  EXITS

   ADDR DATA LB INST
   F000         CT EQU 0 LOOP COUNTER WILL BE REGISTER R0
   F000         PT EQU 1 BLOCK POINTER WILL BE REGISTER R1
   F000         * INITIALIZE
   F000         []
```

Enter: <SP> MOV @BS,CT <SP> INITIALIZE LOOP COUNTER <CR>

Comment: This is the first machine instruction to be entered, it initializes the loop counter (CT) and includes an unresolved forward reference to the location "@BS", which is the address containing the value for "Block Size" (number of data words).

Display:
```
XA
 START ADDRESS = 0000  F000
   CONTROL-E  EXITS

   ADDR DATA LB INST
   F000         CT EQU 0 LOOP COUNTER WILL BE REGISTER R0
   F000         PT EQU 1 BLOCK POINTER WILL BE REGISTER R1
   F000         * INITIALIZE
   F000 C020       MOV @BS,CT INITIALIZE LOOP COUNTER
   F002R0000
   F004         []
```

Comment: The value C020 is the opcode for the MOV instruction and is added by the assembler in the DATA field at address F000. The address counter incremented after the entry of this instruction, to address F002. The statement line for address F002 contains the value 0000 in the data field, preceded by an R, this denotes that an unresolved forward reference has been made and will be used by the assembler for backchaining when the reference is resolved.

Enter: <SP> MOV @BP,PT <SP> INITIALIZE BLOCK POINTER <CR>

Comment: Another machine instruction is entered, this one initializes the Block Pointer. Again, the opcode for the instruction is assembled in the DATA field, the address counter is incremented, and an unresolved forward reference is made (this one at address F006).

Display:
```
XA
  START ADDRESS = 0000  F000
    CONTROL-E  EXITS

    ADDR DATA LB INST
    F000      CT EQU 0 LOOP COUNTER WILL BE REGISTER R0
    F000      PT EQU 1 BLOCK POINTER WILL BE REGISTER R1
    F000      * INITIALIZE
    F000 C020    MOV @BS,CT INITIALIZE LOOP COUNTER
    F002R0000
    F004 C060    MOV @BP,PT INITIALIZE BLOCK POINTER
    F006R0000
    F008      []
```

Enter: <SP> CLR @SUM <CR>

Comment: This machine instruction makes another unresolved reference to clear the address (identified by the label SUM) where the sum of the data contained in the data block will be placed after the add routine is performed. When this statement is assembled, only the last two characters of the symbol SUM will be assembled (i.e., the symbol will be stored as "UM"). This occurs because the TMS9995 assembler accepts only the last two characters of labels and symbols; any preceding characters are ignored. Be careful not to define another symbol ending in the same two characters (UM), or an error will occur.

Display: XA

```
  START ADDRESS = 0000  F000
    CONTROL-E  EXITS

    ADDR DATA LB INST
    F000      CT EQU 0 LOOP COUNTER WILL BE REGISTER R0
    F000      PT EQU 1 BLOCK POINTER WILL BE REGISTER R1
    F000      * INITIALIZE
    F000 C020    MOV @BS,CT INITIALIZE LOOP COUNTER
    F002R0000
    F004 C060    MOV @BP,PT INITIALIZE BLOCK POINTER
    F006R0000
    F008 04E0    CLR @SUM
    F00AR0000
    F00C      []
```

Enter: * MAIN LOOP <CR>

Display: XA

```
  START ADDRESS = 0000  F000
    CONTROL-E  EXITS

    ADDR DATA LB INST
    F000      CT EQU 0 LOOP COUNTER WILL BE REGISTER R0
    F000      PT EQU 1 BLOCK POINTER WILL BE REGISTER R1
    F000      * INITIALIZE
    F000 C020    MOV @BS,CT INITIALIZE LOOP COUNTER
    F002R0000
    F004 C060    MOV @BP,PT INITIALIZE BLOCK POINTER
    F006R0000
    F008 04E0    CLR @SUM
    F00AR0000
    F00C      * MAIN LOOP
    F00C      []
```

Enter: LP <SP> BL @ADD <CR>

Comment: The instruction at address F00C is the first instruction in the main loop, it performs a branch-and-link operation with an unresolved reference to the symbolic address ADD (the start of the subroutine that adds the data in the data block). Note that the symbol ADD is stored in memory as "DD".

Display:
```
XA
  START ADDRESS = 0000  F000
    CONTROL-E  EXITS

    ADDR DATA LB INST
    F000      CT EQU 0 LOOP COUNTER WILL BE REGISTER R0
    F000      PT EQU 1 BLOCK POINTER WILL BE REGISTER R1
    F000      * INITIALIZE
    F000 C020    MOV @BS,CT INITIALIZE LOOP COUNTER
    F002R0000
    F004 C060    MOV @BP,PT INITIALIZE BLOCK POINTER
    F006R0000
    F008 04E0    CLR @SUM
    F00AR0000
    F00C      * MAIN LOOP
    F00C 06A0 LP BL @ADD
    F00ER0000
    F010      []
```

Enter: <SP> DEC CT <SP> DECREMENT LOOP COUNTER <CR>

Display:
```
XA
  START ADDRESS = 0000  F000
    CONTROL-E  EXITS

    ADDR DATA LB INST
    F000      CT EQU 0 LOOP COUNTER WILL BE REGISTER R0
    F000      PT EQU 1 BLOCK POINTER WILL BE REGISTER R1
    F000      * INITIALIZE
    F000 C020    MOV @BS,CT INITIALIZE LOOP COUNTER
    F002R0000
    F004 C060    MOV @BP,PT INITIALIZE BLOCK POINTER
    F006R0000
    F008 04E0    CLR @SUM
    F00AR0000
    F00C      * MAIN LOOP
    F00C 06A0 LP BL @ADD
    F00ER0000
    F010 0600    DEC CT DECREMENT LOOP COUNTER
    F012      []
```

Enter: <SP> JNE LP ADD ANOTHER WORD <CR>

Comment: This instruction compares the value of the loop counter (CT) to zero, if the value is not equal to zero it jumps to the address with the label LP (symbolic address LP).

Display: XA

```
  START ADDRESS = 0000  F000
    CONTROL-E  EXITS

    ADDR DATA LB INST
    F000      CT EQU 0 LOOP COUNTER WILL BE REGISTER R0
    F000      PT EQU 1 BLOCK POINTER WILL BE REGISTER R1
    F000      * INITIALIZE
    F000 C020    MOV @BS,CT INITIALIZE LOOP COUNTER
    F002R0000
    F004 C060    MOV @BP,PT INITIALIZE BLOCK POINTER
    F006R0000
    F008 04E0    CLR @SUM
    F00AR0000
    F00C      * MAIN LOOP
    F00C 06A0 LP BL @ADD
    F00ER0000
    F010 0600    DEC CT DECREMENT LOOP COUNTER
    F012 16FC    JNE LP ADD ANOTHER WORD
    F014      []
```

Enter: <SP> DATA 0 <CR>

Comment: This statement is entered to halt the emulator if it advances past the jump-to-the-symbolic-address-LP statement during execution. (The result of such an action would be to halt the emulator and give a Z-MID - zero opcode - halt status display.)

Display: XA

```
  START ADDRESS = 0000  F000
    CONTROL-E  EXITS

    ADDR DATA LB INST
    F000      CT EQU 0 LOOP COUNTER WILL BE REGISTER R0
    F000      PT EQU 1 BLOCK POINTER WILL BE REGISTER R1
    F000      * INITIALIZE
    F000 C020    MOV @BS,CT INITIALIZE LOOP COUNTER
    F002R0000
    F004 C060    MOV @BP,PT INITIALIZE BLOCK POINTER
    F006R0000
    F008 04E0    CLR @SUM
    F00AR0000
    F00C      *MAIN LOOP
    F00C 06A0 LP BL @ADD
    F00ER0000
    F010 0600    DEC CT DECREMENT LOOP COUNTER
    F012 16FC    JNE LP ADD ANOTHER WORD
    F014 0000    DATA 0
    F016      []
```

Enter:
```
* <CR>
* THIS IS THE ADD SUBROUTINE <CR>
ADD EQU $ <CR>
```

Comment: This statement defines the unresolved forward reference made in location >F00C. Note that the reference is backchained to address >F00E, the location in which the unresolved forward reference appears.

Display:
```
 START ADDRESS = 0000  F000
   CONTROL-E  EXITS

   ADDR DATA LB INST
   F000      CT EQU 0 LOOP COUNTER WILL BE REGISTER R0
   F000      PT EQU 1 BLOCK POINTER WILL BE REGISTER R1
   F000      * INITIALIZE
   F000 C020    MOV @BS,CT INITIALIZE LOOP COUNTER
   F002R0000
   F004 C060    MOV @BP,PT INITIALIZE BLOCK POINTER
   F006R0000
   F008 04E0    CLR @SUM
   F00AR0000
   F00C      *MAIN LOOP
   F00C 06A0 LP BL @ADD
   F00ER0000
   F010 0600    DEC CT DECREMENT LOOP COUNTER
   F012 16FC    JNE LP ADD ANOTHER WORD
   F014 0000    DATA 0
   F016      *
   F016      * THIS IS THE ADD SUBROUTINE
   F016      ADD EQU $
   F00E*F016
   F016      []
```

Enter:
```
<SP> A *PT+,@SUM <SP> <SP> ADD NEXT VALUE <CR>
```

Comment: This statement increments the data pointer (PT) and adds the data in the data block.

Display:

```
ADDR DATA LB INST
F000      CT EQU 0 LOOP COUNTER WILL BE REGISTER R0
F000      PT EQU 1 BLOCK POINTER WILL BE REGISTER R1
F000      * INITIALIZE
F000 C020    MOV @BS,CT INITIALIZE LOOP COUNTER
F002R0000
F004 C060    MOV @BP,PT INITIALIZE BLOCK POINTER
F006R0000
F008 04E0    CLR @SUM
F00AR0000
F00C      *MAIN LOOP
F00C 06A0 LP BL @ADD
F00ER0000
F010 0600    DEC CT DECREMENT LOOP COUNTER
F012 16FC    JNE LP ADD ANOTHER WORD
F014 0000    DATA 0
F016      *
F016      * THIS IS THE ADD SUBROUTINE
F016      ADD EQU $
F00E*F016
F016 A831    A *PT+,@SUM  ADD NEXT VALUE
F018RF00A
F01A      []
```

Enter: <SP> RT <CR>

Comment: This statement returns execution to the statement following the branch-and-link statement.

Display:

```
ADDR DATA LB INST
F000      PT EQU 1 BLOCK POINTER WILL BE REGISTER R1
F000      * INITIALIZE
F000 C020    MOV @BS,CT INITIALIZE LOOP COUNTER
F002R0000
F004 C060    MOV @BP,PT INITIALIZE BLOCK POINTER
F006R0000
F008 04E0    CLR @SUM
F00AR0000
F00C      *MAIN LOOP
F00C 06A0 LP BL @ADD
F00ER0000
F010 0600    DEC CT DECREMENT LOOP COUNTER
F012 16FC    JNE LP ADD ANOTHER WORD
F014 0000    DATA 0
F016      *
F016      * THIS IS THE ADD SUBROUTINE
F016      ADD EQU $
F00E*F016
F016 A831    A *PT+,@SUM  ADD NEXT VALUE
F018RF00A
F01A 045B    RT
F01C      []
```

Enter:

```
* END OF SUBROUTINE <CR>
* <CR>
* HERE IS THE DATA STRUCTURE <CR>
<SP> AORG >F040 <CR>
```

Comment: The AORG directive gives the absolute origination of location >F040 for the next instruction. Each successive instruction (or directive) will reside in the next higher location. Notice that the next address displayed is the one specified by the AORG directive.

Display:

```
F002R0000
F004 C060    MOV @BS,CT INITIALIZE BLOCK POINTER
F006R0000
F008 04E0    CLR @SUM
F00AR0000
F00C        *MAIN LOOP
F00C 06A0 LP BL @ADD
F00ER0000
F010 0600    DEC CT DECREMENT LOOP COUNTER
F012 16FC    JNE LP ADD ANOTHER WORD
F014 0000    DATA 0
F016        *
F016        * THIS IS THE ADD SUBROUTINE
F016        ADD EQU $
F00E*F016
F016 A831    A *PT+,@SUM  ADD NEXT VALUE
F018RF00A
F01A 045B    RT
F01C        * END OF SUBROUTINE
F01C        *
F01C        * HERE IS THE DATA
F01C          AORG >F040
F040        []
```

Enter: SUM BSS 2 <CR>

Comment: This statement reserves a 2-byte buffer at symbolic address SUM and resolves the forward references at locations >F018 and >F00A.

Display:

```
F006R0000
F008 04E0    CLR@SUM
F00AR0000
F00C        *MAIN LOOP
F00C 06A0 LP BL @ADD
F00ER0000
F010 0600    DEC CT DECREMENT LOOP COUNTER
F012 16FC    JNE LP ADD ANOTHER WORD
F014 0000    DATA 0
F016        *
F016        * THIS IS THE ADD SUBROUTINE
F016        ADD EQU $
F00E*F016
F016 A831    A *PT+,@SUM  ADD NEXT VALUE
F018RF00A
F01A 045B    RT
F01C        * END OF SUBROUTINE
F01C        *
F01C        * HERE IS THE DATA
F01C           AORG >F040
F040        SUM BSS 2
F018*F040
F00A*F040
F042        []
```

Enter: BS DATA 7 <CR>

Comment: This statement defines the block size (BS) of the data block DB1 and resolves a forward reference at location >F002.

Display:

```
F00AR0000
F00C        *MAIN LOOP
F00C 06A0 LP BL @ADD
F00ER0000
F010 0600    DEC CT DECREMENT LOOP COUNTER
F012 16FC    JNE LP ADD ANOTHER WORD
F014 0000    DATA 0
F016        *
F016        * THIS IS THE ADD SUBROUTINE
F016        ADD EQU $
F00E*F016
F016 A831    A *PT+,@SUM  ADD NEXT VALUE
F018RF00A
F01A 045B    RT
F01C        * END OF SUBROUTINE
F01C        *
F01C        * HERE IS THE DATA
F01C           AORG >F040
F040        SUM BSS 2
F018*F040
F00A*F040
F042 0007 BS DATA 7
F002*F042
F044        []
```

Enter: BP DATA DB1 <SP> <SP> DATA BLOCK ONE <CR>

Comment: This statement defines the block pointer (BP) as symbol DB1 (by way of an unresolved forward reference) and resolves a previously unresolved forward reference from location >F006. A comment is added following two spaces (<SP><SP>).

Display:

```
F00C 06A0 LP BL @ADD
F00ER0000
F010 0600    DEC CT DECREMENT LOOP COUNTER
F012 16FC    JNE LP ADD ANOTHER WORD
F014 0000    DATA 0
F016      *
F016      * THIS IS THE ADD SUBROUTINE
F016      ADD EQU $
F00E*F016
F016 A831    A *PT+,@SUM  ADD NEXT VALUE
F018RF00A
F01A 045B    RT
F01C      * END OF SUBROUTINE
F01C      *
F01C      * HERE IS THE DATA
F01C         AORG >F040
F040      SUM BSS 2
F018*F040
F00A*F040
F042 0007 BS DATA 7
F002*F042
F044R0000 BP DATA DB1  DATA BLOCK ONE
F006*F044
F046      []
```

Enter: DB1 DATA 27,63,49,28,6,5,18<CR>

Comment: This statement places the data to be added by the program in the next seven successive locations. The DATA directive is preceded by the symbol DB1 in the LABEL column, this resolves a forward reference from location >F044.

Display:

```
F00E*F016
F016 A831    A *PT+,@SUM  ADD NEXT VALUE
F018RF00A
F01A 045B    RT
F01C       * END OF SUBROUTINE
F01C       *
F01C       * HERE IS THE DATA
F01C         AORG >F040
F040       SUM BSS 2
F018*F040
F00A*F040
F042 0007 BS DATA 7
F002*F042
F044R0000 BP DATA DB1  DATA BLOCK ONE
F006*F044
F046 001B DB1 DATA 27,63,49,28,6,5,18
F048 003F
F04A 0031
F04C 001C
F04E 0006
F050 0005
F052 0012
F044*F046
F054 []
```

Enter: <SP> END <CR>

Comment: The END directive quits the TMS9995 Emulator's assembler. The value shown to the right of END in the following display indicates the amount of unresolved references when the assembler was quit. In this case, there are no unresolved forward references which is reflected by the 0000 in the END statement.

Display:

```
F018RF00A
F01A 045B    RT
F01C       * END OF SUBROUTINE
F01C       *
F01C       * HERE IS THE DATA
F01C          AORG >F040
F040       SUM BSS 2
F018*F040
F00A*F040
F042       BS C
F042 0007 BS DATA 7
F002*F042
F044R0000 BP DATA DB1  DATA BLOCK ONE
F006*F044
F046 001B DB1 DATA 27,63,49,28,6,5,18
F048 003F
F04A 0031
F04C 001C
F04E 0006
F050 0005
F052 0012
F044*F046
F054          END   0000
?
```

You can see the program operate, if you desire, as follows:

Display: ?

Enter: MR<CR>

Display: Enter:

```
MR
  WP = 0000    F080
  PC = 0000    F000
  ST = 0000    <CR>
?
```

Comment: This execution of the Modify Registers (MR) command sets the workspace pointer (WP) to a region outside the program area, program counter (PC) to the beginning of the program (>F000), and status register (ST) to 0.

Enter: SNAP,SS,DW,DM(F040,0)<CR>

Comment: The procedure entered above functions to:

1. Freeze the display on the screen to make viewing the execution easier.

2. Execute the program in the single-step (SS) mode.

3. Display the workspace registers (DW).

4. Display the contents of memory location >F040 (SUM buffer).

Display:

```
SNAP SS DW DM(F040 0)
 SS
   WP=F080  PC=F004  ST=C000   NEXT:  F004 C060  MOV  @>F044,R1
 DW
   R0 =  0007  R4 =  0000  R8  =  0000  R12 =  0000
   R1 =  0000  R5 =  0000  R9  =  0000  R13 =  0000
   R2 =  0000  R6 =  0000  R10 =  0000  R14 =  0000
   R3 =  0000  R7 =  0000  R11 =  0000  R15 =  0000
 DM
   F040=0000                                      ..
?
```

Comment: The first instruction has been executed, as indicated by the register display of the SS command (PC=F004). Note that register R0 contains the value loaded into the loop counter (CT) by the instruction at address >F000.

Enter: <SP>

Display:

```
SNAP SS DW DM(F040 0)
 SS
   WP=F080  PC=F008  ST=8000   NEXT:  F008 04E0  CLR  @>F040
 DW
   R0 =  0007  R4 =  0000  R8  =  0000  R12 =  0000
   R1 =  F046  R5 =  0000  R9  =  0000  R13 =  0000
   R2 =  0000  R6 =  0000  R10 =  0000  R14 =  0000
   R3 =  0000  R7 =  0000  R11 =  0000  R15 =  0000
 DM
   F040=0000                                      ..
```

Comment: The second instruction has been executed, as indicated by the register display of the SS command (PC=F008). Note that register R1 now contains the value loaded into the data pointer (PT) by the instruction at address >F004.

The program can be executed to step automatically through the instructions from here on (with the fixed display) as follows:

Enter: *

Comment: The display will remain fixed on the screen, with the following exceptions:

1. Register R0 (loop counter - CT) will decrement by 1 each time that the instruction at address >F000 is executed. (When this value reaches 0, the DATA statement at address >F014 will be encountered and execution will stop.)

2. Register R1 (data pointer - PT) will increment to the address of the next data each time that the instruction at address >F046 is executed. (This loads the next data value into the SUM buffer, address >F044.)

3. The value in the SUM buffer (address >F044) will increase by the amount of the next data statement each time that the instruction at address >F046 is executed.

SECTION 5

COMMUNICATION

5.1 INTRODUCTION

This section contains a discussion of the TMS9995 Emulator's communication capability. The communications system which links the user's terminal, a host computer system, and a PROM programmer or printer is discussed in detail.

The following commands are discussed:

* IPORT - Configures the individual communication ports
* HOST - Establishes and controls communication with a host computer system
* UL - Defines parameters for uploading data from the emulator to an external device
* DL - Defines parameters used for downloading data from an external device to the emulator
* IHC - Defines special control characters used to indicate the beginning and end of transferred object files

The representation of individual characters as well as the structure and control of data records is also explained.

There are several sample communication sessions which present step-by-step procedures for communicating with a variety of common systems.

A trouble-shooting chart assists in diagnosing and solving problems which frequently occur in communicating with external devices.

The functions of the communication link are to:

* Transmit data files from the emulator to an external device (upload)
* Receive data files from an external device and store them in the emulator memory (download)
* Transfer downloaded data received from an external device to a PROM programmer or logging device
* Transmit data from the emulator's memory to a PROM programmer or logging device

5.2 SYSTEM DESCRIPTION

Ports A, C, and D on the XDS chassis are configured as EIA RS-232-C ports and are discussed in the following paragraphs. (Port B is not used at this time and is not included in the discussion.)

* Port A is used to communicate with the user's terminal in the single emulator mode. The AUTOBAUD sequence (invoked with a dual carriage return on power-up) automatically matches port A's baud rate with that of the user's terminal. (Texas Instruments recommends a terminal with a baud rate of 9600 bps for optimum operation.)

* Port C is generally used as a link with a printer to log the emulator's operation or with a PROM programmer to upload the final program into PROM. The initial power-up baud rate matches that of port A after the autobaud sequence. The baud rate can be programmed to a new value by executing the IPORT command.

* Port D links the TMS9995 Emulator with the host system and operates the same as port C. Data files may be transferred in either direction, to or from the host.

All ports on the XDS unit are configured as follows after the autobaud sequence is completed.

* BAUD = Baud rate of user's terminal
* PARITY = EVEN parity
* STOP BITS = 2 bits
* BITS/CHAR = 7 bits

Figure 5-1 illustrates a typical stand-alone XDS system with peripheral devices connected to its communication ports.

```
---------------------          --------------      ------------
|       HOST        |          | TMS9995    |      | PRINTER  |
|     COMPUTER      |          | EMULATOR   |      |    OR    |
|                   |          |     Port C |------|   PROM   |
|                   |----------|Port D      |      |PROGRAMMER|
---------------------          |     Port A |      ------------
          |                    --------------
          |                          |
-------------------            --------------
|      HOST       |            |   USER'S   |
|    TERMINAL     |            |  TERMINAL  |
-------------------            --------------
```

FIGURE 5-1. TMS9995 EMULATOR LINKED TO A HOST COMPUTER SYSTEM

A single emulator connected to a host system can be treated as another terminal, a cassette tape drive, a keyboard/printer, or some other I/O device depending on the host's communication configuration.

The programmer cay use the terminal to interact directly with the host computer. In this case the XDS unit acts as a communication channel to the host. The host computer and the user's terminal may operate at different baud rates. XDS will make the proper adjustments so that this direct communication between the user and the host is possible.

When the emulator is connected to a host computer system, the design engineer is able to create programs on a larger more sophisticated system before testing them in the emulator. The host computer's utility software is available for editing text, macroassembling, linking, and preliminary debugging. Programs created on the host system are then downloaded into the TMS9995 Emulator for execution, simulation, final testing, and debugging. The host system provides the communications hardware as well as the supporting software necessary for data transfer between the TMS9995 Emulator and the host system.

A personal computer connected to Port A can act as the host computer system. In this configuration, object files may be transferred to or from the user's terminal much the same as when using a host computer on Port D.

When operating in the multi-processor mode, as many as nine XDS units can be connected in a linear array . The first XDS unit is connected to the user's terminal through Port A and to the next XDS unit through Port D. The succeeding units use Port A to connect to the preceding XDS unit and Port D to connect to the next unit in the chain. The last XDS unit in the chain is connected to the host computer system through Port D. (See Section 6 for more information on multi-processor emulation.)

5.3 COMMUNICATION COMMANDS

The following commands are used to configure the XDS ports for communication to external devices.

* IPORT - Configures the baud rate, parity, number of stop bits, and the number of data bits for ports C and D.

* HOST - Enters the terminal mode allowing XDS to communicate with the host computer system on port D.

* DL - Configures the download parameters including the load bias, the object code format, the destination and source of the downloaded data, and the handshaking protocols to be used.

* UL - Configures the upload parameters including the start and end address locations, the object code format, the type of data (i.e., 16-bit word or 8-bit byte), the source and destination of the data, and the handshake protocols to be used for data flow control.

* IHC - Defines the control characters used for starting and ending downloads or uploads as well as pass-through characters when PROTOCOL parameters are defined as NONE or TEK. The user then specifies these control-character sequences.

5.3.1 Configuring the Communication Ports

The communication ports must be configured to conform to specifications of the external devices. The IPORT (Initialize Port) command is used to set up the parameters for Ports C and D on the XDS chassis. If the IPORT command is entered using an exclamation point <!> as the terminator, the parameters for the command appear without execution of the command.

Enter: IPORT!<CR>

Display:

```
IPORT!
  PORT [0=HOST("D"), 1=LOG/PROM("C")]                      = 0
  BAUD [19.2K, 9.6K, 4.8K, 2.4K, 1.2K, 600, 300, 110] = 0
  PARITY [0=OFF, 1=ODD, 2=EVEN]                            = 0
  STOP BITS (0=2, 1=1)                                     = 0
  BITS/CHAR (0=7, 1=8)                                     = 0
?
```

Although the IPORT parameter values appear as zeroes above, they are actually determined by the autobaud sequence after power-up of the unit and therefore may be different.

5.3.2 Downloading Data into the Emulator

Once the selected port (port D for the external device or port C for the PROM programmer) has been configured, the download or upload parameters can be defined. If the power-up values for the parameters associated with DL (Download) and UL (Upload) commands do not conform to those of the external device, the appropriate command should be executed at this time. When downloading or uploading through the host port (port D), enter the HOST command after executing the download or upload command.

Entering DL!<CR> in the preview mode, the download parameters appear as follows.

Enter: DL!<CR>

Display:

```
DL!

  LOAD BIAS                                          = 0000
  FORMAT      [0=TI, 1=TEK]                          = 0
  DESTINATION [0=MEMORY, 1=PROM]                     = 0
  PROTOCOL    [0=NONE, 1=TEK, 2=ASR, 3=VAX]          = 0
  SOURCE      [0=HOST, 1=USER]                       = 0
?
```

The value entered in the LOAD BIAS parameter determines the beginning address for the downloaded file. If the data is in TI format, an address other than zero can be specified and the file is considered relocatable. When Tektronix data format is selected, the LOAD BIAS is ignored and the beginning address defaults to zero.

The FORMAT parameter offers a choice of two data formats, TI or TEK (Tektronix). The TI format may be received as either TI normal ASCII or TI compressed code. The emulator automatically makes the necessary adjustments for either format and the user need not specify the TI format for a download. (Note: The specific TI format must be indicated on an upload, however.)

The DESTINATION parameter specifies where the downloaded data is to be directed.

* MEMORY selects emulator memory as the download destination.
* PROM specifies the data is to be transmitted directly to the PROM programmer (or logging device) through Port C.

The PROTOCOL parameter provides four selections for handshaking protocols:

* NONE The IHC command must be executed to define the proper control characters.
* TEK The IHC command must be executed to define the proper control characters.
* ASR Appropriate control characters are automatically defined.
* VAX Appropriate control characters are automatically defined.

Table 5-1 summarizes the values specified for the control characters used in record level control.

TABLE 5-1. SUMMARY OF HANDSHAKING PROTOCOLS FOR RECORD TRANSFER

	PROTOCOL Value			
	NONE	TEK	ASR	VAX*
Download Start	IHC	IHC	CTRL-R	CTRL-V
Download End	IHC	IHC	CTRL-S	CTRL-W
Upload Start	IHC	IHC	CTRL-Q	CTRL-A
Upload End	IHC	IHC	CTRL-@	CTRL-Z
Passthrough	IHC	IHC	CTRL-P	CTRL-P

IHC = Defined by IHC command parameters.
* Power-up value after autobaud.

The SOURCE parameter allows the user to select either the host or the user's terminal as a source of downloaded data. If the user's terminal is a single-user system, such as a personal computer, and data can be stored internally before transfer to the emulator, the user has the option of choosing the terminal as the data source.

NOTE

When downloading from the user's port (port A) using a stand-alone, single-user system such as a personal computer, the file is transmitted immediately after executing DL. There is no download-start character sent, but the appropriate download-end character is required. The ASR handshakes are not supported when downloading from the user's port. Do not execute the HOST command when preparing to use port A.

5.3.2.1 Summary of Steps for Downloads

The steps for performing a download from a host computer system to the emulator are as follows.

1. Execute the IPORT command to configure the host port to the proper baud rate, parity, number of stop bits, and the number of bits per character.

2. Execute the DL command to define the download parameters.

3. Execute the HOST command to enter the terminal mode and begin the data transfer.

4. Log on to the host system.

5. When prompted, send the proper host command to initiate the downloading of the specified file to the emulator.

6. If the host does not automatically send a download-complete character, the user enters the character from the terminal. Most systems will send the character so it probably will not be necessary.

7. If the session is complete, enter the appropriate log-off characters to sign off the host system.

8. Enter a CTRL-E to exit the HOST mode and return control to the monitor.

5.3.3 Uploading Data from Emulator

When uploading data from the emulator to a host, the UL (Upload) command is used to define the upload parameters.

Display: ?

Enter: UL!<CR>

Display:

```
UL!

  START ADDRESS                                        = 0000
  END   ADDRESS                                        = 0000
  FORMAT [0=TI, 1=TICOMP, 2=TEK, 3=INTEL, 4=MR]        = 0
  DATA    (0=WORD, 1=HI BYTE, 2=LO BYTE)               = 0
  DESTINATION [0=HOST, 1=PROM, 2=USER]                 = 0
  PROTOCOL     [0=NONE, 1=TEK, 2=ASR, 3=VAX]           = 0
?
```

The START ADDRESS and END ADDRESS parameters define the segment of emulator memory to be uploaded. In the above example, the user has specified in an earlier execution of the UL command that all of the program memory is to be uploaded.

The FORMAT parameter informs the monitor to send the data formatted as one of the following formats: TI-normal ASCII, TI-compressed, Tektronix hexadecimal, Intel Intellec 8/MDS, or Motorola Exorcisor. Of course, the format chosen will depend on the format required by the external device.

The DATA parameter provides a means to transmit data as whole words (i.e., two bytes) or one byte at a time. Sending data one byte at a time might be necessary with certain object-code formats that require the word be stored in two different hardware locations. If the data is not sent as one word, the user must make two passes to transmit data to the external device. The first pass might use the HI BYTE value and the second pass the LO BYTE value.

The DESTINATION is the port to which the data is to be sent. The specified port must be initialized for compatibility with the external device before an upload is attempted. This requires the execution of the IPORT (Initialize Port) command for both port C (the LOG/PROM port) and port D (the host port).

The PROTOCOL selection includes NONE, Tektronix, ASR, or VAX handshakes. When NONE or TEK is specified, the user must specify the file-level control characters for starting and ending an upload or download using the IHC command. See Table 5-1 for a summary of the remaining PROTOCOL control characters. When ASR or VAX handshakes are selected, the control characters are automatically loaded into the parameters and IHC execution is not necessary.

5.3.3.1 Summary of Steps for Uploads

The steps for performing an upload to a host computer are executed in the following order.

1. Execute the IPORT command to configure the host port to the proper baud rate, parity, number of stop bits, and number of bits per character.

2. Execute the UL command to define the upload parameters.

3. Execute the HOST command to enter the terminal mode and begin the data transfer.

4. Log on to the host system.

5. When prompted, send the proper host command to set up a file to accept the uploaded object code from the emulator.

6. If the host computer does not automatically send a start-upload control character to the emulator, enter the proper character from the terminal and the upload begins.

7. If the session is completed, send the proper log-off characters to sign off.

8. Enter CTRL-E to exit the HOST terminal mode and return control to the monitor.

NOTE

If PROM is selected as a destination, port C must be configured to conform to the data signal specifications of the PROM programmer or logging device (printer) connected to this port.

5.3.3.2 Uploading to PROM Programmer or User's Terminal

When uploading to either a PROM programmer (via Port C) or to the user's terminal (via port A), the UL command initiates the upload when it is executed. The HOST command is not executed.

5.3.3.3 Using the HOST Command

When the HOST command is executed, the TMS9995 Emulator enters the terminal mode of operation. Input and output are directed through port D to the host computer system. The DL command's SOURCE parameter defaults to HOST as does the UL command's DESTINATION parameter.

To exit the HOST command, enter CTRL-E, which will terminate the terminal mode.

5.3.4 Programming Host Control Characters

When performing an upload or a download, special control characters are used to inform the TMS9995 Emulator and the external device that data is about to be transmitted or that data has been received. These control characters can be programmed into the TMS9995 Emulator's monitor so that they conform to those characters recognized by the external device. The IHC (Initialize to Host Control Characters) command is used to accomplish this function.

The IHC command defines special control characters that inform the emulator's monitor that a download has started or terminated, an upload has started or terminated, or that a control character is to be passed on to the host with no action taken. The last feature allows the user to pass a control character through to the host without the XDS system acting on the character.

The IHC command is an interactive command and when executed displays the following parameters for user response.

```
DEPRESS KEY FOR DOWNLOAD START
DEPRESS KEY FOR DOWNLOAD END
DEPRESS KEY FOR UPLOAD START
DEPRESS KEY FOR UPLOAD END
DEPRESS KEY FOR PASS THROUGH CHARACTER
```

Since the control characters used by the IHC command are not printable, the power-up values are set up for a VAX host computer system and are listed as follows.

Download start	=	CTRL-V
Download end	=	CTRL-W
Upload start	=	CTRL-A
Upload end	=	CTRL-Z
Pass through	=	CTRL-P

If the external device does not respond to these particular control characters, the user can use the IHC command to remap any or all of these key sequences to conform to those of the external device. When the prompts are displayed, the user presses the specific control character to be used when data is exchanged.

Selecting VAX or ASR automatically assigns values for these parameters and the IHC command does not have to be executed. If NONE or TEK are selected, the IHC command is executed to define the control-character sequences to be used for upload and download control.

The following flowcharts are presented to give the user a basic understanding of the procedures used by the TMS9995 Emulator for processing input.

* Figure 5-2 illustrates the logic behind the treatment of input from the user's terminal or the host computer system. When characters are entered from the user's terminal they are processed by a service routine.

* Figure 5-3 illustrates details of the user's service routine.

* Figure 5-4 illustrates the host system's service routine.

```
               +-------+
               | START |
               +-------+
                   |
      +---------->|
      |            |
      |            V
      |           * *
      |         *     *
      |       *Input at *   YES
      |     * User's Port *----------+
      |       *    ?     *           |
      |         *     *              V
      |           * *          +----------------+
      |            |           | User's Service |
      |            | NO        |    Routine     |
      |            |           +----------------+
      |            |                   |
      |            | <-----------------+
      |            |
      |            V
      |           * *
      |         *     *
      |       *Input at *   YES
      |     *  Host Port  *----------+
      |       *    ?     *           |
      |         *     *              V
      |           * *          +----------------+
      |            |           | Host's Service |
      |            | NO        |    Routine     |
      |            |           +----------------+
      |            |                   |
      |            | <-----------------+
      |            |
      +------------+
```

FIGURE 5-2. FLOWCHART OF GENERAL INPUT HANDLING ROUTINE

```
     +-----------------+
     | READ CHARACTER  |
     +-----------------+
             |
             V
            * *
          *     *
        *         *
      *  Download-  *     Yes
    *      Start      *---------------+
      *  Character  *                 |
        *    ?    *                   V
          *     *          +---------------------+
            * *            | Send [CR] to Host   |
             |  No         +---------------------+
             |                        |
             |          +---------------------------+
             |          | Receive Host Object File  |
             |          +---------------------------+
             |                        |
             |          +---------------------------+
             |          |   Wait for Download-end   |
             |          | Character or Host Prompt  |
             |          +---------------------------|
             V                        |
            * *                       +-------------------->|
          *     *                                            |
        * Upload- *     YES                                  |
      *    Start    *-----------------+                      |
        *Character*                   |                      |
          *     *            +--------------------+          |
            * *              | Send File to Host  |          |
             |  No           +--------------------+          |
             |                        |                      |
             |               +--------------------+          |
             |               |   Send Upload-End  |          |
             V               | Character to Host  |          |
            * *              +--------------------+          |
          *     *                     |                      |
        *         *                   +-------------------->|
      * Passthrough * Yes                                    |
        *Character* ------------------+                      |
          *  ?  *                     |                      |
            * *          +--------------------------+        |
             |  No       |Read Character from User  |        |
             |           +--------------------------+        |
             |                        |                      |
             |           +--------------------------+        |
             V           | Send Character to Host   |        |
         +-------+       +--------------------------+        |
         |   A   |                    |                      |
         +-------+                    +-------------------->|
                                                             |
                                                         +-------+
                                                         |   B   |
                                                         +-------+
```

FIGURE 5-3. FLOWCHART OF DETAILS OF USER'S SERVICE ROUTINE

```
    +-------+                                                +-------+
    |   A   |                                                |   B   |
    +-------+                                                +-------+
        |                                                        |
       * *                                                       |
     *     *                                                     |
   *Download-*                                                   |
 *End Character*    Yes   +-----------------+                    |
   *    ?    *    ------->| Ignore Character |---------->|
     *     *              +-----------------+                    |
       * *                                                       |
        | No                                                     |
        V                                                        |
       * *                                                       |
     *     *   Yes   +-------------------+                       |
   * CTRL-E   *------>| Exit Terminal Mode |                      |
     *  ?  *         +-------------------+                       |
       * *                                                       |
        | No                                                     |
        V                                                        |
 +---------------+                                               |
 | Send Character |                                              |
 |  to Host Port  |                                              |
 +---------------+                                               |
       |<------------------------------------------------------+
       |
       V
```

FIGURE 5-3. FLOWCHART OF DETAILS OF USER'S SERVICE ROUTINE (Continued)

```
+----------------+
| READ CHARACTER |
+----------------+
        |
        V
       * *
     *     *
   *         *
  *  Download-  *    Yes
 *     Start      *---------------+
  *  Character  *                 |
   *    ?     *                   |
     *     *                      |
       * *                        V
        | No            +--------------------------+
        |               | Receive Host Object File |
        |               +--------------------------+
        |                         |
        |               +--------------------------+
        |               |  Wait for Download-end   |
        |               | Character or Host Prompt |
        V               +--------------------------|
       * *                        |
     *     *                      +-------------------->|
   * Upload- *    YES                                   |
  *   Start    *----------------->+                     |
   *Character*                    |                     |
     *     *              +-------------------+         |
       * *                | Send File to Host |         |
        | No              +-------------------+         |
        |                         |                     |
        |                 +-------------------+         |
        |                 |  Send Upload-End  |         |
        |                 | Character to Host |         |
        |                 +-------------------+         |
        |                         |                     |
        V                         +-------------------->|
       * *                                              |
     *     *                                            |
   *Download-*    Yes     +-------------------+         |
  *End Character* ------->| Ignore Character  |-------->|
   *    ?    *            +-------------------+         |
     *     *                                            |
       * *                                              |
        | No                                            |
        V                                               |
+----------------+                                      |
| Send Character |                                      |
| to User's Port |                                      |
+----------------+                                      |
       |<-----------------------------------------------+
       |
       V
```

FIGURE 5-4. FLOWCHART OF DETAILS OF HOST'S SERVICE ROUTINE

5.4 DATA TRANSFER

When discussing data formats the following data structures must be considered.

* The character-level bit patterns representing individual characters

* Record formats for an object file

* File structure

5.4.1 Character Representation

When the emulator is powered up, characters are generally represented by the following standard asyncronous bit pattern.

* One start bit - LOW

* Seven data bits - HIGH or LOW

* One parity bit - HIGH or LOW

* Two stop bits - HIGH

The bit pattern can be changed by using the IPORT (Initialize Port) command. The number of data bits can be set to either seven or eight bits. If a parity bit is transmitted, it immediately follows the data bits. The number of stop bits may be designated as either one bit or two bits.

The parameter values, which are set by the IPORT command, must conform to the respective parameter values on the external device's communication port. The object-file format used may also be a determining factor in the configuration of these parameters. The power-up bit pattern is illustrated in Figure 5-5.

If the external device is configured to ODD or EVEN parity, the PARITY parameter value (from the IPORT command) must match the parity on the external device when uploading from the emulator. When the IPORT command's PARITY parameter is designated as ODD or EVEN, the emulator will set the parity bit (#9 in Figure 5-5 above) accordingly as the data is transmitted to the external device. However, the emulator does not check for parity when receiving data, so the parity bit is ignored.

When the number of bits per character (BITS/CHAR) is set to eight bits with the parity bit enabled, the bit pattern transmitted is as shown in Figure 5-6.

```
--------------------------------------------------------------------

               |<-------- Data Bits ------>|
                ---------       -----       -------------
 1 ------>   s |   |   |       |   |       | p | s | s | s
             t |   |   |       |   |       | a | t | t | t
             a |   |   |       |   |       | r | o | o | a
             r |   |   |       |   |       | i | p | p | r
             t | 1 | 1 | 0   0 | 1 | 0   0 | t |   |   | t
 0 ------> |---|   |   |---|---|   |---|---| y |   |   |---|
              LSB                       MSB
  Bit # -->  1   2   3   4   5   6   7   8   9  10  11

           MSB = Most Significant Bit
           LSB = Least Significant Bit
--------------------------------------------------------------------
```

FIGURE 5-5. THE POWER-UP BIT PATTERN FOR DATA COMMUNICATIONS

```
--------------------------------------------------------------------

               |<---------- Data Bits --------->|
                ---------       -----       -----     ---------
 1 ------>   s |       |       |   |       |   | p | s | s | s
             t |       |       |   |       |   | a | t | t | t
             a |       |       |   |       |   | r | o | o | a
             r |       |       |   |       |   | i | p | p | r
             t | 1   1 | 0   0 | 1 | 0   0 | 1 | t |   |   | t
 0 ------> |---|   |   |---|---|   |---|---|   |-y-|   |   |---|
              LSB                           MSB
  Bit # -->  1   2   3   4   5   6   7   8   9  10  11

           MSB = Most Significant Bit
           LSB = Least Significant Bit
--------------------------------------------------------------------
```

FIGURE 5-6. BIT PATTERN WITH 8 BITS/CHARACTER

When TI-compressed object-code format is used, the data is treated as eight-bit binary data and not as standard ASCII characters. In this case, the BITS/CHAR parameter must be set for transmitting and receiving data fields with eight bits per character.

Table 5-2 summarizes the bit patterns based upon various configurations of the PARITY, DATA BITS, and STOP BITS in the data signal.

TABLE 5-2. POSSIBLE BIT PATTERNS FOR PARITY, DATA BITS, and STOP BITS

PARITY (ODD or EVEN)	No. STOP BITS	No. DATA BITS	Bits Transmitted: Data 1	2	3	4	5	6	7	8	Parity Bit	Stop 1	Stop 2
OFF	2	7	x	x	x	x	x	x	x			x	x
OFF	2	8	x	x	x	x	x	x	x	x		x	x
OFF	1	7	x	x	x	x	x	x	x			x	
OFF	1	8	x	x	x	x	x	x	x	x		x	
* ON	2	7	x	x	x	x	x	x	x		x	x	x
ON	2	8	x	x	x	x	x	x	x	x	x	x	x
ON	1	7	x	x	x	x	x	x	x		x	x	
ON	1	8	x	x	x	x	x	x	x	x	x	x	

* Power-up configuration with PARITY = EVEN.

5.4.2 Control of Data Transfer

Control of data transfer to or from the TMS9995 Emulator may be considered at three levels.

* Character This level uses either XON/XOFF characters or the RS-232-C circuits for control signals.
* Record This level is supported by the monitor for ASR and Tektronix protocols which use special characters at the end of each record as the controlling signals.
* File This level uses special records or control characters at the beginning and end of each object file for control signals.

5.4.3 Character-level control

Standard I/O XON and XOFF interrupt functions are supported on downloads (except when Tektronix Object-Code Format is specified). Data is loaded into an 80-character buffer. When the buffer is within 20 characters of being full, XOFF is sent to interrupt transmission from the external device. The monitor sends the XON signal when the buffer has only 20 characters remaining.

XON and XOFF are combined with hardware handshakes to ensure control of data transfer by using the Ready-to-Send (RTS) line on the RS-232 circuit. This line is set HIGH (XON) until an XOFF signal is sent by the emulator. The RTS line is then set low until an XON signal is sent by the emulator. RTS returns to HIGH at that point.

It is also possible to use the Clear-to-Send (CTS) line for character transfer control. If the external device is programmed to use this line for control, the XDS unit will respond to the signals generated on this line.

5.4.4 Record-level Control

The PROTOCOL parameter values TEK and ASR automatically configure the EIA port for communicating with an external device which uses these protocols.

5.4.4.1 Tektronix protocol

The Tektronix (TEK) handshake protocol uses special character sequences entered at the end of a record to indicate successful transfer of the record.

* ASCII "0" followed by a <CR> is sent by the emulator to indicate that a record has been received without error. The checksum data is used to detect errors in the data.

* ASCII "7" followed by a <CR> is sent if the record appears to have a transmission error. This indicates that the record should be retransmitted. The upload session is aborted after 15 unsuccessful attempts to transmit a record.

NOTE

When using Tektronix Object-Code Format, the XON/XOFF and CTS/RTS handshakes are not functional.

5.4.4.2 ASR Protocol

The ASR handshaking protocol causes the TMS9995 Emulator to act as a Texas Instruments Model 733 ASR/KSR data terminal. When the communication port is configured for ASR handshaking, the TMS9995 monitor responds to DC1 = ON and to DC3 = OFF as well as RTS control signals.

5.4.5 Object File-level Control

The TMS9995 Emulator's communication protocol supports five data formats.

* TI normal ASCII
* TI compressed
* Tektronix (Tek) hexadecimal
* Intel Intellec 8/MDS
* Motorola Exorcisor

The TI formats and the Tektronix are available for data that is uploaded or downloaded. The Intel and Motorola formats are used only in uploads from the emulator to the host.

Specific information regarding these formats is provided in Appendix D.

5.5 SAMPLE COMMUNICATION SESSIONS

The following is a sample session in which data is transferred from the host system to the emulator.

If the communication parameters have not been configured for the external devices (port D for the host system and port C for the logging device), the user must first do this. The use of the IPORT command is explained with an example in Section 3.

The IPORT, IHC, UL, and DL commands will be executed for each of the following examples to illustrate the complete process involved. The user should keep in mind that once the parameters for these commands are defined, the commands do not have to be executed again until a change in the parameter values is needed.

Example 1: Downloading object files from a DS990 DX10 host computer using the TTY/EIA interface.

Display: ?

Enter: IPORT<CR>

```
Display:                                                     Enter:

IPORT
  PORT [0=HOST("D"), 1=LOG/PROM("C")]                    = 0  <CR>
  BAUD [19.2K, 9.6K, 4.8K, 2.4K, 1.2K, 600, 300, 110] = 0  1.2K<CR>
  PARITY [0=OFF, 1=ODD, 2=EVEN]                          = 0  <CR>
  STOP BITS (0=2, 1=1)                                   = 0  <CR>
  BITS/CHAR (0=7, 1=8)                                   = 0  <CR>
?
```

The Download parameters are defined next.

Enter: DL<CR>

```
Display:                                          Enter:

DL
  LOAD BIAS                               = 0000  <CR>
  FORMAT [0=TI, 1=TEK]                    = 0     <CR>
  DESTINATION [0=MEMORY, 1=PROM]          = 0     <CR>
  PROTOCOL [0=NONE, 1=TEK, 2=ASR, 3=VAX]  = 0     ASR<CR>
  SOURCE [0=HOST, 1=USER]                 = 0     <CR>
?
```

Comment: The values have been set-up to copy concatenate using the ASR device's service routine.

Enter: HOST<CR>

Display: CONTROL-E EXITS
?

Comment: Entering the HOST command invokes the terminal mode. The CONTROL-E EXITS banner is displayed and the monitor waits for the download-start control character.

The <ESC>! log-on sequence is sent from the user's terminal to initiate the communication session.

Enter: <ESC><!>

Display: SYSTEM COMMAND INTERPRETER - PLEASE LOG ON
USER ID: [ ]

Enter: The user enters a valid identification code

Display: PASSCODE: [ ]

Enter: The user enters a valid passcode.

Comment: The emulator is now logged on to the host DS990 system as a terminal. The user may enter any valid DS990 command. The COPY CONCATENATE command will be used to transfer data from the DS990 to cassette port 02 which is connected to the XDS unit.

Display: []

Enter: CC<CR>

Display:	Enter:
COPY/CONCATENATE	
INPUT ACCESS NAME(S):	U.TESTFILE<CR>
OUTPUT ACCESS NAME:	CS02 <CR>
REPLACE:	NO<CR>
BACKGROUND:	NO<CR>
MAXIMUM RECORD LENGTH:	<CR>

Comment: The user enters the information requested in the above prompts specifying the source filename (U.TESTFILE) and the destination device (CS02). After the MAXIMUM RECORD LENGTH is specified the host computer sends a start-download character to the emulator and the object module is sent from the host and is stored in the emulator's memory.

Display: []

Comment: When the [] prompt returns to the emulator display, the download is complete.

Enter: Q<CR>

Comment: Q logs the user off the host system.

Enter: <CTRL-E>

Comment: A <CTRL-E> typed from the user's terminal exits the terminal mode and returns to the input mode with the object file now stored in the emulator memory.

Display: ?

Example 2: Uploading object files to the host DS990 computer system using the TTY/EIA interface.

Display: ?

Enter: IPORT<CR>

```
Display:                                                        Enter:

IPORT
  PORT [0=HOST("D"), 1=LOG/PROM("C")]                     = 0  <CR>
  BAUD [19.2K, 9.6K, 4.8K, 2.4K, 1.2K, 600, 300, 110] = 0  1.2K<CR>
  PARITY [0=OFF, 1=ODD, 2=EVEN]                           = 0  <CR>
  STOP BITS (0=2, 1=1)                                    = 0  <CR>
  BITS/CHAR (0=7, 1=8)                                    = 0  <CR>
?
```

The host port is now configured and the UL command is executed next.

Display: ?

Enter: UL<CR>

```
Display:                                                        Enter:

UL
  START ADDRESS                                     = 0000      <CR>
  END   ADDRESS                                     = 0000      <CR>
  FORMAT [0=TI, 1=TICOMP, 2=TEK, 3=INTEL, 4=MR]     = 0         <CR>
  DATA (0=WORD, 1=HI BYTE, 2=LO BYTE)               = 0         <CR>
  DESTINATION [0=HOST, 1=PROM, 2=USER]              = 0         <CR>
  PROTOCOL [0=NONE, 1=TEK, 2=ASR, 3=VAX]            = 0         ASR<CR>
?
```

Comment: The parameters are now specified with the values to copy concatenate using the ASR device's service routine.

Display: ?

Enter: HOST<CR>

Display: CONTROL-E EXITS
?

Comment: The user must log on to the host and supply the proper identification codes when prompted to do so.

Enter: <ESC><!>

Display: SYSTEM COMMAND INTERPRETER - PLEASE LOG ON
USER ID: []

Enter: The user enters a valid identification code.

Display: PASSCODE: []

Enter: The user enters a valid passcode.

Comment: The emulator is now logged on to the host DS990 system as a terminal. The user may enter any valid DS990 command. The COPY CONCATENATE command is used to receive data from the terminal.

Display: []

Enter: CC<CR>

Display:	Enter:
COPY/CONCATENATE	
INPUT ACCESS NAME(S):	CS01CR>
OUTPUT ACCESS NAME:	U.TESTFILE<CR>
REPLACE:	NO<CR>
BACKGROUND:	NO<CR>
MAXIMUM RECORD LENGTH:	<CR>

Comment: The user enters the information requested in the above prompts specifying the source of the file (CS01) and the destination filename (U.TESTFILE). After the MAXIMUM RECORD LENGTH is specified, the host computer sends a start-upload character to the emulator and the object module is sent from the emulator and is stored in the host as file U.TESTFILE.

Display: UPLOAD COMPLETE
[]

Enter: Q<CR>

Comment: The <Q> is sent to log off the DS990.

Enter: <CTRL-E>

Display: ?

Comment: CTRL-E exits the terminal mode and returns to the monitor input mode.

Example 3: Downloading object files from a VAX host computer.

The VT100 terminal on the VAX is connected to Port D on the emulator.

Display: ?

Enter: IPORT<CR>

```
Display:                                                    Enter:

IPORT
  PORT [0=HOST("D"), 1=LOG/PROM("C")]                  = 0  <CR>
  BAUD [19.2K, 9.6K, 4.8K, 2.4K, 1.2K, 600, 300, 110] = 0  <CR>
  PARITY [0=OFF, 1=ODD, 2=EVEN]                        = 0  <CR>
  STOP BITS (0=2, 1=1)                                 = 0  <CR>
  BITS/CHAR (0=7, 1=8)                                 = 0  <CR>
?
```

Comment: The actual values entered here will depend on the specific needs of the user.

The Download parameters are defined next.

Enter: DL<CR>

```
Display:                                           Enter:

DL
  LOAD BIAS                                 = 0000     <CR>
  FORMAT [0=TI, 1=TEK]                      = 0        <CR>
  DESTINATION [0=MEMORY, 1=PROM]            = 0        <CR>
  PROTOCOL [0=NONE, 1=TEK, 2=ASR, 3=VAX]    = 0        VAX<CR>
  SOURCE [0=HOST, 1=USER]                   = 0        <CR>
?
```

Comment: The values entered here will set-up the VAX protocol.

Comment: The download parameters have been defined.

Enter: HOST<CR>

Log on to the VAX system using the standard logon procedure.

Display: $

Enter: TYPE A.TESTFILE<CTRL-V>

Display: UPLOAD COMPLETE
$

Comment: The UPLOAD COMPLETE message and $ prompt appear when the download is complete.

Enter: <Log-off sequence>

Enter: CTRL-E<CR>

Display: ?

Comment: Exit HOST command and return control to the monitor.

Example 4: Uploading object file to a VAX host computer.

Enter: IPORT<CR>

```
Display:                                                      Enter:

IPORT
  PORT [0=HOST("D"), 1=LOG/PROM("C")]                     = 0  <CR>
  BAUD [19.2K, 9.6K, 4.8K, 2.4K, 1.2K, 600, 300, 110] = 0  <CR>
  PARITY [0=OFF, 1=ODD, 2=EVEN]                           = 0  <CR>
  STOP BITS (0=2, 1=1)                                    = 0  <CR>
  BITS/CHAR (0=7, 1=8)                                    = 0  <CR>
?
```

Comment: The actual values entered here will depend on the specific needs of the user.

The host port is now configured and the UL command is executed next.

Enter: UL<CR>

```
Display:                                                      Enter:

UL
  START ADDRESS                                    = 0000      <CR>
  END   ADDRESS                                    = 0000      <CR>
  FORMAT [0=TI, 1=TICOMP, 2=TEK, 3=INTEL, 4=MR]    = 0         <CR>
  DATA (0=WORD, 1=HI BYTE, 2=LO BYTE)              = 0         <CR>
  DESTINATION [0=HOST, 1=PROM, 2=USER]             = 0         <CR>
  PROTOCOL [0=NONE, 1=TEK, 2=ASR, 3=VAX]           = 0         VAX<CR>
?
```

Enter: HOST<CR>

Log on to the VAX system using the standard logon procedure.

Display: $

Enter: CREATE A.TESTFILE<CR><CTRL-A>

Display: $

Comment: The $ prompt appears when the upload is complete.

Enter: Log-off sequence

Enter: CTRL-E<CR>

Display: ?

Comment: CTRL-E exits terminal mode and returns control to the monitor.

Example 5: Downloading object file from DS990 host computer through an 820 keyboard printer port.

Port D on the emulator is connected to an 820 printer port on the DS990 computer.

When configuring the DL parameter values, all values are set to 0.

Enter: DL<CR>

```
Display:                                                     Enter:

DL
  LOAD BIAS                                      = 0000      <CR>
  FORMAT [0=TI, 1=TEK]                           = 0         <CR>
  DESTINATION [0=MEMORY, 1=PROM]                 = 0         <CR>
  PROTOCOL [0=NONE, 1=TEK, 2=ASR, 3=VAX]         = 0         <CR>
  SOURCE [0=HOST, 1=USER]                        = 0         <CR>
?
```

Display: ?

Enter: HOST<CR>

Display: CONTROL-E EXITS
?

Comment: The user has invoked the terminal mode by entering the HOST command.

Enter: <ESC><!>

Display: SYSTEM COMMAND INTERPRETER - PLEASE LOG ON
USER ID: []

Enter: The user enters a valid identification code

Display: PASSCODE: []

Enter: The user enters a valid passcode.

Comment: The emulator is now logged on to the host DS990 system as a terminal. The user may enter any valid DS990 command. The SHOW FILE command will be used to transfer data from the DS990 to the emulator.

Display: []

Enter: SF<CR>

Display: SHOW FILE
FILE PATHNAME: []

Enter: U.TESTFILE<CTRL-V>

Display: []

Comment: The [] prompt indicates that the download is complete.

Enter: Q<CR>

Comment: The <Q> is sent to log off the DS990.

Enter: <CTRL-E>

Display: ?

Comment: CTRL-E exits the terminal mode and returns to the monitor.

Example 6: Downloading object file from DS990 host computer through an 810 printer port.

Port D on the emulator is connected to an 810 Printer Port on the DS990 computer. The 810 printer is an output-only device and therefore the user must enter commands to the DS990 through a terminal attached to the computer.

When configuring the DL parameter values, all values are set to 0 except PROTOCOL which is set to ASR.

Enter: DL<CR>

```
Display:                                                    Enter:

DL
  LOAD BIAS                                    = 0000       <CR>
  FORMAT [0=TI, 1=TEK]                         = 0          <CR>
  DESTINATION [0=NORMAL, 1=PROM]               = 0          <CR>
  PROTOCOL [0=NONE, 1=TEK, 2=ASR, 3=VAX]       = 0          ASR<CR>
  SOURCE [0=HOST, 1=USER]                      = 0          <CR>
?
```

Enter: HOST<CR>

Display: CONTROL-E EXITS

Enter: <CTRL-V>

Comment: The user has invoked the terminal mode on the emulator by entering the HOST command and entered the download-start character.

The log-on sequence is performed on a VDT on the DS990 computer.

Enter: <ESC><!>

Display: SYSTEM COMMAND INTERPRETER - PLEASE LOG ON
USER ID: []

Enter: The user enters a valid identification code

Display: PASSCODE: []

Enter: The user enters a valid passcode.

Comment: The emulator is now logged on to the host DS990 system as a terminal. The user may enter any valid DS990 command. The PRINT FILE (PF) command will be used to transfer data from the DS990 to cassette port 02 which is connected to the XDS unit.

Display: []

Enter: PF<CR>

Display:	Enter:
PRINT FILE - MULTIPLE COPIES	
FILE PATHNAME(S):	U.TESTFILE<CR>
ANSI FORMAT?:	NO<CR>
LISTING DEVICE:	LP01<CR>
DELETE AFTER PRINTING:	NO<CR>
NUMBER OF LINES/PAGE:	<CR>
COPIES:	1<CR>

Note: The NUMBER OF LINES/PAGE must be greater than the length of the object file being transferred. This avoids the problem of extra lines being generated by page breaks.

Comment: When the file has been "printed" to the emulator, use the VDT on the emulator to enter the following.

Enter: <CTRL-W><CTRL-E>

Example 7: Uploading and Downloading object files through port A using a personal computer as the terminal.

The personal computer is connected to port A and must be configured to seven data bits, EVEN parity, and two stop bits. The baud rate is determined by the autobaud procedure.

When executing the UL command, select the USER port as the DESTINATION. The object file is then transmitted through port A to the personal computer and is terminated by the upload-end character.
When executing the DL command, select the USER port as the SOURCE. The emulator now waits for the object file from the user's port.

The UL and DL commands initiate the data transfer. When using port A, the HOST command is NOT used for establishing the communication link.

Communication Trouble Shooting Chart

Symptom	Cause
Unable to communicate with host computer	Baud rate, parity, and/or number of stop bits set by IPORT command are not compatible with host computer parameters. Cables may not be properly connected. Cables may not be connected to correct port on XDS or on the host computer. Switch settings on the communication card may be set incorrectly. Check setting configurations in the XDS Hardware Installation and Operation Manual. The cable lines to pins 2 and 3 may be swapped. The hardware lines are not producing the proper signals (DSR, CTS) from the host.
The host-port-off-line message is displayed.	DSR or CTS lines to XDS are low. Change the switch settings or have host send the proper signals.
An UPLOAD-COMPLETE message occurs while in terminal mode without an upload being performed.	Emulator received an upload-start character from the host on the terminal. Use a different handshake or use IHC command to define a new set of control characters and the select NONE for the PROTOCOL parameter in the UL and DL commands.
"ERRORS INVALID TAG" is displayed while in terminal mode without a download being performed.	Emulator received a download-start character from either the host or the user's terminal. Use a different handshake or use IHC command to define a new set of control characters and the select NONE for the PROTOCOL parameter in the UL and DL commands.
Object file is not in memory after a download	Check DESTINATION parameter for proper selection. Check all other DL command parameters.

Symptom	Cause
Characters are lost during data transfers.	Does the host respond to XON/XOFF or RTS/CTS? If RTS/CTS are recognized as control signals, are the lines connected?
Emulator exits the terminal mode without a CTRL-E being entered.	Does the terminal (intelligent or personal computer) send a CTRL-E for other control functions? Is one of the dedicated hardware lines, #17 or #25, being used?
Unable to communicate with modem.	Does modem require a signal over line #20 (DSR)? If so, change the switches to drive the line low. Note: The emulator must supply all lines to the modem that a terminal would normally supply.
The IHC command does not appear to work correctly	The UL or DL command must be executed after the IHC command is executed in order for the new IHC values to take effect.

SECTION 6

MULTI-PROCESSING

6.1 INTRODUCTION

This section discusses the operation of the XDS system when two or more emulators are connected in a daisy chain and controlled by one user's terminal. This allows emulation of as many as nine processors in a multi-processing environment. The block diagram in Figure 6-1 illustrates a typical three-emulator configuration. A logging device may be connected to unit #1 to log multi-processing development sessions. The multi-processing system may be connected to a host computer via the RS-232-C interface on Port D of the last emulator in the chain. Refer to the appropiate XDS Installation and Operation Guide for further details regarding hardware connections for the multi-processing mode of operation.

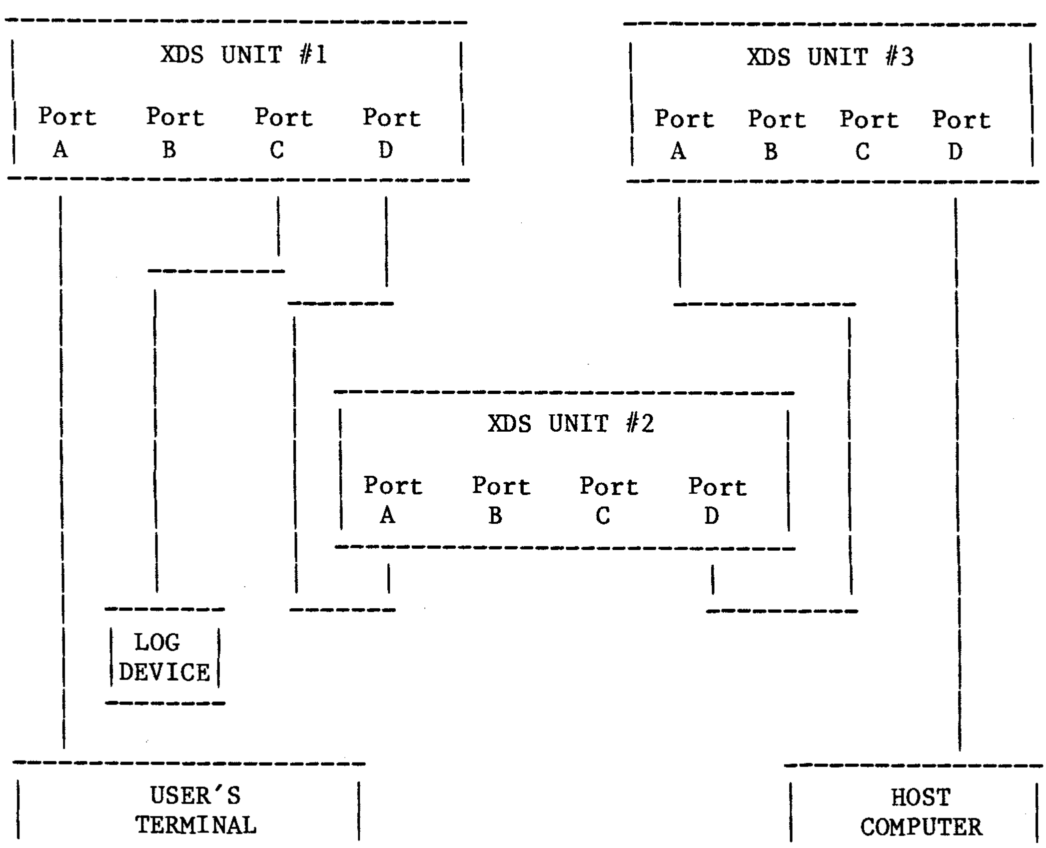


FIGURE 6-1. THREE XDS UNITS IN MULTI-PROCESSING CHAIN

Several concepts must be explained in order to understand the operation of the XDS system in the multi-processing mode. These concepts are presented in the following sections.

6.1.1 TMS9995 Emulator Placement

When the TMS9995 Emulator is used in a daisy chain (as depicted in Figure 6-1) it should be placed as close as possible to the host side of the chain. The TMS9995 Emulator is a single-processor emulator, which means that the same processor must handle all EIA emulator/system communication along with in-circuit emulation. In order to pass communication data from Port A to Port D, the TMS9995 Emulator is required to suspend in-circuit emulation, service the EIA communication, and then return to in-circuit emulation. Placing the emulator closest to the host computer in the daisy chain minimizes the interruptions of in-circuit emulation.

6.2 MULTI-PROCESSING MODE

6.2.1 Initializing the Multi-processing Mode

The multi-processing mode is initialized using the IMP (Initialize Multi-processing Mode) command. When starting cold the following sequence is used to boot up the system and initialize the multi-processing mode.

Power-on

Enter: <CR><CR>

Comment: The two <CR>'s initiate communication between the user's terminal and the XDS unit directly attached to the terminal. The autobaud sequence automatically synchronizes the baud rate of the first XDS unit to that of the user's terminal. For optimum operation a terminal with a baud rate of 9,600 bps is recommended.

Display:

```
                         TEXAS INSTRUMENTS

              TMS9995       XDS       VERSION 1.3.0

COMMANDS:

  INIT      IM      DR      RUN     BP      TR      HOST    IMP
  IPORT     DM      MR      CRUN    BPM     TRM     IHC     IMD
  IPRM      MM      DIO     SS      BPIO    TRIO    UL      ID
  ICC               MIO     SRR             TRIX    DL      BGND
  RCC                                       SOR
  RESTART

  MAP       FILL    XA      DPS     SSB     IT      LOG     GRUN
            FIND    XRA     DHS     DSB     DT      SNAP    TRUN
            DW              DTS     CSB             HELP    GHALT
                                    CASB            DV      THALT

VARIABLES:
  PC        R       LGT     C       INTM
  ST                AGT     OV
  WP                EQ      OP

?
```

Enter: IMP<CR>

Display: AUTOPOLL [0=NO, 1=YES] =0 []

Enter: <CR>

Comment: Autopolling (discussed later in this section) is a special feature which provides status information when an emulator stops.

If a display of status information is desired when the emulator halts operation, the IMP command requires the user to enable AUTOPOLL.

Assuming a three-emulator system, the terminal displays the identification number of the last emulator in the chain followed by the banner for the first unit and the "?" prompt for input. Control remains with the first unit at this point.

Display:

```
LAST EMULATOR = 3
TMS9995  XDS VERSION 1.3.0
?
```

When the IMP command is executed, each emulator in the chain is automatically assigned an identification number (ID #). The XDS unit attached to the user's terminal is assigned ID #1. Unit #1 attempts to establish communication by sending a special character to the next unit in the chain. When this second unit responds with the correct character sequence, it is assigned the next identification number (ID #2). Unit #2 attempts communication with the next unit in the chain; when the proper response is received, the next unit is assigned its ID number. This process is repeated until the last unit in the chain fails to receive the proper response from the host (or no response, if the multi-processing chain is not connected to a host).

When the last unit fails to receive the proper response, it sends its ID # to the terminal. This ID # is displayed as the LAST EMULATOR = ID #, along with the banner from unit #1 as illustrated above. This display of the last emulator's ID # gives the user the opportunity to determine whether all units in the chain are properly connected. If there are more units in the chain than indicated by the ID # of the last emulator, all cable connections and communication card switch settings should be checked. If the IMP command fails, all emulators should be reset and IMP reexecuted.

NOTE

Do not connect more than nine XDS Units in the multi-processing chain.

After the IMP command is executed, the emulators in the chain are assigned their identification numbers and are initialized for multi-processing. Unit #1 is left in contact with the user's terminal and is ON-LINE in what is called FOREGROUND mode. Only one unit at a time can operate in foreground. The remaining units are OFF-LINE or in BACKGROUND mode. These concepts are discussed more completely below.

6.2.2 Initializing Communications with Logging Device and Host Computer

Once the multi-processing mode is booted, the system must be configured for communication with the logging device (i.e., printer) and the host computer system. Although this may be done any time during the multi-processing session, it is preferable at the beginning of the session. It is somewhat frustrating to attempt a download only to find that the communication port is not properly configured.

The IPORT command is used to configure the communication ports just as in the single-processor mode. However, when the IPORT command is executed during the multi-processing mode, only Port C (on unit #1) or Port D (on the last unit) is initialized. Port C on unit #1 is used for the logging device, and Port D on the last unit is the communication link with the host system. It makes no difference which unit is in foreground when IPORT is executed; only these two ports are initialized.

The HOST command invokes the terminal mode through PORT D of the last unit to the host computer system. This establishes communications with the host for uploading from or downloading to specified emulator units. See Section 5, Communications, for more information regarding these commands.

6.2.3 Resetting the Emulators

Two kinds of reset conditions may be selected for a multi-processing configuration: global and local reset. A global reset occurs in all emulators reset when one global-reset emulator is RESET. All global-reset emulators are reset when all of the units (both global and local) are removed from the multi-processing mode.

The local-reset configuration allows only the local-reset unit to respond to its RESET button. The IMP command must be executed if the ID number of the locally-reset emulator is greater than the ID number of the emulator currently in foreground.

To reenter the multi-processing mode after a global reset or a local reset under the above described condition, the IMP command must be reexecuted. However, it is not ne essary to reconfigure the communication ports, nor to reexecute the IMD command.

The emulator is configured for global or local reset with a DIP switch setting for the XDS Model 11, or by a jumper for the XDS Model 22. The appropiate installation and operation guide should be consulted for te location and description of the switch settings and jumper placement. Table 6-1 summarizes these settings for the local and global reset conditions.

RESET CONFIGURATION	COMMUNICATION/MEMORY EXPANSION BOARD SETTINGS	
	XDS MODEL 11 SWITCH	XDS MODEL 22 JUMPER
GLOBAL	S1 - 8 ON	E9 - E10
LOCAL	S1 - 8 OFF	E10 - E11

6.2.4 Establishing Individual Processing Modes

After ID numbers have been assigned to all the emulators, the user may communicate with a specific unit by entering a "#" symbol followed by the unit's ID number. Thus, entering #3<CR> connects the user's terminal to the third unit in the chain. When the terminal is in direct contact with a specified unit, the unit is operating in FOREGROUND. When a unit is in foreground, it behaves much the same as a single processor. The monitor accepts commands just as if the terminal were directly connected to only that processor. This provides a means for loading test data or object code into memory and for setting up specific test conditions via monitor commands and their parameters.

When the #3<CR> is entered immediately after executing the IMP command, the terminal displays the third unit's ID # and the "?" input prompt as follows.

```
Display:  ?

Enter:    #3<CR>

Display:  #3
          ?
```

6.2.5 Initializing the Processing Mode for Individual Units

Once the multi-processing mode is properly initialized using the IMP command, the IMD (Initialize Mode) command may be used to configure the individual units. The IMD command is executed only if the user desires different parameter values (as shown below). If these values are appropriate, then the user need not execute IMD.

```
Enter:    IMD!<CR>

Display:  IMD!
            PROCESSOR MODE [0=IND, 1=MASTER, 2=SLAVE] = 0
            START SYNCHRONOUS [0=NO, 1=YES]           = 0
```

Each unit to be changed is called into foreground with the #n directive before the IMD command is issued. When the specified unit is in foreground, the IMD command is entered, and new parameter values are assigned, as demonstrated in the following illustration.

```
Display:  ?

Enter:    #3<CR>

Display:  #3
          ?

Comment:  Unit #3 is now in foreground and may be reconfigured as a
          master to START SYNCHRONOUSLY in the following manner.

Enter:    IMD<CR>

Display:    PROCESSOR MODE [0=IND, 1=MASTER, 2=SLAVE]  = 0 []

Enter:    MASTER<CR>
```

Display: START SYNCHRONOUS [0=NO, 1=YES] = 0 []

Enter: Y<CR>

The PROCESSOR MODE parameter differentiates among three states: INDEPENDENT, MASTER, or SLAVE. Any unit may be assigned any one of these states. Thus, all units may be independents, all may be slaves, or all may be masters. The master-slave relationship provides a means for halting units synchronously. When any master or slave stops, all slave units stop. Master and independent units are not affected by other emulators that stop.

The second parameter (START SYNCHRONOUS) introduces the concept of synchronized running. If two or more units are directed to run synchronously by virtue of a YES answer to the START SYNCHRONOUS prompt, they become a special subset of the multi-processors in the chain. The units in this subset are connected by the EIA line on Pin 17 of the RS-232-C connector (RUN/HALT line) through an open collector gate. This gives each emulator the ability to drive the line low or to release the line. The last emulator in the multi-processing chain has a pull-up resistor connected to the line. Thus, the line goes high when released by all emulators.

6.2.6 The RUN/HALT Line

The line which controls the synchronized RUN/HALT status of XDS units in a multi-processing chain is called the RUN/HALT line. When this line is HIGH, the emulator can RUN; if the line is LOW, the emulator cannot run. If an emulator is configured as "START SYNCHRONOUS" and is given the command to RUN, it will begin running only after this line goes high.

Any master or slave emulator configured as START SYNCHRONOUS holds the line LOW until told to RUN. However, as long as the line remains LOW, the emulator cannot start running. Thus, until all emulators configured as START SYNCHRONOUS are ordered to RUN, none of the master or slave units can begin its run.

The RUN/HALT line may assume any of the following three states.

* LOW
* RELEASED
* HIGH

When the line is HIGH, emulators configured as START SYNCHRONOUS begin to run if ordered to RUN. When this line is LOW, the emulator assumes the "WAITING TO RUN" mode and waits for the line to go HIGH before starting its run.

The line cannot go HIGH until it is released by all emulators. Thus, its release by the last emulator in the multi-processing chain allows the line to go HIGH. Table 6-1 summarizes the effect of the PROCESSING MODE and START SYNCHRONOUS parameters on the emulators in the multi-processing chain.

An emulator is "connected" to the RUN/HALT line through the IMD command. The term connected means that when the emulator is not running, it drives the line low. When the emulator is running it releases the line. All master and slave units (but no independents) are connected to the line.

Each emulator is able to read the line and may allow the line to halt its run mode when the line is in a transition from high to low. This feature is known as the RUN/HALT mechanism. A hardware circuit samples and holds the state of the RUN/HALT line when the emulator stops. If the line is high, the emulator knows that it is the first unit to stop and (if AUTOPOLLING is enabled) goes into foreground and displays status information.

As indicated earlier, all slaves have the RUN/HALT mechanism activated; so, a slave can run only if all slaves and masters are running (i.e., the RUN/HALT line must be high). All other units will continue running despite the state of the line ONCE THEY HAVE STARTED RUNNING.

TABLE 6-2. MULTI-PROCESSING STATUS AFTER EXECUTING THE IMD COMMAND

PROCESSOR MODE	START SYNCH	RUNS ON GRUN	CONNECTED TO LINE	HALTS ON TRANSITION TO LOW	WAITS TO START ON HIGH
MASTER	NO	YES	YES	NO	NO
MASTER	YES	YES	YES	NO	YES
SLAVE	NO	YES	YES	YES	YES
SLAVE	YES	YES	YES	YES	YES
INDEP	NO	NO	NO	NO	NO
INDEP	YES	YES	NO	NO	YES

After executing the IMP command, any XDS unit may be addressed by entering the #n directive. The unit that is currently in foreground will return to background mode and unit #n will assume foreground operation. When all units in the chain have been initialized by the IMD command, and each unit has the program and test conditions specified, emulation of the multi-processing environment may begin.

6.2.7 Processor RUN/HALT Operations

An emulator is RUNNING if it has been issued a RUN command and is executing a program stored in program memory. The RUN command may take three forms.

* RUN — A simple RUN command issued to each emulator individually.

* GRUN — The Group RUN command initiates a synchronous RUN of the members in the group (i.e., masters, slaves, and independents configured as START SYNCHRONOUS).

* TRUN — The Total RUN command is the same as a RUN command to each individual emulator. The members of the group will start synchronously while those not in the group start RUNNING immediately.

There are several conditions which may cause an emulator to halt execution.

* It is called into foreground by the #n command.

* An error condition arises.

* A breakpoint or trace-buffer full interruption occurs.

* A GHALT (Group HALT) command halts execution of all members of a group.

* A THALT (Total HALT) command halts execution of all units in the multi-processing chain.

* A HALT signal is generated when a master or slave unit stops.

6.2.8 The Group

A group is defined as all XDS units that are designated as masters, slaves, or independents configured as START SYNCHRONOUS. The members of the group may be in any of the three PROCESSOR MODE states: independent, master, or slave. The START SYNCHRONOUS condition operates independently of the PROCESSOR MODE, with the exception of slave units, which are always START SYNCHRONOUS (even if the user enters NO for the START SYNCHRONOUS prompt).

The individual group members configured as START SYNCHRONOUS can start running if, and only if, all other members configured as START SYNCHRONOUS are running. A unit that is ordered to run will wait in the background mode until it receives the signal (i.e., the EIA #17 RUN/HALT line goes HIGH) that all other group members configured as START SYNCHRONOUS are running.

The conditions that start a group running are as follows.

* The user issues a RUN command to each unit

  - START SYNCHRONOUS units begin running after the final unit receives the RUN.

  - Since independent units are not connected to the line, they cannot delay other units by driving the line low.

  - START SYNCHRONOUS independent units will wait until the line is released before starting to run.

  - The last non-independent to receive the RUN command releases the RUN/HALT line, which then goes high.

  - When the RUN/HALT line finally goes high, all units in the "WAITING TO RUN" mode begin to run.

* Issuing a GRUN comand

  - All START SYNCHRONOUS masters, all slaves, and START SYNCHRONOUS independents begin running synchronously when the line goes high.

  - Independents that not configured as START SYNCHRONOUS do not respond to a GRUN comand.

6.2.9 Background Operation

When a RUN, GRUN, or TRUN command is issued, the emulators affected by the command go into the BACKGROUND mode of operation. In this mode, the emulator is effectively disconnected from the multi-processor chain. This means that the emulator appears to be inactive, even though there may be a program running. The unit assumes an OFF-LINE condition and is unable to accept input or to produce output.

Entering the #n directive (where n is the identification number of a specified unit) brings unit #n into foreground and halts program execution. This causes the unit to return to the command entry mode. This in turn affects any other unit that shares a master-slave relationship with the unit brought into foreground.

When all units are operating in background, the terminal displays the following message.

Display: BACKGROUND or AUTOPOLLING
?? ???

Comment: The actual display depends on the AUTOPOLL parameter which is defined when the IMP command is executed.

When an emulator in stand-alone mode stops running, it immediately displays some status information. But emulators in the multi-processing mode may not be able to display any information because they may not be in foreground. Therefore, the AUTOPOLL feature provides a priority scheme for displaying appropriate information when a halt condition arises.

It may be important to know which unit in the group stopped first. A polling sequence in the network can provide this information. When AUTOPOLL is enabled, this sequence alternately polls the emulators for a halt condition, and the keyboard for character input. If the polling sequence detects a halt condition, the emulator that caused the halt switches to foreground. When AUTOPOLLING followed by "???" appears on the terminal screen, the emulator that caused the halt will report because it has the pertinent information.

When all units are in background, the only acceptable input from the keyboard is #n<CR>, /n<CR>, or <ESC>. All other characters are ignored and will not be echoed to the terminal screen. The #n entry brings unit #n into foreground to accept further keyboard input. Entering /n requests unit #n to display its current status. Typing <ESC> returns the last unit which went into background to foreground operation.

6.2.10 Requesting Status

When /n is entered, unit #n will respond to the terminal with a message which reports the current status of the unit. These messages and their interpretations are as follows:

* READY FOR INPUT — The monitor in Unit #n has entered the input phase and waits for commands to be entered into the input buffer.

* WAITING TO RUN — The emulator is in background and is waiting to begin running. This could occur if the BGND command were issued and no RUN command has been executed or if the unit were in a group and waiting for others in the group to begin their runs.

* RUNNING — The emulator is currently executing a program.

* WAITING TO OUTPUT — The emulator is still in background and is waiting to link with an I/O Port for output.

* RUN MODE COMPLETE — The emulator is in a group and is waiting for all of the others in the group to halt.

6.2.11 Flowchart Summaries of Processing Modes

The flowcharts in Figures 6-2, 6-3, and 6-4 summarize the relationships among the master, slave, and independent multi-processors. Figure 6-5 illustrates the emulator response to a halt condition.

```
        -------------------
        |       RUN       |                  1
        -------------------
                 |
                 V
        -------------------
        | Change Status to |                 2
        | "WAITING TO RUN" |
        -------------------
                 |
                 V
        -------------------
        |   Release R/H   |                  3
        |      Line       |
        -------------------
                 |
                 V
                * *
              *     *
            *  Start  *    No
          * Synchronous *--------------+     4
            *    ?    *                |
              *     *                  |
                * *                    |
           Yes   |                     |
                 |<-------------+      |
                 V              |      |
                * *             |      |
              *     *           |      |
            * RUN/HALT*    No   |      |     5
          *  Line=HIGH  *-------+      |
            *    ?    *                |
              *     *                  |
                * *                    |
           Yes   |                     |
                 |<--------------------+
                 V
        -------------------
        |  Change Status  |                  6
        |       to        |
        |    "RUNNING"    |
        -------------------
                 |
                 V
        -------------------
        |  Goto RUN mode  |                  7
        -------------------
                 |
                 V
        -------------------
        |       END       |
        -------------------
```

FIGURE 6-2. FLOWCHART FOR MASTER UNIT AFTER RECEIVING A RUN, TRUN, OR GRUN COMMAND

The following describes the components of the flowchart in Figure 6-2. The numbers refer to those on the right of the flowchart symbols and describe specific elements of the chart.

1. When a master receives a command to RUN, the RUN/HALT line is low. This is always the case because a unit automatically holds the line LOW when it is designated as a master in the IMD command.

2. The emulator goes into the "WAITING TO RUN" mode.

3. When the master unit goes into WAITING TO RUN mode, it releases the RUN/HALT line.

4. If the emulator is configured as START SYNCHRONOUS, it waits for the line to go high before entering the RUNNING mode.

5. The HALT mechanism is enabled and the emulator will respond to the state of the R/H line.

6. The emulator changes its status to RUNNING and proceeds to the RUN mode.

7. The emulator is not halted by the RUN/HALT line going low. It can be halted by entering the #n command (where n = the emulator's ID #).

```
 ------------------
|       RUN        |              1
 ------------------
         |
         V
 ------------------
| Change Status to |              2
| "WAITING TO RUN" |
 ------------------
         |
         V
 ------------------
|   Release R/H    |              3
|      Line        |
 ------------------
         |<------------+
         V             |
        * *            |
      *     *          |
    *RUN/HALT *   No   |          4
   *  Line=HIGH  *-----+
    *    ?     *
      *     *
        * *
         |
         V
 ------------------
|  Enable Halt     |              5
|  Mechanism       |
 ------------------
         |
         V
 ------------------
|  Change Status   |
|       to         |              6
|   "RUNNING"      |
 ------------------
         |
         V
 ------------------
|  Goto RUN mode   |              7
 ------------------
         |
         V
 ------------------
|       END        |
 ------------------
```

FIGURE 6-3. FLOWCHART FOR SLAVE UNIT AFTER RECEIVING A RUN, TRUN, OR GRUN COMMAND

The following describes the components of the flowchart in Figure 6-3. The numbers refer to those on the right of the flowchart symbols and describe specific elements of the chart.

1. When a slave receives a command to RUN, the RUN/HALT line is low. This is always the case because a unit automatically holds the line LOW when it is designated as a slave during execution of the IMD command.

2. The emulator goes into the "WAITING TO RUN" mode.

3. When the slave unit goes into WAITING TO RUN mode, it releases the line. If the emulator is the last to get a RUN command, the line will go high and the unit will start running. If it is not the last, then it waits until the line goes high to start its run.

4. A slave is always configured as START SYNCHRONOUS and therefore waits for the line to go high before going into the RUNNING mode.

5. Slave units enable the HALT mechanism at this point so that they can force the line to go LOW when they halt execution.

6. The emulator changes its status to RUNNING and goes into the RUN mode.

7. All slaves are halted by the RUN/HALT line going low. Thus, for a slave to be running, the RUN/HALT line must be high. All non-slave units continue to run regardless of the state of the RUN/HALT line ONCE THEY HAVE STARTED RUNNING.

```
        -------------------
        |       RUN       |                    1
        -------------------
                 |
                 V
        -------------------
        | Change Status to |                   2
        | "WAITING TO RUN" |
        -------------------
                 |
                 V
                * *
              *     *     No
            *  Start  *
          * Synchronous *---------------+      3
            *    ?    *                 |
              *     *                   |
                * *                     |
                 |                      |
                 |<-------------+       |
                 V              |       |
                * *             |       |
              *     *           |       |
            * RUN/HALT*    No   |       |
          *  Line=HIGH  *-------+       |      4
            *    ?    *                 |
              *     *                   |
                * *                     |
                 |<---------------------+
                 V
        --------------------
        |  Change Status   |
        |       to         |                   5
        |   "RUNNING"      |
        --------------------
                 |
                 V
        --------------------
        |  Goto RUN mode   |                   6
        --------------------
                 |
                 V
        --------------------
        |       END        |
        --------------------
```

FIGURE 6-4. FLOWCHART FOR INDEPENDENT UNIT AFTER RECEIVING A RUN, TRUN, OR GRUN COMMAND

The following describes the components of the flowchart in Figure 6-4. The numbers refer to those on the right of the flowchart symbols and describe specific elements of the chart.

1. When an independent receives a command to RUN, the RUN/HALT line may be either high or low. An independent has no control over the line and causes no change on the line.

2. The emulator goes into the "WAITING TO RUN" mode.

3. If the emulator is not configured as START SYNCHRONOUS, it goes directly into the RUNNING mode.

4. If the emulator is configured as START SYNCHRONOUS, it waits for the line to go high before going into the RUNNING mode.

5. The emulator changes its status to RUNNING and goes to the RUN mode.

6. The emulator is not halted by the RUN/HALT line going low. It can be halted by entering the #n command (where n = the emulator's ID#).

```
 ----------------------                            
 |     Execution      |                               1
 |      Halted        |
 ----------------------
           |
           V
          * *
        *     *
      *RUN/HALT *     No
    *  Line=HIGH  *-----------------+
      *   on     *                  |                 2
        *halt?*                     |
          * *                       |
           |  Yes                   V
           |                ----------------------    3
           |                | Change status to   |
           |                | "WAITING TO OUTPUT" |
           V                ----------------------
 -----------------------                |
 |   Change Status to  |                |             4
 | "RUN MODE COMPLETE" |                |
 -----------------------                |
           |                            |
           V                            |
          * *                           |
        *     *                         |
      *         *     No                |
    * AUTOPOLLING *----------+          |             5
      *    ?    *            |          |
        *     *              |          |
          * *                |          |
       Yes |                 |          |
           V                 |          |
 -----------------------     |          |
 | Return To FOREGROUND |    |          |             6
 -----------------------     |          |
           |                 V          V
           |<----------------+----------+
           V
 -----------------------
 |         END         |
 -----------------------
```

FIGURE 6-5. EMULATOR STATUS AFTER EXECUTION IS HALTED

The following describes the components of the flowchart in Figure 6-5. The numbers refer to those on the right of the flowchart symbols and describe the specific elements of the chart.

1. When an emulator stops for any reason, slaves and masters always drive the R/H line low. Independents do not since they are not "connected" to the line.

2. Only a slave or master emulator that stops first will detect the line as HIGH when it stops. Independents don't care about the state of the line and react as though the line were HIGH.

3. Emulators that stop after the line goes low will change their status to "WAITING TO OUTPUT" and will display their output when they return to foreground.

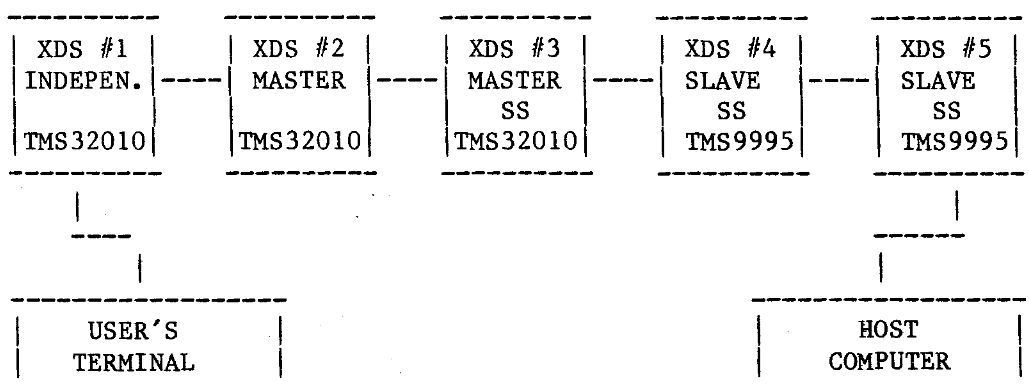
4. The first emulator to stop will change the operating status to "RUN MODE COMPLETE" and test for AUTOPOLLING mode.

5. If AUTOPOLLING is enabled, the first emulator to stop will return to foreground. If AUTOPOLLING is not enabled, the first emulator to stop will wait until recalled to foreground before displaying the appropriate status information.

6. When the emulator reenters foreground, status information is displayed, and the emulator returns control to the monitor which displays the "READY FOR INPUT" question mark prompt.

The TRUN and THALT commands affect all of the processors in the chain. The TRUN (Total RUN) command starts all emulators in the multi-processor chain. Those emulators in the group configured as START SYNCHRONOUS will do so, and the remaining processors will start when they are ready. The THALT (Total HALT) command halts all emulators in the multi-processor chain.

The GRUN and GHALT commands affect only group members including masters, slaves, and independents that are configured as START SYNCHRONOUS. Again, those units that are configured as START SYNCHRONOUS will do so, while masters that are not configured as START SYNCHRONOUS will begin running when they are told.



SS = Start Synchronous

FIGURE 6-6. MULTI-PROCESSING USING FIVE XDS UNITS

6.3 EXAMPLES

For convenience, the sample sessions in the following examples are presented in a "walk-through" format (i.e., if the user is required to enter data through the keyboard, the data is preceded by "Enter:"; the displays appearing on the VDT screen are preceded by "Display:"; and explanations of entries, displays, command functions, etc. are preceded by "Comment:"). Two examples of multi-processing are presented:

Example 1: configuring a five-processor chain.

Example 2: sample session using a three-processor chain.

6.3.1 Configuring a Five-Processor Chain

Figure 6-6 is a diagramatic model of five emulators that have been configured using the following series of steps. In order to simplify explanation, and to show the responses of the TMS9995 Emulator, this example uses five TMS9995 Emulators; however, emulators from different microprocessor families may be used (as represented in figure 6-6).

POWER-ON

Enter: <CR><CR>

Display:

```
                          TEXAS INSTRUMENTS

              TMS9995      XDS      VERSION 1.3.0

COMMANDS:

  INIT      IM     DR     RUN    BP     TR     HOST   IMP
  IPORT     DM     MR     CRUN   BPM    TRM    IHC    IMD
  IPRM      MM     DIO    SS     BPIO   TRIO   UL     ID
  ICC              MIO    SRR           TRIX   DL     BGND
  RCC                                   SOR
  RESTART

  MAP       FILL   XA     DPS    SSB    IT     LOG    GRUN
            FIND   XRA    DHS    DSB    DT     SNAP   TRUN
            DW            DTS    CSB           HELP   GHALT
                                 CASB          DV     THALT

VARIABLES:
  PC        R      LGT    C        INTM
  ST               AGT    OV
  WP               EQ     OP

?
```

Enter: IMP<CR>

Display: AUTOPOLL [0=NO, 1=YES] = 0 []

Enter: <CR>

Display:

```
LAST EMULATOR = 5
TMS9995  XDS VERSION 1.3.0
?
```

Enter: IPORT<CR>

Comment: The port must be configured by entering the IPORT command.

Display: PORT [0=HOST ("D"), 1=LOG/PROM ("C")] = 0 []

Enter: <CR>

Comment: The user has selected Port D for configuration. Only Port D in the last emulator in the chain will be configured.

Display: BAUD[19.2K, 9.6K, 4.8K, 2.4K, 1.2K, 600, 300, 110] = 0 []

Enter: 6<CR>

Comment: The user has entered a new value for the baud rate of 300 bps. The host RS-232-C Port must be set to this same baud rate.

Display: PARITY[0=OFF, 1=ODD, 2=EVEN] = 0 []

Enter: <CR>

Comment: The user has selected the NO PARITY option.

Display: NUMBER OF STOP BITS (0=2, 1=1) = 0 []

Enter: <CR>

Comment: The user has selected the value of two stop bits which must conform to the same value for the host system.

Display: BITS/CHAR (0=7, 1=8) = 0 []

Enter: <CR>

Comment: The user has selected seven data bits per character.

Display: ?

Comment: Port D in unit #5 has now been configured for communication with the host system. Unit #1 remains in foreground and the user may enter any monitor command to unit #1 for execution.

Enter: IMD<CR>

Comment: Entering the IMD command allows the user to initialize the operating mode for the unit that is currently in foreground (unit #1 in this case).

Display: PROCESSOR MODE [0=IND, 1=MASTER, 2=SLAVE] = 0 []

Enter: <CR>

Display: START SYNCHRONOUS [0=NO, 1=YES] = 0 []

Enter: <CR>

Comment: The user has specified that unit #1, which is currently operating in foreground, will act as an independent unit and will not be a member of the group. Program object code may be entered by a download from the host system, from the keyboard via the IM command, or through the Assembler. Monitor commands may be entered into the emulator input buffer(s) for execution at the appropriate time.

Display: ?

Enter: #2<CR>

Comment: The user has called unit #2 into foreground. Unit #1 now goes into background. The user has complete access to unit #2 and may enter any monitor command for execution by unit #2.

Display: #2
?

Enter: IMD<CR>

Display: PROCESSOR MODE [0=IND, 1=MASTER, 2=SLAVE] = 0 []

Enter: 1<CR>

Display: START SYNCHRONOUS [0=NO, 1=YES] = 0 []

Enter: <CR>

Comment: The user has specified that emulator #2, which is currently operating in foreground, will act as a master unit and as such is automatically a member of the group. Program object code may be entered by a download from the host system, from the keyboard via the IM command, or through the Assembler. Monitor commands may be entered into the emulator input buffer(s) for execution at the appropriate time.

Display: ?

Enter: #3<CR>

Comment: Unit #3 is called into foreground.

Display: #3
?

Enter: ID<CR>

Display: #3
TMS9995 XDS VERSION 1.3.0
?

Enter: IMD<CR>

Display: PROCESSOR MODE [0=IND, 1=MASTER, 2=SLAVE] = 0 []

Enter: 1<CR>

Display: START SYNCHRONOUS [0=NO, 1=YES] = 0 []

Enter: 1<CR>

Comment: The user has specified that unit #3, which is currently operating in foreground, will act as a master unit and will start simultaneously with all other units that have the START SYNCHRONOUS parameter enabled. Program object code may be entered by a download from the host system, from the keyboard via the IM command, or through the Assembler. Monitor commands may be entered into the emulator input buffer(s) for execution at the appropriate time.

Display: ?

Enter: #4<CR>

Comment: Unit #4 is called into foreground.

Display: #4
?

Enter: IMD<CR>

Display: PROCESSOR MODE [0=IND, 1=MASTER, 2=SLAVE] = 0 []

Enter: 2<CR>

Display: START SYNCHRONOUS [0=NO, 1=YES] = 0 []

Enter: 1<CR>

Comment: The user has specified that emulator #4 will act as a slave unit and will start simultaneously with all other units that have the START SYNCHRONOUS parameter enabled. Slave units automatically become configured as START SYNCHRONOUS regardless of the response to the START SYNCHRONOUS prompt. Program object code may be entered by a download from the host system, from the keyboard via the IM command, or through the Assembler. Monitor commands may be entered into the emulator input buffer(s) for execution at the appropriate time.

Display: ?

Enter: #5<CR>

Comment: Unit #5 is called into foreground.

Display: #5
?

Enter: IMD<CR>

Display: PROCESSOR MODE [0=IND, 1=MASTER, 2=SLAVE] = 0 []

Enter: 2<CR>

Display: START SYNCHRONOUS [0=NO, 1=YES] = 0 []

Enter: <CR>

Comment: The user has specified that emulator #5 will be a slave unit which makes it a member of the group, configured as START SYNCHRONOUS. Program object code may be entered by a download from the host system, from the keyboard via the IM command, or through the Assembler. Monitor commands may be entered into the emulator input buffer(s) for execution at the appropriate time.

Display: ?

Enter: #1<CR>

Comment: Unit #1 is now called back into foreground.

NOTE

> The diagramatic model shown in Figure 6-6 shows five emulators from different microprocessor famlies operating in the multi-processing mode. The examples and sample sessions in this section pertain to five TMS9995 Emulators in a multi-processing chain in order to show the TMS9995's responses from any given location in the chain.

The model presented in Figure 6-6 illustrates the relationships established by the PROCESSOR MODE and START SYNCHRONOUS parameters. In this example, units #2, #3, #4, and #5 have been assigned to the group. This means that these units will respond to the GRUN and GHALT commands. Units #1 and #2 are able to start running independently of the other units because they are not configured as START SYNCHRONOUS.

If the user issues RUN commands to each unit in sequential order, units #1 and #2 will start running immediately, while units #3, #4 , and #5 will not start running until all of these units are ready to run. Even though unit #3 is a master and #4 is a slave, #3 must wait until #4 is running to start its run.

There are two other conditions that must be met before unit #4 (or any slave unit) can start its run. All master units must either be running or ready to run, and all other slave units must be ready to run before the slave units can begin running.

The master/slave relationship affects the HALT condition, but not the START condition. In this case, if unit #3 stops before #4, it will cause units #4 and #5 to stop. Thus, if any of the master units or slave units stop, all slave units will stop. So, if unit #2 ,#3, #4, or #5 stops then both units #4 and #5 will stop.

If the user issues a GRUN command, units #3, #4, and #5 will start running simultaneously. To get the remaining units running requires that the user issue RUN commands to each unit individually.

6.3.2 Sample Session of Multi-processing Mode

This sample session is an exercise in running a three-processor system.

Display: ?

Enter: IMP<CR>

Display: AUTOPOLL [0=NO, 1=YES] = 0 []

Enter: <CR>

Display: LAST EMU = 3
TMS9995 XDS VERSION 1.3.0

Enter: IMD<CR>

Display: PROCESSOR MODE [0=IND, 1=MASTER, 2=SLAVE] = 0 []

Enter: 1<CR>

Display: START SYNCHRONOUS [0=NO, 1=YES] = 0 []

Enter: 1<CR>

Comment: Unit #1 has been configured as a master on the START SYNCHRONOUS line.

Display: ?

Enter: ID<CR>

Display: #1
TMS9995 XDS VERSION 1.3.0

?

Enter: #2<CR>

Comment: Unit #2 is brought into FOREGROUND and unit #1 is placed into background.

Display: ?

Enter: IMD<CR>

Display: PROCESSOR MODE [0=IND, 1=MASTER, 2=SLAVE] = 0 []

Enter: <CR>

Display: START SYNCHRONOUS [0=NO, 1=YES] = 0 []

Enter: <CR>

Comment: Unit #2 has been configured as an independent.

Display: ?

Enter: #3<CR>

Display: ?

Enter: IMD<CR>

Display: PROCESSOR MODE [0=IND, 1=MASTER, 2=SLAVE] = 0 []

Enter: 2<CR>

Display: START SYNCHRONOUS [0=NO, 1=YES] = 0 []

Enter: <CR>

Comment: Unit #3 has been configured as a slave and is automatically "connected" to the START SYNCHRONOUS line.

Display: ?

Enter: RUN<CR>

Display: BACKGROUND
??

Enter: /1<CR>

Display: EMULATOR READY FOR INPUT
BACKGROUND
??

Enter: /2<CR>

Display: EMULATOR READY FOR INPUT
BACKGROUND
??

Enter: /3<CR>

Display: EMULATOR WAITING TO RUN
BACKGROUND
??

Comment: Unit #3 has been ordered to run but must wait until all slave and master units are also told to run. In this case unit #1, a master unit, has not been told to run, so will prevent any slave from running until it begins its run.

Enter: #2<CR>

Display: ?

Enter: RUN<CR>

Display: BACKGROUND
??

Enter: /1<CR>

Display: EMULATOR READY FOR INPUT
BACKGROUND
??

Enter: /2<CR>

Display: EMULATOR RUNNING
BACKGROUND
??

Enter: /3<CR>

Display: EMULATOR WAITING TO RUN
BACKGROUND
??

Enter: #1<CR>

Display: ?

Enter: RUN<CR>

Display: BACKGROUND
??

Enter: /1<CR>

Display: EMULATOR RUNNING
BACKGROUND
??

Enter: /2<CR>

Display: EMULATOR RUNNING
BACKGROUND
??

Enter: /3<CR>

Display: EMULATOR RUNNING
BACKGROUND
??

Enter: #2<CR>

Comment: The #2 directive recalls unit #2 into FOREGROUND. Program execution is halted.

Display: KEY
WP=____ PC=____ ST=____

Enter: BGND<CR>

Display: BACKGROUND
??

Enter: /1<CR>

Display: EMULATOR RUNNING
BACKGROUND
??

Enter: /2<CR>

Display: EMULATOR READY FOR INPUT
BACKGROUND
??

Enter: /3<CR>

Display: EMULATOR RUNNING
BACKGROUND
??

Enter: #3<CR>

Display: KEY
WP=____ PC=____ ST=____

Enter: BGND<CR>

Display: BACKGROUND
??

Enter: /1<CR>

Display: EMULATOR RUNNING
BACKGROUND
??

Enter: /2<CR>

Display: EMULATOR READY FOR INPUT
BACKGROUND
??

Enter: /3<CR>

Display: EMULATOR READY FOR INPUT
BACKGROUND
??

Enter: #1<CR>

Display: KEY
WP=____ PC=____ ST=____

Enter: GRUN<CR>

Display: BACKGROUND
??

Enter: /1<CR>

Display: EMULATOR RUNNING
BACKGROUND
??

Enter: /2<CR>

Display: EMULATOR READY FOR INPUT
BACKGROUND
??

Enter: /3<CR>

Display: EMULATOR RUNNING
BACKGROUND
??

Comment: GRUN has started only those in the group (i.e., Masters, slaves, and independents configured to run synchronously).

Enter: #1<CR>

Display: KEY
WP=____ PC=____ ST=____

Enter: #2<CR>

Display: KEY
WP=____ PC=____ ST=____

Enter: #3<CR>

Display: KEY
WP=____ PC=____ ST=____

Enter: TRUN<CR>

Display: BACKGROUND
??

Enter: /1<CR>

Display: EMULATOR RUNNING
BACKGROUND
??

Enter: /2<CR>

Display: EMULATOR RUNNING
BACKGROUND
??

Enter: /3<CR>

Display: EMULATOR RUNNING
BACKGROUND
??

Comment: TRUN starts all emulators.

SECTION 7

BREAKPOINT/TRACE FUNCTION

7.1 INTRODUCTION

The breakpoint-trace board must be installed in the XDS chassis in order to utilize the features discussed in this section. When this board is installed, the performance of the emulator is increased significantly. The user is able to set up breakpoints on any or all memory access operations (i.e., read, write, and instruction acquisition) as well as any or all I/O operations (read and write). Refer to the XDS Breakpoint-Trace Module Installation Guide for details on the installation of this board.

Another useful feature of the breakpoint-trace board is the ability to trace execution of a program under controlled test conditions. When the trace mode is invoked, information regarding opcodes, addresses, and data in memory is stored as a series of trace samples. These samples are then available for display at the discretion of the user.

Breakpoint/trace functions utilize masking and extended data and address probes. The user should refer to Appendix B for a discussion of masking and how the XDS system uses masks. Appendix C provides detailed information on extended data and address probes using the EXTENDED TRACE CABLE.

This section provides information regarding hardware breakpoint functions using the BP (Breakpoint), BPM (Breakpoint Memory), and BPIO (Breakpoint Input/Output) commands and their parameters. The trace function is explained in terms of the TR (Trace), TRM (Trace Memory), TRIO (Trace Input/Output), and TRIX (Trace Instruction Acquisition Extended) commands and their parameters.

Sample sessions illustrate the trace and breakpoint operations and interactions. Special programs demonstrate the TRIX command and its specialized function.

The Breakpoint and Trace commands are used to set up the conditions to halt execution (breakpoint) and/or to trace the execution. These two functions can be independently controlled by the user. There are also features that allow the breakpoint and trace functions to interact.

Breakpoint and trace criteria are defined by the user in terms of the following microprocessor functions:

Memory Cycles - Reads, writes, instruction acquisition, and/or the data on the data bus during these cycles.

I/O Cycles - Reads and writes.

Each cycle can also be qualified by: (1) an address, a pair of addresses, or an address range; and/or (2) eight external data probes which can be connected to any standard TTL source.

7.2 BREAKPOINTS

A breakpoint is a hardware interrupt which halts execution of the user's program and returns emulator control to the keyboard (or host computer) via the monitor program. Hardware breakpoint conditions are defined by the user prior to using any RUN command.

NOTE

When a single emulator is used, the RUN command initiates execution of the program in memory. When in the multi-processing mode, program execution may also be initiated using TRUN (Total RUN) or GRUN (Group RUN). Future use of the RUN command will imply that TRUN and GRUN may also be used to start program execution if the unit is in a multi-processing mode of operation.

As a program is executed, the breakpoint-trace board tracks the conditions of execution and compares these to the hardware breakpoint criteria specified by the user. A breakpoint event is recorded each time the breakpoint criteria and execution conditions match. The user also specifies the number of events to be recorded before program execution is halted. When all conditions defined by the user are satisfied, program execution is halted and control of the emulator is returned to the monitor.

7.2.1 Initializing Event and Delay Counts

Breakpoint event counts, delay counts, and address masks are initialized by executing the BP (Breakpoint) command (it is not used, however, to define the criteria for a breakpoint). If the command is entered so that the parameters are displayed using the exclamation point terminator, the display appears as follows.

```
BP!<CR>
  EVENT COUNT (0 - 7FFF)      = 0000
  DELAY COUNT (0 -  7FF)      = 000
  ADDRESS MASK (ONES ENABLE) = FFFF
```

The EVENT COUNT parameter of the BP command allows the user to define the number of breakpoint events recorded before an actual breakpoint occurs.

The DELAY COUNT parameter provides additional delay capability. When the user specifies a delay count, the TRACE function is used to decrement the delay count.This requires that one of the trace commands be executed so that trace samples can be taken for the delay count. Delay counting does not begin until the EVENT COUNT reaches zero. After the specified number of delays (trace samples) are recorded, program execution is finally halted. The DELAY COUNT allows the TRACE MEMORY to be "windowed" around the event that causes execution to halt.

When a program is executed, the RUN command resets the breakpoint and trace circuits to operate with all the parameters specified in the breakpoint and trace commands. This includes clearing all trace samples previously taken and events counted. The SS (Single-Step) and CRUN (Continue RUN) commands, however, perform execution without losing previous breakpoint and trace history as long as the event, delay, or trace counts are not changed. All other breakpoint and trace parameters may be changed without losing this history.

When a breakpoint or trace parameter is changed, excluding one or more of the counts mentioned above, a pause will be observed on the first execution of the SS or CRUN command. This pause is the time it takes for the parameter change(s) to be incorporated into the breakpoint and trace circuits.

The ADDRESS MASK parameter allows regions of memory to be specified as part of the breakpoint condition. (Refer to Appendix B for details concerning the concept of data and address masking.)

To define the criteria for a breakpoint requires the use of the BPM (Breakpoint Memory) and the BPIO (Breakpoint I/O) commands. The BPM command sets parameters for the memory access cycle when a program instruction is executed. The BPIO command sets parameters for I/O (CRU - Communication Register Unit) access.

7.2.2 Memory Cycle Breakpoint Events

Entering the BPM command for display using the exclamation point terminator results in the following display.

```
BPM!<CR>
  QUALIFIER [OFF, MA, MR, MW, IAQ]     = 0
  BREAKPOINT ADDRESS #1                = 0000
  BREAKPOINT ADDRESS #2                = 0000
  RANGE INDICATOR [0=NO, 1=YES]        = 0
  EMU DATA COMPARE WORD                = 0000
  EMU DATA COMPARE MASK (ONES ENABLE)  = 0000
  EXT DATA COMPARE BYTE                = 00
  EXT DATA COMPARE MASK (ONES ENABLE)  = 00
```

7.2.2.1 Selecting Memory Qualifiers for Breakpoints

The parameters associated with the BPM command define the criteria for a breakpoint to occur for memory cycle operations. The user qualifies the type of memory cycle operation that will be recorded as an event. These qualifiers are defined as follows.

Qualifier = 0 (OFF) This disables the memory breakpoint. Memory cycle operations are not recorded as breakpoint events.

Qualifier = 1 (MA) This selection indicates that any memory access cycle will be recorded as a breakpoint event. This means that memory read, memory write, and instruction acquisition operations are considered as breakpoint events.

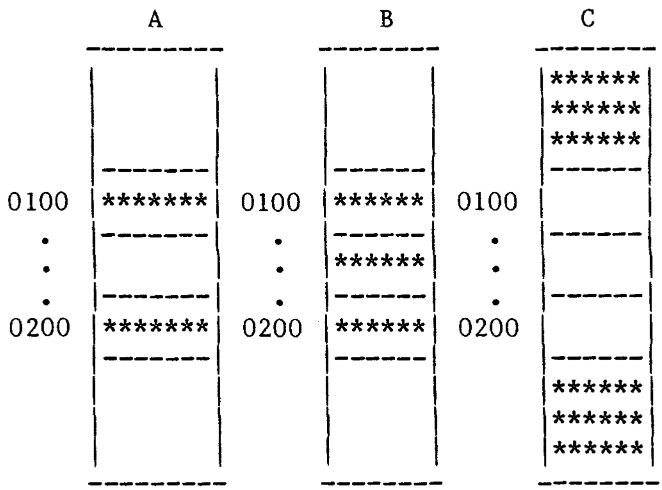
Qualifier = 2 (MR) This value indicates that any memory read or instruction acquisition (which is a form of memory read) operation will be recorded as a breakpoint event.

Qualifier = 3 (MW) This value records a breakpoint event for a memory write cycle operation only.

Qualifier = 4 (IAQ) This selection indicates that an instruction acquisition cycle operation is to be recorded as a breakpoint event.

7.2.2.2 Selecting Addresses for Breakpoint Events.

Figure 7-1 illustrates the use of the ADDRESS and RANGE INDICATOR parameters to specify address locations as breakpoint event criteria.



	A	B	C
ADDRESS #1	0100	0100	0200
ADDRESS #2	0200	0200	0100
RANGE IND.	NO	YES	YES

******* = Included in the range for breakpoint/trace.

FIGURE 7-1. BREAKPOINT/TRACE ADDRESSING

The user must specify the region of the program memory that will be scanned to determine breakpoint events. This region of memory is specified by entering values into the two ADDRESS parameters. These values may be interpreted in three ways.

* If the RANGE INDICATOR parameter is disabled (i.e. set to 0=NO), the qualifying address locations specified in ADDRESS #1 and ADDRESS #2 are considered as single independent locations. In this case, only machine cycle activity at these two address locations will be monitored as possible breakpoint events.

* If the RANGE INDICATOR is ON (i.e., set to 1=YES), the two address locations become the lower and upper boundaries, respectively, for a range of address locations that qualify breakpoint events. In this case, machine cycle activity occurring within these boundaries (including the location of the ADDRESS values) that meet all other specified breakpoint criteria will be recorded as breakpoint events. Any address location outside the specified range will not qualify even though it satisfies all other criteria for a breakpoint event.

* If the value in ADDRESS #2 is less than that in ADDRESS #1, the boundaries exclude the range between the values as eligible breakpoint event locations when the RANGE INDICATOR is on. Thus, all address locations outside the range (excluding the ADDRESS value locations) will be considered as breakpoint events if all other criteria are satisfied. This provides a means of specifying two separate address ranges for breakpoint events.

The RANGE INDICATOR parameter enables (1=YES) or disables (0=NO) the use of the address ranges specified in ADDRESS #1 and ADDRESS #2. If this parameter is OFF, only the two ADDRESS values are used to determine breakpoint events. If the parameter is ON, the address range specified is used. Note that setting the ADDRESS MASK in the BP command may also provide a means for specifying multiple ranges to be used for breakpoints.

NOTE

The monitor checks for the exclusive address range before performing address masking.

7.2.2.3 Setting Data Compare Words and Masks

(Please refer to Appendix B for details concerning data and address masking.) The EMU DATA COMPARE WORD parameter allows the user to specify that a particular data-bus value (i.e., data, instruction, or address) is to be used as another requirement for memory accesses to qualify as breakpoint events. If the EMU DATA MASK is enabled, the user may use special bits in the data compare word as criteria for breakpoint events. By specifying a data mask, each ZERO-bit in the data mask word establishes a "don't care" condition.

If the data word is specified as 001A and the data mask is set to FFF0, all data words that have 001X, (where X is any hexadecimal value between 0 and F), will be recorded as satisfying this requirement for a breakpoint event. If the mask is set to FFFF then 001A is the only value that will satisfy this breakpoint event criterion.

7.2.2.4 Setting Extended Data Compare Bytes and Masks

(Please refer to Appendix B for details concerning data and address masking.) I/O accesses are serial in nature; due to this and other processor characteristics, data comparison for qualifying I/O data must be done using the EXTENDED TRACE CABLE and the Extended Data Compare Bytes and Masks. The EXT DATA COMPARE BYTE refers to the eight data bits on the EXTENDED TRACE CABLE. The value of the extended data compare byte is limited to the number of data bits remaining after the NUMBER OF EXTENDED ADDRESS BITS is set.

The EXT DATA MASK parameter operates in the same way as the EMU DATA MASK discussed above, except that it applies to the extended eight-bit data field. The value of the extended data mask is limited to the number of data bits remaining after the NUMBER OF EXTENDED ADDRESS BITS is set.

7.2.2.5 Summary of Breakpoint Event Criteria

A breakpoint event is recorded when the following Boolean Equation is satisfied.

```
BREAKPOINT EVENT = BREAKPOINT ADDRESS IN RANGE
                               *
                     DATABUS QUALIFIED (see Note below)
                               *
                   BREAKPOINT QUALIFIER SATISFIED
                               *
[EXTENDED DATA QUALIFIED * EXTENDED ADDR. BITS QUALIFIED]
```

Note: Calculation of DATA BUS Qualification: If the following equation is zero then the DATA BUS is qualified.

<DATA BUS> ⊕ (<DATA COMPARE WORD>*<DATA MASK>)

7.2.3 I/O Cycle Breakpoint Events.

The user has the option of specifying a breakpoint event on a memory cycle, an I/O cycle, or both types of cycle. Again, the user specifies the criteria for considering the I/O cycle as a breakpoint event. The process is very similar to the memory operations discussed above, except that there are no emulator data compare or emulator data compare mask actions.

The BPIO command is used to define the parameters for breakpoint events during I/O operations. If BPIO is entered using the exclamation point terminator, the display appears as follows.

```
BPIO!<CR>
  QUALIFIER [OFF, IOA]                  = 0
  BREAKPOINT ADDRESS #1                 = 0000
  BREAKPOINT ADDRESS #2                 = 0000
  RANGE INDICATOR [0=NO, 1=YES]         = 0
  EXT DATA COMPARE BYTE                 = 00
  EXT DATA COMPARE MASK (ONES ENABLE) = 00
```

7.2.3.1 Setting I/O Access Qualifiers

The criteria for establishing a breakpoint event (during an I/O operation to any of the 32K CRU BITS addressable on the TMS9995 Emulator) are as follows.

Qualifier = 0 (OFF) This disables the breakpoint operation during I/O cycles.

Qualifier = 1 (IOA) This defines any read-from- or write-to-CRU as an acceptable criterion for a breakpoint event.

7.2.3.2 Setting I/O Breakpoint Addresses and Ranges

The BREAKPOINT ADDRESS #1 and BREAKPOINT ADDRESS #2 parameters operate the same way as these parameters in the BPM command. However, the addresses specified refer to CRU Addresses rather than memory locations. The maximum value for these parameters is FFFF. If the user specifies ADDRESS #1 as 0 and ADDRESS #2 as FFFF, events that occur on any of the CRU addresses are considered as potential breakpoint events when the RANGE INDICATOR is ON. If the RANGE INDICATOR is OFF then I/O cycles on CRU addresses 0000 and FFFF only will qualify as breakpoint events.

NOTE

Address bit 15 is not used and is a "don't care" in I/O (CRU) addresses.

The remaining parameters for the BPIO command are the same as those for the BPM command discussed above (except for the omission of the emulator data compare word and emulator data compare mask). These parameters must be set in the same way as for memory operations in order to specify conditions for breakpoint events during I/O operations.

7.2.4 Counting Breakpoint Events and Interaction with Trace

The EVENT COUNT parameter is set when the BP command is executed. Its value refers to the total number of all qualifying events that will be recorded before program execution is halted. Any combination of memory and I/O events will be totaled to satisfy the number selected by the user (for example, if EVENT COUNT is set to 30 and 5 events are I/O cycles, then 25 memory events will satisfy the number required for a break to occur). The DELAY COUNT will begin after the 30 events have caused the EVENT COUNT to go to zero. If the user specified a DELAY COUNT of 10, then the actual break will occur after the 10 trace samples that follow the final event.

In some cases, the number of trace samples taken may exceed the specified value of the delay count because of the nature of the instruction being executed, or because the emulator can only stop on an instruction boundary.

There are other circumstances in which the number of trace samples may exceed the delay count, and the user should be aware of this possibility. For example, breakpoint evaluation occurs one clock cycle after the occurrence of the breakpoint condition. Consequently, breakpoint events on single cycle instructions, or on the last cycle of any multi-cycle instruction, cause an additional instruction to be executed prior to halting.

7.3 TRACE

The trace function of the breakpoint-trace board provides a record of the machine cycles that occur during the execution of a program. Each traceable cycle is sampled, recorded, and stored in the trace buffer so that it can be recalled for display or printing at the discretion of the user.

Although similar to the breakpoint function, trace is a separate and distinct operation. They may interact with one another in some circumstances. For example, a satisfaction of breakpoint conditions that results in an actual break in the execution of the program will terminate the trace; and vice versa, if the trace reaches the end of the trace buffer, the result is (optionally) the same as a breakpoint; and program execution is halted. (If the response to the TRACE COUNT parameter of the TR command is zero, an infinite trace count is selected, and the traces wrap around and overwrite from the beginning of the trace buffer.) Also, as stated earlier, the trace function must be invoked if the delay count for breakpoints is used.

7.3.1 Initializing the Trace Mode

The TR (Trace) command configures the TRACE COUNT, ADDRESS MASK, and TRACE MODE parameters and begins the process of establishing the criteria for traceable samples. If the TR is entered using the exclamation point terminator, the command and its parameters are displayed as follows.

```
TR!<CR>
  TRACE COUNT (1-7FF, 0=INFINITE)    = 000
  ADDRESS MASK (ONES ENABLE)         = FFFF
  TRACE MODE [0=TRM & TRIO, 1=TRIX] = 0
```

7.3.1.1 Setting the Trace Count

The TRACE COUNT parameter is actually a specification of the number of trace samples to be stored in the trace buffer. The trace buffer is a dynamically allocated amount of memory with a maximum value of 7FF (2047). If the user does not want program execution halted when the trace buffer is full, he may choose 0 for the trace count. This allows an indefinite number of trace samples to be stored. However when the buffer is full, the succeeding trace samples are written to the buffer beginning with location zero. The storage of trace samples will continue to wrap around the trace buffer until execution is terminated.

When a program is executed, the RUN command resets the breakpoint and trace circuits to operate with all the parameters specified in the breakpoint and trace commands. This includes clearing all trace samples taken and events counted. The SS (Single-Step) and CRUN (Continue RUN) commands, however, perform execution without losing previous breakpoint and trace history as long as the event, delay, or trace counts are not changed. All other breakpoint and trace parameters can be changed without losing this history.

When a parameter is changed, excluding one or more of the counts mentioned above, a pause will be observed on the first execution of the SS or CRUN command. This pause is the time it takes for the parameter change(s) to be incorporated into the breakpoint and trace circuits.

7.3.1.2 Selecting the Trace Address Mask

(Please refer to Appendix B for details concerning data and address masking.) The ADDRESS MASK operates in the same manner as that of the BP command for breakpoint locations. Judicious selection of a mask value may provide the user with several ranges of addresses to be traced. For example, if the TRACE ADDRESS #1 value is set to 0, TRACE ADDRESS #2 is set to 80, and the ADDRESS MASK is assigned a value of 3FFF, four address ranges are defined. The resulting ranges that are "in bounds" are 0000 - 0080, 4000 - 4080, 8000 - 8080, and C000 - C080.

7.3.1.3 Selecting the Trace Mode

The TRACE MODE defines the type of machine cycle to be identified as traceable. Selecting a 0 value indicates that the user wants to trace samples related to memory and/or CRU access. (Note that the square brackets indicate that the user may specify the values as the alphabetic codes rather than their numeric codes.)

If the TRM & TRIO values are selected for the TRACE MODE parameter, the TRM and TRIO commands should be executed to ensure that the qualifying criteria are properly set. Choosing the TRIX value means that the TRIX command is used to establish the criteria for traceable samples related to memory accesses outside the ranges specified by the TRACE ADDRESS parameters and RANGE INDICATORS.

7.3.2 Tracing Memory Access Samples

Entering the TRM command for display using the exclamation point terminator results in the following display.

```
TRM!<CR>
  QUALIFIER [OFF, MA, MR, MW, IAQ]       = 0
  TRACE ADDRESS #1                       = 0000
  TRACE ADDRESS #2                       = 0000
  RANGE INDICATOR [0=NO, 1=YES]          = 0
  EMU DATA COMPARE WORD                  = 0000
  EMU DATA COMPARE MASK (ONES ENABLE)    = 0000
  EXT DATA COMPARE BYTE                  = 00
  EXT DATA COMPARE MASK (ONES ENABLE)    = 00
```

7.3.2.1 Selecting Memory Access Trace Qualifiers

The parameters associated with the TRM command define the criteria for a trace to occur for memory operations. The user qualifies the type of memory operation that will be recorded as a traceable sample. These qualifiers are defined as follows.

Qualifier = 0 (OFF)	This disables the memory trace. No trace samples of memory operations are recorded.
Qualifier = 1 (MA)	This selection enables the trace for all memory operations. Thus, any memory access (read, write, or instruction acquisition) will be recorded as a traceable sample.
Qualifier = 2 (MR)	This value enables the trace for memory read and instruction acquisition operations only.
Qualifier = 3 (MW)	This value enables the trace for a memory write operation only.
Qualifier = 4 (IAQ)	This selection indicates that only instruction acquisition operations are to be recorded as traceable samples.

7.3.2.2 Selecting Memory Addresses for Tracing

The two ADDRESS parameters and the RANGE INDICATOR function in a complementary fashion. The RANGE INDICATOR parameter enables or disables the use of the address ranges specified in TRACE ADDRESS #1 and TRACE ADDRESS #2 to define the boundaries of a range of values. If the RANGE INDICATOR is OFF, only the two TRACE ADDRESS values qualify as traceable samples. If RANGE INDICATOR is ON, the range of addresses between the two TRACE ADDRESS parameter values is used.

If TRACE ADDRESS #1 is greater than TRACE ADDRESS #2, the addresses inside the range are excluded as traceable. If only one address location is to be used, then both address values should be the same. Note that setting the ADDRESS MASK in the TR command may also provide a means for specifying multiple ranges to be used for traces.

NOTE

The monitor checks for the exclusive address range before performing address masking.

7.3.2.3 Selecting Data Compare Words and Masks for Tracing

(Please refer to Appendix B for details concerning data and address masking.) The EMU DATA COMPARE WORD parameter allows the user to specify that a particular value on the data bus is to be used as another requirement for a traceable sample. If the EMU DATA COMPARE MASK is enabled, the user may specify ranges of the data compare word to be used as a criterion for a traceable sample. By specifying the data compare mask, each ZERO-bit in the data mask is set to a "don't care" condition. Thus, if the data word is specified as 001A and the data compare mask is set to FFF0, all data words that have 001X (where X is any hexadecimal value between 0 and F) will be recorded as satisfying this requirement for a traceable sample. If the mask is set to FFFF, then only the value 001A will satisfy this trace criterion.

7.3.2.4 Selecting Extended Data Compare Bytes and Masks for Tracing

(Please refer to Appendix B for details concerning data and address masking.) The EXT DATA COMPARE BYTE refers to the extended address parameter associated with the INIT command. When the emulator is first initialized using the INIT command, the last parameter is the NUMBER OF EXTENDED ADDRESS BITS. The original power-up default sets the number of address bits for the emulator to 16 bits (0 - FFFF) with 8 bits (one byte) allocated as data bits. If the user chooses to extend the address bits, any number of bits may be added up to the total of 8 bits available. The number of data bits is reduced by the number of bits designated as extended address bits. Thus, if the user chooses to add 4 bits to the address field, the data field is reduced to 4 bits.

The EXT DATA COMPARE MASK parameter operates in the same way as the EMU DATA COMPARE MASK discussed above except that it applies to the data field byte remaining after the extended address bits have been assigned.

7.3.2.5 Summary of Trace Sample Criteria

A trace sample is recorded when the following Boolean Equation is satisfied.

```
TRACE SAMPLE ENABLE =  TRACE ADDRESS IN RANGE
                                 *
                        DATABUS QUALIFIED (see Note below)
                                 *
                       TRACE QUALIFIER SATISFIED
                                 *
   [EXTENDED DATA QUALIFIED * (EXTENDED ADDR. BITS QUALIFIED)]
```

NOTE: Calculation of DATA BUS Qualification: If the following equation is zero then the DATA BUS is qualified.

```
<DATA BUS> ⊕ (<DATA COMPARE WORD>*<DATA MASK>)
```

7.3.3 Tracing I/O Cycle Samples

The user has the option of specifying traceable samples on a memory operation and/or an I/O operation. When the I/O operation is chosen, data is read from the data bus and is assumed to be from an external device connected to the specified I/O port(s). Again, the user specifies the criteria for considering such data as traceable samples. The process is very similar to the trace-memory operations discussed above, except that there are no emulator data compare word or emulator data compare mask actions.

The TRIO command is used to define the parameters for traceable samples during I/O operations. If TRIO is entered using the exclamation point terminator, the display appears as follows.

```
TRIO!<CR>
  QUALIFIER [OFF, IOA]                   = 0
  TRACE ADDRESS #1                       = 0000
  TRACE ADDRESS #2                       = 0000
  RANGE INDICATOR [0=NO, 1=YES]          = 0
  EXT DATA COMPARE BYTE                  = 00
  EXT DATA COMPARE MASK (ONES ENABLE)    = 00
```

7.3.3.1 Selecting I/O Access Qualifiers

The criteria for establishing a traceable sample during an I/O operation in any CRU address on the emulator board are qualified as follows.

Qualifier = 0 (OFF) This disables the trace operation during I/O cycles.

Qualifier = 1 (IOA) This defines any read from or write to port as a an acceptable criterion for a traceable sample.

7.3.3.2 Selecting I/O Addresses for Tracing

The TRACE ADDRESS #1 and TRACE ADDRESS #2 parameters operate the same way as these parameters in the TRM command. However, the addresses specified refer to bit numbers rather than memory locations. The maximum value for these parameters is therefore 32760 (FFFF). This is because address bit 15 is not used and is a "don't care". If the user specifies TRACE ADDRESS #1 as 0 and TRACE ADDRESS #2 as FFFF, samples that occur on any of the CRU addresses are considered as potential traceable samples, if the RANGE INDICATOR is ON. If the user chooses CRU address 0 for TRACE ADDRESS #1 and CRU address FFFF for TRACE ADDRESS #2 and the RANGE INDICATOR is OFF, only I/O cycles that occur at CRU addresses 0 and FFFF will be eligible for consideration as traceable samples. If TRACE ADDRESS #1 is greater than TRACE ADDRESS #2 then the range specified is excluded as traceable.

7.3.3.3 Selecting the Remaining TRIO Parameters

The remaining parameters for the TRIO command are the same as those for the TRM command discussed in Paragraphs 7.3.2.3 and 7.3.2.4 above (except for the omission of the EMU DATA COMPARE WORD and EMU DATA COMPARE MASK). These parameters must be set for I/O operations in the same way as they are for memory operations in order to specify conditions for traceable samples during I/O operations. Although the parameters are similar in function, they do apply to independent commands. The user must keep this in mind when configuring the system for memory and/or I/O operations.

7.3.4 Tracing Memory Accesses Outside the Specified Range(s)

If the user specifies the TRIX as the TRACE MODE when executing the TR command, an expanded memory access capability is invoked. There may be operation codes in the address range(s) that access memory locations outside the specified range(s) specified in the TRIX command.

The advantage of the TRIX command is that the non-IAQ memory or I/O cycles are not limited, since the range specified in the command is an IAQ (Instruction Acquisition) range.

For example, the user wants to trace a limited number of instructions (i.e., a very small range). He can still see memory and I/O cycles that are outside the specified range, because the memory and I/O cycles that are traced are a result of the instruction inside the range.

If the TRIX command is entered using the exclamation point terminator, the display appears as follows:

```
TRIX!<CR>
  OPCODE QUALIFIER? [0=NO, 1=YES]          = 0
  MEMORY QUALIFIER  [OFF, MA, MR, MW]      = 0
  I/O QUALIFIERS [OFF, IOA]                = 0
  TRACE ADDRESS #1                         = 0000
  TRACE ADDRESS #2                         = 0000
  RANGE INDICATOR [0=NO, 1=YES]            = 0
  EXT DATA COMPARE BYTE                    = 0
  EXT DATA MASK (ONES ENABLE)              = 0
```

7.3.4.1 Selecting the TRIX Opcode Qualifiers for Tracing

In order to use the OPCODE QUALIFIER, the user should first execute the SOR (Set Opcode Range) command. The SOR!<CR> display shows the associated parameters and their values as follows.

```
SOR!
  LOWER OPCODE BOUND (LS NIBBLE IGNORED)     = 0
  UPPER OPCODE BOUND (LS NIBBLE IGNORED)     = 0
  [0=DELETE RANGE, 1=ADD RANGE]              = 0
```

The user is able to specify the range of opcodes that will qualify for traceable samples using this command. The opcode is logically ANDed with FFF0 to truncate the last four bits of the opcode and make them "don't care" masks. If the OPCODE QUALIFIERS parameter is disabled, this command is ignored and its parameter values are void.

The range of opcode values specified when SOR is executed may be deleted from the existing set of ranges by selecting 0=DELETE RANGE, or added to the set by selecting 1=ADD RANGE. Any ranges that are defined will remain in effect until cleared by the DELETE/ADD RANGE parameter. Thus, the OPCODE RANGES specified by the execution of this command are cumulative until cleared.

The UPPER and LOWER BOUNDS allow the user to specify a set of opcodes for tracing. For example, perhaps the user wants to trace all JUMP instructions. This feature allows him to select the opcode range that includes only the JUMP operations.

7.3.4.2 Setting Other TRIX Parameters

Returning to the TRIX command, the MEMORY QUALIFIER and I/O QUALIFIER parameters have the same function as these parameters in the TRM and TRIO commands respectively. The remaining parameters are also identical to those of these two commands. (Please refer to the discussion above for explanations of these parameters.)

7.3.4.3 Summary of TRIX Trace Sample Criteria

A trace sample is recorded using the TRIX mode when the following Boolean Equation is satisfied.

```
        CYCLE = INSTRUCTION ACQUISITION (IAQ) (See Note 1)
                                  *
                TRACE ADDRESS IN RANGE (See Note 1)
                                  *
      [EXTENDED DATA QUALIFIED + (EXTENDED ADDR.)] (See Note 1)
                                  +
                   QUALIFIED IAQ HAS BEEN SEEN
                                  *
            [MEMORY CYCLE * MEMORY QUALIFIER SATISFIED
                                  +
               I/O CYCLE * I/O QUALIFIER SATISFIED]
```

NOTES: 1. These terms set a "QUALIFIED IAQ HAS BEEN SEEN" flag.

2. Once the "QUALIFIED IAQ HAS BEEN SEEN" flag is set, the next unqualified IAQ will reset it.

7.4 DISABLING THE BREAKPOINT/TRACE FUNCTIONS

Once the BPM, TRM, BPIO, TRIO, or TRIX command is executed, the breakpoint-trace board option remains in effect until all qualifiers (both breakpoint and trace qualifiers) are turned OFF. If the user has executed the BPM, BPIO, TRM, TRIO, or TRIX command, then the qualifiers must be reset. If the user wants to disable hardware breakpoints, the qualifiers for the BPM and BPIO commands must be set to their OFF values. The same thing applies to the trace capability. The qualifiers in the TRM and TRIO or the TRIX commands must be set to OFF to disable tracing.

7.5 SAMPLE SESSIONS

Example 1: Trace Mode. Enter the following test program using the XA (Execute Assembler) Command.

NOTE

* For the purpose of clarity, the columns have been expanded in the following displays.

* The TMS9995 Assembler uses two-character labels (as with the following example). If more than two characters are entered in the label field (LB) the assembler will ignore all but the last two.

* The user's entries in the following portion of this example are in the columns beneath the "LB" and "INST" headings.

Enter: XA<CR>

```
Display:                  Enter:

XA
  START ADDRESS = 0000    A<CR>
    CONTROL-E EXITS

    ADDR       DATA       LB        INST
    000A       04C4       IN        R4<CR>
    000C       0201      <SP>       LI R1,10<CR>
    000E       000A
    0010       0202      <SP>       LI R2,>1234<CR>
    0012       1234
    0014       0203      <SP>       LI R3,>10<CR>
    0016       0010
    0018       38C2       LP        MPY R2,R3<CR>
    001A       C803      <SP>       MOV R3,@>100<CR>
    001C       0100
    001E       C804      <SP>       MOV R4,@>102<CR>
    0020       0102
    0022       C0C1      <SP>       MOV R1,R3<CR>
    0024       0601      <SP>       DEC R1<CR>
    0026       16F8      <SP>       JNE LP<CR>
    0028       10F0      <SP>       JMP IN<CR>
    002A       0000      <SP>       END<CR> 0000
?
```

The user entered <SP>END<CR> to exit the XA command. The value displayed in the END instruction indicates the number of errors in the program entry.

An alternate method of exiting the XA command is to enter <CTRL>E, in which case the number of errors is displayed in the DATA column of the display's last line.

Backspace is not allowed; <SPACE> END or <CTRL>E are the only ways to quit the ASSEMBLER.

The program may be inspected in reverse assembled code by using the XRA (Execute Reverse Assembler) command.

Enter: XRA<CR>

```
Display:                                        Enter:

XRA
  START ADDRESS               = 000A           A<CR>
  NUMBER OF INSTRUCTIONS      = 000E           E<CR>
    000A 04C4  CLR  R4
    000C 0201  LI   R1,>000A
    0010 0202  LI   R2,>1234
    0014 0203  LI   R3,>0010
    0018 38C2  MPY  R2,R3
    001A C803  MOV  R3,@>0100
    001E C804  MOV  R4,@>0102
    0022 C0C1  MOV  R1,R3
    0024 0601  DEC  R1
    0026 16F8  JNE  >0018
    0028 10F0  JMP  >000A
    002A 0000  DATA >0000
    002C 1000  NOP
    002E 1000  NOP
```

```
Enter:  MR<CR>
```

```
Display:       Enter:

MR
  WP = 0000    40<CR>
  PC = 0000    A<CR>
  ST = 0000    <CR>
```

The workspace pointer is set to allocate addresses >0040 through >005E to workspace registers R0 through R15, the program counter is set to start program execution at location >000A (the address containing the first instruction in the program), and the status register is set to the value 0000.

```
Enter:  MR.RUN<CR>
```

```
Display:

MR.RUN
 MR
   WP=0040
   PC=000A
   ST=0000
 RUN
    RUNNING
```

```
Enter:  <any keystroke>
```

Display:

```
 RUN
    RUNNING
    KEY
    WP=0040   PC=001A   ST=D000
```

The halt status display shows that a key was pressed on the keyboard and caused a halt.

Enter: TR<CR>

```
Display:                                                 Enter:

TR
  TRACE COUNT (1 - 7FF, 0=INFINITE)             = 000    20<CR>
  ADDRESS MASK (ONES ENABLE)                    = FFFF   FFFF<CR>
  TRACE MODE [0=TRM & TRIO, 1=TRIX]             = 0      <CR>
```

The number of trace samples to be taken is set to 20, the address mask is enabled (which indicates that only those addresses which match the compare values will qualify as trace samples), and memory and/or I/O accesses have been selected for tracing.

Enter: TRM<CR>

```
Display:                                            Enter:

TRM
  QUALIFIER [ OFF, MA, MR, MW, IAQ ]         = 0      MA<CR>
  TRACE ADDRESS  #1                          = 0000   <CR>
  TRACE ADDRESS  #2                          = 0000   FFFF<CR>
  RANGE INDICATOR [0=NO, 1=YES]              = 0      YES<CR>
  EMU DATA COMPARE WORD                      = 0000   <CR>
  EMU DATA COMPARE MASK (ONES ENABLE)        = 0000   <CR>
  EXT DATA COMPARE BYTE                      = 00     <CR>
  EXT DATA COMPARE MASK (ONES ENABLE)        = 00     <CR>
```

By entering the MA selection in response to the QUALIFIER prompt, it is indicated that all memory access operations will qualify as trace samples. The TRACE ADDRESS range is set to cover all memory from >0000 to >FFFF.

Enter: MR.RUN<CR>

The "MR." portion of the above procedure sets the WP, PC, and ST registers to the values specified in the last execution of the MR command (>0040, >000A, and 0000, respectively).

Display:

```
MR.RUN
 MR
  WP = 0040
  PC = 000A
  ST = 0000
 RUN
    RUNNING
    TMF
    WP=0040  PC=0018  ST=D000
```

The TMF (shown in the halt status display above) shows that program execution halted because the number of traces specified in the TRACE COUNT parameter of the TR command (>20) had been taken. The traces can be displayed as follows:

Enter: IT<CR>

Display: Enter:

```
IT
  FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000   <CR>
```

Entering <CR> in response to this prompt accepts the present parameter value of 0 and selects the entire trace memory for inspection.

Display:

```
IT
  FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000

    INDEX CYCLE QUAL  EXTQUALS       ADDR DATA  RVRS ASSEMBLY

    0000        IAQ   11111111       000A 04C4  CLR  R4
    0001        IAQ   11111111       000C 0201  LI   R1,>000A
    0002        MW    11111111       0048 0000
    0003        MR    11111111       000E 000A
    0004        IAQ   11111111       0010 0202  LI   R2,>1234
    0005        MW    11111111       0042 000A
    0006        MR    11111111       0012 1234
    0007        IAQ   11111111       0014 0203  LI   R3,>0010
    0008        MW    11111111       0044 1234
    0009        MR    11111111       0016 0010
    000A        IAQ   11111111       0018 38C2  MPY  R2,R3
    000B        MW    11111111       0046 0010
    000C        MR    11111111       0044 1234
    000D        MR    11111111       0046 0010
    000E        MW    11111111       0046 0001
    000F        IAQ   11111111       001A C803  MOV  R3,@>0100
    0010        MW    11111111       0048 2340
    0011        MR    11111111       0046 0001
    0012        MR    11111111       001C 0100
[]
```

Enter: <SP>

Entering the <SP> (or <CR>) scrolls to the next page of samples. Note that the two displays overlap (each shows index numbers >000D through >0012) since only >20 (32, decimal) samples were taken and >12 (19,decimal) samples are displayed on each screen.

Display:

INDEX	CYCLE	QUAL	EXTQUALS	ADDR	DATA	RVRS	ASSEMBLY
000D		MR	11111111	0046	0010		
000E		MW	11111111	0046	0001		
000F		IAQ	11111111	001A	C803	MOV	R3,@>0100
0010		MW	11111111	0048	2340		
0011		MR	11111111	0046	0001		
0012		MR	11111111	001C	0100		
0013		IAQ	11111111	001E	C804	MOV	R4,@>0102
0014		MW	11111111	0100	0001		
0015		MR	11111111	0048	2340		
0016		MR	11111111	0020	0102		
0017		IAQ	11111111	0022	C0C1	MOV	R1,R3
0018		MW	11111111	0102	2340		
0019		MR	11111111	0042	000A		
001A		IAQ	11111111	0024	0601	DEC	R1
001B		MW	11111111	0046	000A		
001C		MR	11111111	0042	000A		
001D		IAQ	11111111	0026	16F8	JNE	>0018
001E		MW	11111111	0042	0009		
001F		IAQ	11111111	0018	38C2	MPY	R2,R3

[]

Enter: Q

Entering "Q" quits the IT command.

Enter: BP<CR>

Display:		Enter:
BP		
EVENT COUNT (0 - 7FFF)	= 0000	5<CR>
DELAY COUNT (0 - 7FF)	= 000	30<CR>
ADDRESS MASK (ONES ENABLE)	= FFFF	<CR>

The breakpoint criteria are defined: five events will occur before program execution is interrupted (EVENT COUNT = 5), >30 traces are to be taken after the event counter reaches zero (DELAY COUNT = 30), and the ADDRESS MASK is enabled to specify that only those addresses which exactly match the compare value will be allowed to qualify as events. Now that the BP command's parameters are defined, other breakpoint criteria can be set using the BPM (Breakpoint, Memory) command:

Enter: BPM<CR>

Display: Enter:

```
BPM
  QUALIFIER [ OFF, MA, MR, MW, IAQ ]        = 0      MA<CR>
  BREAKPOINT ADDRESS  #1                    = 0000   18<CR>
  BREAKPOINT ADDRESS  #2                    = 0000   26<CR>
  RANGE INDICATOR [0=NO, 1=YES]             = 0      YES<CR>
  EMU DATA COMPARE WORD                     = 0000   <CR>
  EMU DATA COMPARE MASK (ONES ENABLE)       = 0000   <CR>
  EXT DATA COMPARE BYTE                     = 00     <CR>
  EXT DATA COMPARE MASK (ONES ENABLE)       = 00     <CR>
```

The MA option is selected in the QUALIFIER parameter to allow all memory accesses to qualify as breakpoints. The address range is selected as locations >0018 through >0026, inclusive. Reset the registers and execute the program again (this time, the ";" terminator follows the MR command and suppresses the display resulting from its execution):

Enter: MR;RUN<CR>

Display:

```
MR;RUN
 RUN
    RUNNING
    TMF
    WP=0040  PC=0018  ST=D000
```

The TMF in the halt status display shows that the number of traces selected in the TR command (>20) were taken. Execute the IT command, beginning with the oldest sample, to view the trace samples that were taken (again, note the overlap in the first and second displays):

Enter: IT<CR>

Display: Enter:

```
IT
  FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000   <CR>
```

Display:

```
IT
  FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000

    INDEX CYCLE QUAL  EXTQUALS       ADDR DATA  RVRS ASSEMBLY

    0000        IAQ   11111111       000A 04C4  CLR  R4
    0001        IAQ   11111111       000C 0201  LI   R1,>000A
    0002        MW    11111111       0048 0000
    0003        MR    11111111       000E 000A
    0004        IAQ   11111111       0010 0202  LI   R2,>1234
    0005        MW    11111111       0042 000A
    0006        MR    11111111       0012 1234
    0007        IAQ   11111111       0014 0203  LI   R3,>0010
    0008        MW    11111111       0044 1234
    0009        MR    11111111       0016 0010
    000A  EVNT  IAQ   11111111       0018 38C2  MPY  R2,R3
    000B        MW    11111111       0046 0010
    000C        MR    11111111       0044 1234
    000D        MR    11111111       0046 0010
    000E        MW    11111111       0046 0001
    000F  EVNT  IAQ   11111111       001A C803  MOV  R3,@>0100
    0010        MW    11111111       0048 2340
    0011        MR    11111111       0046 0001
    0012  EVNT  MR    11111111       001C 0100
[]
```

Enter: <SP>

Display:

```
    INDEX CYCLE QUAL  EXTQUALS       ADDR DATA  RVRS ASSEMBLY

    000D        MR    11111111       0046 0010
    000E        MW    11111111       0046 0001
    000F  EVNT  IAQ   11111111       001A C803  MOV  R3,@>0100
    0010        MW    11111111       0048 2340
    0011        MR    11111111       0046 0001
    0012  EVNT  MR    11111111       001C 0100
    0013  EVNT  IAQ   11111111       001E C804  MOV  R4,@>0102
    0014        MW    11111111       0100 0001
    0015        MR    11111111       0048 2340
    0016  *EVT  MR    11111111       0020 0102
    0017  EVNT  IAQ   11111111       0022 C0C1  MOV  R1,R3
    0018        MW    11111111       0102 2340
    0019        MR    11111111       0042 000A
    001A  EVNT  IAQ   11111111       0024 0601  DEC  R1
    001B        MW    11111111       0046 000A
    001C        MR    11111111       0042 000A
    001D  EVNT  IAQ   11111111       0026 16F8  JNE  >0018
    001E        MW    11111111       0042 0009
    001F  EVNT  IAQ   11111111       0018 38C2  MPY  R2,R3
[]
```

Enter: Q

Display:

?

Reset the TRACE COUNT to infinite:

Enter: TR<CR>

```
Display:                                                 Enter:

TR
  TRACE COUNT (1 - 7FF, 0=INFINITE)           = 020     0<CR>
  ADDRESS MASK (ONES ENABLE)                  = FFFF    <CR>
  TRACE MODE [0=TRM & TRIO, 1=TRIX]           = 0       <CR>
```

Reset the registers using the MR command and the ";" terminator; then reexecute the program.

Enter: MR;RUN<CR>

Display:

```
RUN
    RUNNING
    HBP
    INDEX CYCLE QUAL  EXTQUALS      ADDR DATA  RVRS ASSEMBLY
          *EVT  MR    11111111      0020 0102
```

Notice that the halt status display shows a hardware breakpoint (HBP) as the reason for the halt. Index number 0016 of the following trace display contains the *EVT flag in the CYCLE column, identifying the event that caused the halt.

Enter: IT<CR>

```
Display:                                                 Enter:

IT
  FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000      <CR>
```

Display:

```
IT
  FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000
    INDEX CYCLE QUAL  EXTQUALS      ADDR DATA  RVRS ASSEMBLY

    0000        IAQ   11111111      000A 04C4  CLR  R4
    0001        IAQ   11111111      000C 0201  LI   R1,>000A
    0002        MW    11111111      0048 0000
    0003        MR    11111111      000E 000A
    0004        IAQ   11111111      0010 0202  LI   R2,>1234
    0005        MW    11111111      0042 000A
    0006        MR    11111111      0012 1234
    0007        IAQ   11111111      0014 0203  LI   R3,>0010
    0008        MW    11111111      0044 1234
    0009        MR    11111111      0016 0010
    000A  EVNT  IAQ   11111111      0018 38C2  MPY  R2,R3
    000B        MW    11111111      0046 0010
    000C        MR    11111111      0044 1234
    000D        MR    11111111      0046 0010
    000E        MW    11111111      0046 0001
    000F  EVNT  IAQ   11111111      001A C803  MOV  R3,@>0100
    0010        MW    11111111      0048 2340
    0011        MR    11111111      0046 0001
    0012  EVNT  MR    11111111      001C 0100
[]
```

Enter: <SP>

Display:

```
    INDEX CYCLE QUAL  EXTQUALS      ADDR DATA  RVRS ASSEMBLY

    0013  EVNT  IAQ   11111111      001E C804  MOV  R4,@>0102
    0014        MW    11111111      0100 0001
    0015        MR    11111111      0048 2340
    0016  *EVT  MR    11111111      0020 0102
    0017  EVNT  IAQ   11111111      0022 C0C1  MOV  R1,R3
    0018        MW    11111111      0102 2340
    0019        MR    11111111      0042 000A
    001A  EVNT  IAQ   11111111      0024 0601  DEC  R1
    001B        MW    11111111      0046 000A
    001C        MR    11111111      0042 000A
    001D  EVNT  IAQ   11111111      0026 16F8  JNE  >0018
    001E        MW    11111111      0042 0009
    001F  EVNT  IAQ   11111111      0018 38C2  MPY  R2,R3
    0020        MR    11111111      0044 1234
    0021        MR    11111111      0046 000A
    0022        MW    11111111      0046 0000
    0023  EVNT  IAQ   11111111      001A C803  MOV  R3,@>0100
    0024        MW    11111111      0048 B608
    0025        MR    11111111      0046 0000
[]
```

Enter: Q

Note that index number 0016 is not the last line of the trace display, even though it was the event that set the event counter to zero. This occurred because the DELAY COUNT parameter of the BP command was set to a non-zero value (30) and this number of trace samples were taken after the EVENT COUNT went to zero (trace index number 0016). The trace display shows only >F samples after the *EVT because the IT command was quit by entering "Q"; if a <CR> or <SP> had been entered instead, the remaining trace samples would have been shown.

Example 2: Using the TRIX command to trace all memory accesses resulting from the execution of IAQs. The sample program for this example is reverse assembled below. Enter the program using the XA (Execute Assembler) command.

Display:

```
XRA
  START ADDRESS             = 0000
  NUMBER OF INSTRUCTIONS    = 0011
    0000 0200  LI    R0,>FFFF
    0004 0201  LI    R1,>BEAD
    0008 06A0  BL    @>0016
    000C 0580  INC   R0
    000E 16FC  JNE   >0008
    0010 0000  DATA  >0000
    0012 0000  DATA  >0000
    0014 0000  DATA  >0000
    0016 C081  MOV   R1,R2
    0018 0B12  SRC   R2,1
    001A A042  A     R2,R1
    001C 045B  RT
    001E 0000  DATA  >0000
    0020 0000  DATA  >0000
    0022 0000  DATA  >0000
    0024 0000  DATA  >0000
    0026 0000  DATA  >0000
```

This example shows how all memory accesses are traced, which are the results of the IAQs in the subroutine of the sample program (addresses >0016 through >001C). The advantage of using TRIX (instead of TRM) to trace the IAQ memory cycles is that the data manipulations can be seen (TRM would show only the IAQ's).

Enter: TR<CR>

```
Display:                                                  Enter:

TR
  TRADE COUNT (1 - 7FF, 0=INFINITE)              = 000   <CR>
  ADDRESS MASK (ONES ENABLE)                     = FFFF  <CR>
  TRACE MODE [0=TRM & TRIO, 1=TRIX]              = 0     1<CR>
```

The TRIX TRACE MODE with infinite TRACE COUNT is selected in the TR (Trace) command; the address mask is enabled.

Enter: TRIX<CR>

```
Display:                                              Enter:

TRIX
  OPCODE QUALIFIERS? [0=NO, 1=YES]            = 0      <CR>
  MEMORY QUALIFIER [ OFF, MA, MR, MW ]        = 0      MA<CR>
  I/O QUALIFIER [ OFF, IOA ]                  = 0      <CR>
  TRACE ADDRESS  #1                           = 0000   0016<CR>
  TRACE ADDRESS  #2                           = 0000   001C<CR>
  RANGE INDICATOR [0=NO, 1=YES]               = 0      YES<CR>
  EXT DATA COMPARE BYTE                       = 00     <CR>
  EXT DATA COMPARE MASK (ONES ENABLE)         = 00     <CR>
```

In the execution of the TRIX command, the MEMORY QUALIFIER is set to MA to allow all memory accesses resulting from the IAQ's in the program to qualify as trace samples. A range of >0016 through >001C is specified for the TRACE ADDRESSES, this sets the addresses to be traced to the subroutine where the data is manipulated.

The program can now be run, and the resulting trace displayed, as follows:

NOTE

* The procedure to initiate the run also resets the workspace pointer, program counter, and status registers to the values set in the last execution of the MR (Modify Registers) command, by executing the MR command with a period terminator (MR.). In this case the reset values are 0040, 0000, and 0000 for the workspace pointer, program counter, and status registers, respectively.

* Notice that the DT (Display Trace) command is executed in the procedure in the parenthetical entry mode. The values in the parentheses specify that the display is to start with the oldest sample (000) and show the entire trace memory (7FE).

Enter: MR.RUN DT(000,7FE)<CR>

Display:

```
MR.RUN DT(000,7FE)
 MR
  WP = 0040
  PC = 0000
  ST = 0000
 RUN
    RUNNING
    Z-MID
    WP=0040  PC=0010  ST=3000

 DT
  FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000
  NUMBER OF SAMPLES                          = 7FE

    INDEX CYCLE QUAL  EXTQUALS     ADDR DATA  RVRS ASSEMBLY

    0000        IAQ   11111111     0016 C081  MOV  R1,R2
    0001        MR    11111111     0042 BEAD
    0002        IAQ   11111111     0018 0B12  SRC  R2,1
    0003        MW    11111111     0044 BEAD
    0004        MR    11111111     0044 BEAD
    0005        IAQ   11111111     001A A042  A    R2,R1
    0006        MW    11111111     0044 DF56
    0007        MR    11111111     0044 DF56
    0008        MR    11111111     0042 BEAD
    0009        IAQ   11111111     001C 045B  RT
    000A        MW    11111111     0042 9E03
    000B        MR    11111111     0056 000C
```

The DT display shows the IAQ's (indexes 0000, 0002, 0005, and 0009) in the subroutine where the data manipulations occurred (addresses >0016 through >001C). The memory accesses (both read and write) resulting from these IAQs are also shown in the display.

Example 3: Using the TRIX command to trace opcodes in a program. Another useful function of the TRIX command is its ability to trace opcodes and opcode ranges used in programs. The program for this example is reverse assembled below. Enter the program using the XA (Execute Assembler) command.

```
XRA
  START ADDRESS            = 0000
  NUMBER OF INSTRUCTIONS   = 0011
    0000 1000  NOP
    0002 0420  BLWP @>000A
    0006 0000  DATA >0000
    0008 1000  NOP
    000A 0014  DATA >0014
    000C 000E  DATA >000E
    0012 0380  RTWP
    0014 9E03  CB   R3,*R8+
    0016 0001  DATA >0001
    0018 0002  DATA >0002
    001A 0003  DATA >0003
    001C 0004  DATA >0004
    001E 0005  DATA >0005
    0020 0006  DATA >0006
    0022 0007  DATA >0007
    0024 0008  DATA >0008
```

The BLWP (Branch-and-Link to Workspace Pointer vector) opcode, >0420, is traced in this example. The opcode range is set to >0400 through >0430 in the SOR (Set Opcode Range) command (BLWP is opcode 0420):

Enter: SOR<CR>

```
Display:                                                Enter:

SOR
  LOWER OPCODE BOUND (LS NIBBLE IGNORED) = 0000   400<CR>
  UPPER OPCODE BOUND (LS NIBBLE IGNORED) = 0000   430<CR>
  ( 0=DELETE RANGE, 1=ADD RANGE )        = 0      1<CR>
```

The TR (Trace) command must be executed to set the trace count (infinite, in this case); define the address mask (all bits enabled); and select the trace mode (TRIX).

Enter: TR<CR>

```
Display:                                                Enter:

TR
  TRACE COUNT (1 - 7FF, 0=INFINITE)      = 000    <CR>
  ADDRESS MASK (ONES ENABLE)             = FFFF   <CR>
  TRACE MODE [0=TRM & TRIO, 1=TRIX]      = 0      1<CR>
```

Next, the TRIX criteria are defined. Opcode qualifiers are selected, the memory qualifier is set to MA (all memory accesses associated with IAQ's are allowed to qualify for trace samples); and the full range of memory is selected for tracing (0000 through FFFF).

Enter: TRIX<CR>

```
Display:                                            Enter:

TRIX
  OPCODE QUALIFIERS? [0=NO, 1=YES]        = 0       YES<CR>
  MEMORY QUALIFIER [ OFF, MA, MR, MW ]    = 0       MA<CR>
  I/O QUALIFIER [ OFF, IOA ]              = 0       <CR>
  TRACE ADDRESS  #1                       = 0000    <CR>
  TRACE ADDRESS  #2                       = 0000    FFFF<CR>
  RANGE INDICATOR [0=NO, 1=YES]           = 0       YES<CR>
  EXT DATA COMPARE BYTE                   = 00      <CR>
  EXT DATA COMPARE MASK (ONES ENABLE)     = 00      <CR>
```

The sample program can now be run, and the resulting trace displayed, as follows:

NOTE

* The procedure to initiate the run also resets the workspace pointer, program counter, and status registers to the values set in the last execution of the MR (Modify Registers) command, by executing the MR command with a period terminator (MR.). In this case the reset values are 0040, 0000, and 0000 for the workspace pointer, program counter, and status registers, respectively.

* Notice that the DT (Display Trace) command is executed in the procedure in the parenthetical entry mode. The values in the parentheses specify that the display is to start with the oldest sample (000) and will show the entire trace memory (7FE).

Enter: MR.RUN DT(000,7FE)<CR>

Display:

```
MR.RUN DT(000,7FE)
 MR
   WP=0040
   PC=0000
   ST=0000
 RUN
    RUNNING
    Z-MID
    WP=0040   PC=0006   ST=0000

 DT
  FIRST SAMPLE (0-7FE, 0=OLDEST, 7FF=NEWEST) = 000
  NUMBER OF SAMPLES                          = 7FE

    INDEX CYCLE QUAL  EXTQUALS       ADDR DATA  RVRS ASSEMBLY
    0000        IAQ   11111111       0002 0420  BLWP @>000A
    0001        MR    11111111       0004 000A
    0002        MR    11111111       000A 0014
    0003        MR    11111111       000C 000E
    0004        MW    11111111       002E 0040
    0005        MW    11111111       0030 0006
    0006        MW    11111111       0032 0000

?
```

The DT display shows the cycles that were traced during the execution of the program. Notice that the first trace sample is an IAQ (a result of the TRIX command) and that the trace is further limited to the opcode range specified in the SOR command (a result of a YES answer to the OPCODE QUALIFIERS prompt in the TRIX command). All memory cycles associated with the IAQ (trace index 0000) were also traced (indexes 0001 through 0006). This occurred because the TRIX command MEMORY QUALIFIER was set to MA to allow all memory cycles associated with the IAQ(s) being traced to also qualify as trace samples.

SECTION 8

ERROR CODES

The following is a list of error codes, messages, and explanations of the various error conditions that may arise during the operation of the XDS TMS9995 Emulator.

Code	Type of Error	Error Description
0XXX	SCAN PASS ERROR	An error was found during the first pass of the command buffer. No commands have been executed.
01XX	TERMINATION ERROR	An invalid character was used for termination of input.
0100	Invalid Buffer Terminator	A character was used to terminate input to the buffer which is not a valid buffer terminator.
0101	Invalid Terminator	A character was used to terminate a command which was not a valid command terminator.
0105	Terminator Not Expected	When using the parenthesized mode of entry, one or more parameters were left out of a command.
0108	End of Buffer Expected	The end of the input buffer is expected, data appears after an asterisk in the command line.
02XX	COMMAND ERROR	The command that was entered was not recognized by the TMS9995 Emulator Monitor Program.
0202	Invalid Command	The command that was entered was misspelled, run together with other commands, or was not recognized as valid for this emulator.
0209	Index Not Expected	A numeric index was found associated with a non-indexed command or register.
0210	Invalid With Other Input	An input (command) was scanned which can only be executed when it is the only command in the input buffer.

Code	Type of Error	Error Description
03XX	SYNTAX ERROR	An invalid character was used as a separator.
0304	Invalid Separator	The character entered was not a valid separator.
0306	Separator Not Expected	A separator was found which was not expected.
0307	Separator Expected	A character was found that was not a separator when a separator was expected.
04XX	PARAMETER ERROR	An error occurred in a parameter entry.
0403	Invalid Parameter	Characters entered are not valid for parameter entries.
040F	Parameter Too Large	Parameter entered was found to be greater than the maximum value allowed for this parameter.
05XX	VARIABLE ERROR	An error occurred in a variable entry.
050A	Variable Index Not Expected	An index was added to a non-indexed variable.
050B	Invalid Variable Index	Either no index, or a nondecimal index input, was found.
050C	Variable Index Too Large	The variable index that was entered was greater than the maximum value allowed.
050D	Variable Not Writable	An attempt was made to assign a value to a read-only variable.
06XX	COMMAND USAGE ERROR	A command is used in a procedure out of the sequence expected by the monitor or another command must be executed before the command producing the error can be executed.
0601	Snap Must Be First	The SNAP command was entered as other than the first command in the buffer.
0602	Cursor Control Not Initialized	Cursor control must be initialized (ICC command executed) before the SNAP command can be used.

Code	Type of Error	Error Description
1XXX	EXECUTION PASS ERROR	An error occurred during the second pass of the command buffer. All commands preceding the erroneous command have been executed.
10XX	LOAD ERROR	An error occurred while loading the user's object module into the emulator's RAM.
1000	Invalid Tag	An invalid tag was encountered during the loading of the object module.
1010	Checksum Error	The checksum value, at the end of the line in the object file with a "7" tag, did not agree with the line contents.
1011	Upload End Address Less Than Start Address	An attempt was made to upload a file with the end address defined as less than the start address.
1012	Upload Error - 15 Retries	A Tektronix Hexadecimal tagged object record was unsuccessfully transmitted 15 times in response to a request to re-transmit.
1013	EXT. REF Tag in OBJ Module	When downloading a TI OBJECT file an external reference tag was encountered. This tag was ignored. This message is printed after the download to inform the user that there is an unresolved reference in the object file. The program may still be executed, but it may cause an error.
1014	ASR Handshake Valid Only in HOST Mode	The ASR handshake is not supported when uploading to the PROM port or USER port.
1015	Invalid TEK Handshake	The emulator did not receive a valid Tektronix handshake on an upload.
1016	Use Emulator #1	When in the multi-processing mode, an emulator other than that designated as #1 was in the foreground and was issued a command which is invalid except when issued to EMU #1. This error can occur only in the multi-processing environment.

Code	Type of Error	Error Description
1112	Log Device Off-Line	The LOG command has turned itself OFF because a character could not be output to the log device. Check to ensure that the connection between Port C on the XDS chassis and the log device is correct.
1113	PROM Port Off-Line	The PROM upload has been aborted. Either, or both, of the CTS (Clear To Send) or the DSR (Data Set Ready) signals has been lost on Port C. Check that the proper EIA cable is installed and that the communication switches on the communication board are set correctly.
1114	Host Port Off-Line	The HOST mode has been aborted because either (or both) CTS (Clear to Send) or DSR (Data Set Ready) signals on Port D are inactive. Check to that the proper EIA cable is installed and that communication switches on the Communication Board are properly set.
1125	Too Many Software Breakpoints	The software breakpoint limit (10, total) has been exceeded. If another breakpoint is to be set, then an existing one must be cleared.
1130	No Breakpoint Board In Unit	A hardware breakpoint command was executed but the breakpoint hardware is not installed in the XDS chassis.
1140	Parity on Memory Expansion	The memory expansion board (in the XDS chassis) reported a parity error. If an external clock is being used, any interrupt in the target clock signal will cause a malfunction of the expansion memory refresh cycles. This will cause loss of data. If possible, repeat session using the internal (emulator) clock to isolate the problem.
1145	Non Existent Breakpoint	The user attempted to delete a breakpoint that does not exist.
1150	WARNING - Overlap With EMU RAM	The user selected emulator RAM (addresses >0000 through >03FF) then attempted to map expansion memory in same locations.
1155	No Memory Board In Unit	A map command was attempted, or an attempt was made to use expansion memory, without the expansion memory board installed in the XDS chassis.

Code	Type of Error	Error Description
1167	Expansion Memory Needs INIT	The expansion memory must be initialized before executing the command that caused the error.
1175	Address is Not RAM	The user attempted to set a software breakpoint on an address that has not been mapped as RAM. A software breakpoint will not work unless the address at which it is set is RAM.
17XX	MULTI-PROCESSING ERRORS	Command execution was terminated because of a conflict found during execution.
1701	Invalid Without IMP Command	The command entered is not valid without first initializing the string of emulators with the IMP (Initialize Multi-processing Mode) command.
1705	Emulator #X Is Not Ready to Run	Emulator X (where X is a positive integer between 1 and 9) has been requested to run but cannot because it is waiting to output. This error occurs when emulator X is in background mode and a TRUN (Total RUN) or GRUN (Group RUN) command has been executed.
1710	Last Emulator = 1	The IMP (Initialize Multi-processing Mode) command was executed when the emulator was not set up for multi-processing or when the cabling for the multi-processing setup is defective.
1715	Invalid With IMP	The command that was entered is invalid in the multi-processing mode.

APPENDIX A

DIAGNOSTIC MODE

A.1 INTRODUCTION

The commands discussed in this section are used when the XDS system is in the diagnostic mode of operation. The diagnostic mode of operation is used when the user needs the help of a Texas Instrument's Regional Technology Center (RTC) to trouble shoot the XDS. The XDS is tied to the Regional Technology Center by telephone lines using the second EIA port and a Modem. Reference the XDS Hardware Installation and Operation Manual for the hardware connections required for the diagnostic mode of operation. Figure A-1 shows a block diagram of the system set up in the diagnostic mode of operation.

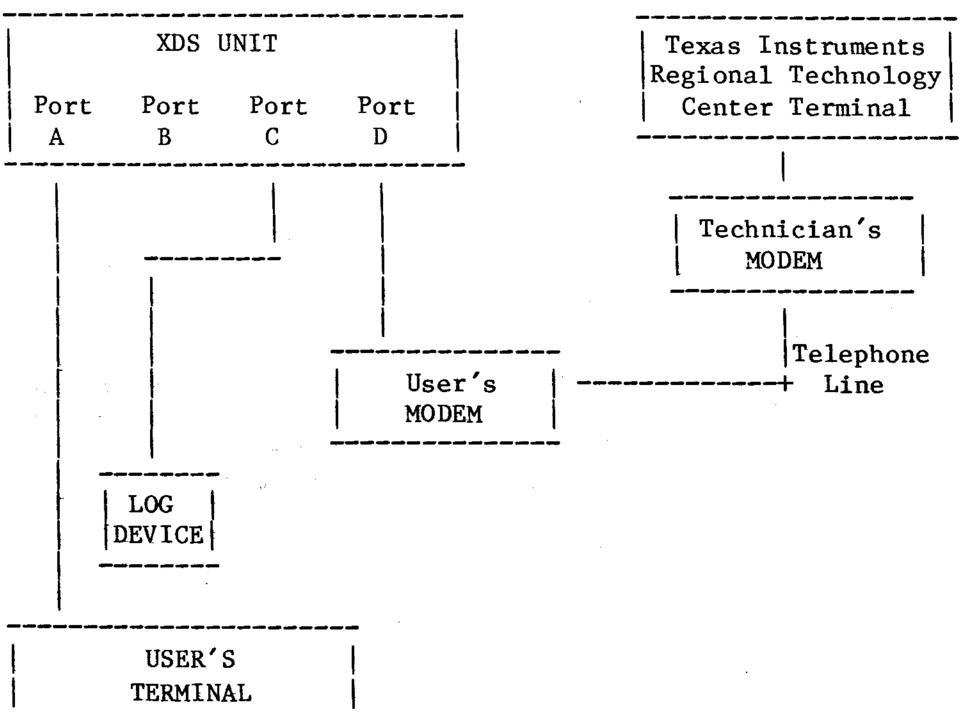


FIGURE A-1. XDS SYSTEM FOR THE DIAGNOSTIC MODE.

A.2 DIAGNOSTIC MODE COMMANDS

When the XDS is in the diagnostic configuration the following commands may be used.

A.2.1 Diagnostic Mode - DIAG

The DIAG command places the XDS system in the diagnostic mode of operation. From this point the user terminal will echo all entries made by the Texas Instruments Regional Technical Center Technician and responses made by the monitor. The operator may not enter any commands. This arrangement may be reversed by entering a 1 for the ECHO ONLY parameter instead of a 0. This discussion of the diagnostic mode of operation will assume that a 0 was entered for the ECHO ONLY parameter.

The parameters of the command are as follows:

Enter: DIAG<CR>

Display: ECHO ONLY (0=NO, 1=YES)

Enter a 0 to turn over control of the system to the RTC terminal.

Enter a 1 to keep control of the system on the user's terminal and echo all entries and responses to the RTC terminal.

A.2.2 Message - MESG

The Message command allows the user and the Texas Instruments RTC technician to communicate with each other. This saves the expense of a second telephone line for communication. When the Texas Instruments RTC technician sends a message it must be answered by the operator. The service technician cannot do anything with the test terminal until the message is answered. When the roles are reversed this command will be executed from the user terminal and the RTC terminal will receive it and enter the answer. An example of a message command follows:

EXAMPLES:

Example 1. Diagnostic Mode (Echo enabled) on User's Terminal.

User's terminal	Service Technician's Terminal
Display: ?	
Enter: DIAG (0)<CR>	
Display: ?	
Enter: MESG<CR> I AM HAVING PROBLEMS<CR> WITH THE DM COMMAND<CR><CR>	I AM HAVING PROBLEMS WITH THE DM COMMAND
	Enter: TRY ENTERING DM<CR><CR>
Display: TRY ENTERING DM	

Comment: The double carriage return transfers control of the message function to the other party. In this example, with Echo enabled, the user maintains control of the monitor and is the only one that can issue commands to the emulator.

Example 2. Echo Mode Disabled

Service Technician's Terminal	User's Terminal
Display: ?	
Enter: MESG<CR> I AM GOING TO DISPLAY<CR> MEMORY<CR> WITH THE DM COMMAND<CR><CR>	I AM GOING TO DISPLAY MEMORY WITH THE DM COMMAND
	Enter: GO AHEAD, I AM READY<CR><CR>
Display: GO AHEAD, I AM READY	

The Service Technician may now enter the DM command and the display will show up on both terminals.

A.2.3 Quit Diagnostics - QDIAG

The quit diagnostic command takes the XDS system out of the diagnostic mode and returns it to normal operation. The QDIAG command is entered by the service technician on the service terminal if the echo is disabled. If the echo is enabled then the QDIAG command must be entered on the user's terminal. After the QDIAG command is executed the user has access to all other stand alone commands again and the service technician is off line. An example of the command is shown below.

Enter: QDIAG<CR>

Display: USER TERMINAL IS NOW A VDT
?
The service technician's terminal is now off line.

APPENDIX B

DATA AND ADDRESS MASKS

Several commands make extensive use of address and data masking. This appendix provides some background information on masks to help the user better understand the concept so that this feature will be of more use.

When the user specifies a mask, it is entered as a hexadecimal value. If the hexadecimal code is converted to its binary equivalent, the "0" bits are considered as "don't care" values. This means that when an address value or a data value is compared to some benchmark value, the bits in the same positions as the 0's in the mask are ignored.

This concept is best illustrated by using an example. Table B-1 illustrates how a mask might be used to reference a range of values. In this example, the benchmark value is 001A and the data mask is FFF8. As shown in Table B-1, the last three bits are 0 or don't care values. All values from 0018 - 001F are included in the masked set. Every position in the mask that has a binary 1 is considered a "do care" value. This means that the binary bit in the value being tested must be identical to that in the benchmark value to be included in the masked set.

TABLE B-1. COMPARING BENCHMARK DATA 001A TO A MASK OF FFF8

Specified DATA:

Hexadecimal	0	0	1	A
Binary	0 0 0 0	0 0 0 0	0 0 0 1	1 0 1 0

Specified DATA MASK:

Hexadecimal	F	F	F	8
Binary	1 1 1 1	1 1 1 1	1 1 1 1	1 0 0 0

Data values included in masked breakpoint/trace set:

Hexadecimal	0	0	1	8
Binary	0 0 0 0	0 0 0 0	0 0 0 1	1 0 0 0
Hexadecimal	0	0	1	9
Binary	0 0 0 0	0 0 0 0	0 0 0 1	1 0 0 1
Hexadecimal	0	0	1	A
Binary	0 0 0 0	0 0 0 0	0 0 0 1	1 0 1 0

TABLE B-1. COMPARING BENCHMARK DATA 001A TO A MASK OF FFF8 (CONTINUED)

Data values included in masked breakpoint/trace set (continued):

Hexadecimal	0	0	1	B
Binary	0 0 0 0	0 0 0 0	0 0 0 1	1 0 1 1
Hexadecimal	0	0	1	C
Binary	0 0 0 0	0 0 0 0	0 0 0 1	1 1 0 0
Hexadecimal	0	0	1	D
Binary	0 0 0 0	0 0 0 0	0 0 0 1	1 1 0 1
Hexadecimal	0	0	1	E
Binary	0 0 0 0	0 0 0 0	0 0 0 1	1 1 1 0
Hexadecimal	0	0	1	F
Binary	0 0 0 0	0 0 0 0	0 0 0 1	1 1 1 1

Table B-1 illustrates that if the three right-most binary digits are set to zero, the masked set consists of eight values. The data mask has enabled the selection of eight possible breakpoint/trace values. As a rule of thumb, every binary 0 in the mask results in two values in the masked set. The number of values included in the masked set may be determined by the following formula.

$$X = 2 ^\wedge N$$

X = number of elements in masked set

N = number of binary 0's in mask

Table B-2 illustrates this principle using a mask with two binary 0's. As shown in the table, only four values are eligible for inclusion in the masked set.

TABLE B-2. COMPARING BENCHMARK DATA 001A TO A MASK OF FFFC

Specified DATA:

Hexadecimal	0	0	1	A
Binary	0 0 0 0	0 0 0 0	0 0 0 1	1 0 1 0

Specified DATA MASK:

Hexadecimal	F	F	F	C
Binary	1 1 1 1	1 1 1 1	1 1 1 1	1 1 0 0

Data values included in the breakpoint/trace masked set:

Hexadecimal	0	0	1	8
Binary	0 0 0 0	0 0 0 0	0 0 0 1	1 0 0 0
Hexadecimal	0	0	1	9
Binary	0 0 0 0	0 0 0 0	0 0 0 1	1 0 0 1
Hexadecimal	0	0	1	A
Binary	0 0 0 0	0 0 0 0	0 0 0 1	1 0 1 0
Hexadecimal	0	0	1	B
Binary	0 0 0 0	0 0 0 0	0 0 0 1	1 0 1 1

APPENDIX C

EXTENDED ADDRESS AND DATA

When the TMS9995 Emulator is initialized using the INIT command, there are two parameters that have an effect on breakpoints and traces involving the memory on the target system. These parameters involve the use of extended addressing which requires the use of the EXTENDED TRACE CABLE. This cable can be attached to the target system by the user in any desired manner.

The first INIT command parameter that is important in extended addressing is the EMULATOR RAM? parameter. This provides the user with the option of using the emulator memory. For designs implementing more than 64K bytes of RAM, Texas Instruments recommends that all of the RAM be supplied by the user and that NO be selected for EMULATOR RAM.

The second parameter used for extended addressing is the NUMBER OF EXTENDED ADDRESS BITS. The extended address bits are used only by the breakpoint-trace board for setting breakpoint or tracing signals on the target system.

When the emulator is powered-up, the number of address bits available for the emulator is 16 bits (for >FFFF or 64K address bytes). This is the number of memory locations available on the TMS9995 Emulator. On power-up, the number of extended address bits is equal to zero bits. The signals that are transmitted over the lines on the EXTENDED TRACE CABLE are considered as data at this point. However, when the NUMBER OF EXTENDED ADDRESS BITS parameter is set to a value greater than zero, the number of data lines decreases by the amount designated as extended address bits. Thus, if the number of extended address bits is set to two bits, the number of data lines decreases by two. This means that there are six lines available as data lines with the other two lines being used for address bits. By enabling the two extended address bits the user is able to trace and breakpoint up to address 3FFFF (256K) locations.

Table C-1 presents a summary of the signals on the EXTENDED TRACE CABLE and the color codes for identifying the proper data line.

TABLE C-1. DATA SIGNAL VALUES FOR EXTENDED TRACE CABLE LINES

Data Bit Number	Color of Line
MSB 7	Violet
. 6	Blue
. 5	Green
. 4	Yellow
. 3	Orange
. 2	RED
. 1	Brown
LSB 0	Black
Ground	White

NOTE

All external address or data bits require that the EXTENDED TRACE CABLE lines be properly connected by the user.

The theory behind the extended address concept might better be explained by using an example. Assume that the target system implements a paging scheme that generates an extra address bit to select the "page of RAM" being accessed. This allows the system to access 128K locations (17 bits or 1FFFF). The total number of bits for breakpoint-trace is 17 bits leaving seven bits for use as extended data probes.

The line carrying the extra address bit must be connected by using the Black line on the EXTENDED TRACE CABLE from the target system to the breakpoint-trace board connector (P3). The regular 16 address bits are transmitted to the breakpoint-trace board by way of the target cable inside the emulator.

Table C-2 summarizes the configuration of the bits on the EXTENDED TRACE CABLE after the NUMBER OF EXTENDED ADDRESS BITS is set to one bit. As shown, Bit #0 (Black) becomes the Most Significant Bit for the extended address and Bit #1 (Brown) becomes the Least Significant Bit for the extended data bits.

TABLE C-2. EXTENDED ADDRESS BITS = 1, EXTENDED DATA BITS = 7

Color	Bit Number	Address/Data
Violet	7 (MSB)	Data
Blue	6 .	Data
Green	5 .	Data
Yellow	4 .	Data
Orange	3 .	Data
Red	2 .	Data
Brown	1 (LSB)	Data
Black	0 (MSB)	Address

In this example the lines for extended data bits are 'manually' connected to desired data points on the user's target hardware. The extended address line (Black) must be connected to the MSB address bit on the target system.

To summarize the use of extended addressing, the following items should be noted.

* The number of extended address bits is determined when the INIT command is executed. It is recommended that the number of extended address bits should not be changed during a session. If INIT is reexecuted and the number of address bits is changed, then all parameters must be reset and BP,BPM, BPIO, TR, TRM, TRIO, TRIX must be reevaluated.

* Extended addressing only applies to hardware breakpoints and traces through the use of the breakpoint-trace board.

* The total number of extended address bits and extended data bits is always eight.

* Connections involving the EXTENDED TRACE CABLE are completely under the control of the user.

* If the extended address bits are used to address memory locations, this memory must be supplied on the user's target system. The maximum number of emulator memory locations is 64K bytes.

APPENDIX D

OBJECT FILE FORMATS

D.1 TI Object Record Format

When TI data formats are specified, the object code field values begin with tags which are used to specify characteristics of the value in a given field. Generally, if TI ASCII code is specified as the data format, each field is four characters long. When TI-compressed format is designated each field contains four binary digits. There are some exceptions to this rule and these are noted in Table D-1 below. This table summarizes the tags used in TI-formatted data. The beginning-of-module tag is an ASCII 0 (0 tag)in standard object code and a binary one (1) in compressed object format. This tag is used to distinguish compressed and uncompressed object modules during uploads and downloads when TI formats are selected.

When object code is downloaded into the emulator's memory, the tags inform the monitor to perform one of several operations on the data that follows the tag. In some cases, as noted in Table D-1 below, the TMS9995 Emulator ignores the tag and does not perform the directed operation.

TABLE D-1. TI FORMATTED OBJECT RECORDS AND TAGS IN THE TMS9995 EMULATOR

Tag	Field 1	Field 2	Field 3	Emulator Action
***	Module Definition			
0	PSEG Length	Program ID(8)	-	Ignored
M	DSEG Length	$DATA	0000	Ignored
M	Blank Common Len.	$BLANK	Common #	Ignored
M	CSEG Length	Common Name(6)	Common #	Ignored
M	CBSEG Length	$CBSEG	CBSEG #	Ignored
***	Entry Point Definition			
1	Absolute Address	-	-	Ignored
2	P-R Address	-	-	Ignored
***	Load Address			
9	Absolute Address	-	-	Processed
A	P-R Address	-	-	Processed
S	D-R Address	-	-	Processed
P	C-R Address	Common or CBSEG #	-	Processed
***	Data			
B	Absolute Value	-	-	Processed
C	P-R Address	-	-	Processed
T	D-R Address	-	-	Processed
N	C-R Address	Common or CBSEG #	-	Processed
***	External Definitions			
6	Absolute Value	Symbol(6)	-	Ignored
5	P-R Address	Symbol(6)	-	Ignored
W	D-R/C-R Address	Symbol(6)	Common #	Ignored
***	External References			
3	P-R Chain Address	Symbol(6)	-	Error
4	Absol. Chain Add.	Symbol(6)	-	Error
X	D-R/C-R Chain Add.	Symbol(6)	Common #	Error
E	Symbol Index No.	Absolute Offset	-	Error
***	Symbol Definitions			
G	P-R Address	Symbol(6)	-	Ignored
H	Absolute Value	Symbol(6)	-	Ignored
J	D-R/C-R Address	Symbol(6)	Common #	Ignored
***	Force External Link			
U	0000	Symbol(6)	-	Ignored

TABLE D-1. TI FORMATTED OBJECT RECORDS AND TAGS IN THE TMS9995 EMULATOR (CONTINUED)

Tag	Field 1	Field 2	Field 3	Emulator Action
***	Secondary External Reference			
V	P-R Address of Chain Entry	Symbol(6)	-	Error
Y	Absolute Address of Chain	Symbol(6)	-	Error
Z	D-R/C-R Address of Chain	Symbol(6)	Common #	Error
Tag	Field 1	Field 2	Field 3	Emulator Action
***	Checksum			
7	Value	-	-	Processed
***	Ignore Checksum			
8	Any Value	-	-	Processed
***	Load Bias			
D	Absolute Address	-	-	Processed
***	End of Record			
F	-	-	-	Processed
***	Repeat Count			
R	Value	Repeat Count	-	Processed
***	Program ID(?)			
I	P-R Address	Program ID(8)	-	Ignored
***	COBOL Segment Reference			
Q	Record Offset	CBSEG #	-	Ignored
***				

The tag characters and the operations that are most commonly encountered during emulation procedures are explained below.

Tag character 0 is specific for the TMS9995 microprocessor. This 0 tag character is followed by two fields. The first field contains the number of bytes in the program segment and the second eight-character field contains the program identifier assigned to the program by an assembly directive.

Tag characters 9 and A are used with load addresses for data that follows. Tag character 9 is used when the load address is absolute. Tag character A is used for relocatable code. The hexadecimal field that follows the tag contains the address at which the subsequent data word is to be loaded.

Tag characters B and C are used with data words that are to be stored in memory. Tag character B is used when the data value is absolute (for example, an instruction word or a word that contains text characters or absolute constants). Tag character C is used for words that contain relocatable addresses. The hexadecimal field contains the data word. The data word is placed in the memory location specified in the preceding load address field, or in the memory location that follows the word that was loaded last.

Tag character 7 precedes the checksum, which is an error detection word. The checksum is formed as the record is being written. It is the twos complement of the sum of the eight-bit ASCII values of each character in the object record from the first tag character of the record through (and including) the checksum tag 7. If the checksum field is preceded by the tag character 8 (instead of 7) the checksum is ignored. The 8 tag can be used when object code is changed in editing and the programmer wants to ignore the checksum value.

All external reference tag characters (3, 4, X, E, V, Y, and Z) indicate to the monitor that some unresolved condition exists. These tags will generate error codes. When these tags are encountered in an object file, the monitor reads through the tags and continues the down load. When the HOST mode is exited, the monitor prints the error message "EXTERNAL REF FOUND IN OBJECT MODULE." Only the external reference tags will cause such an error condition. If the user has downloaded more than one object file during a single HOST session, there is no way to determine which file contained the external reference tag.

Tag character F indicates the end of record and can be followed by ASCII characters.

The last record of an object code file begins with a colon (:) in the first character position. This record is called the End-of-Module separator record.

An example of a data stream downloaded in TI format is illustrated below. The tag characters are underlined in the example to enhance understanding of their functions.

```
00008TASK    A0000B000AB0200B0000BC0008F7EEF
:      TASK          021/83    12:32:54
```

ASCII representation of TI-Standard Object-Code Format. (First record only.)

```
00  00  8T  AS  K       A   00
00  B0  00  AB  02  00  B0  00
0B  C0  00  8F  7E  EF
```

Hexadecimal representation of TI-Standard Object-File Format. (First record only.)

```
0030  3030  3854  4153  4B20  2020  2041  3030
3030  4230  3030  4142  3032  3030  4230  3030
3042  4330  3030  3846  3745  4546  2020  2020
```

Hexadecimal representation of TI-Compressed Object-File Format. (First record only.)

```
3000  0854  4153  4B20  2020  2041  0000  4200
0A42  0200  4200  0042  C000  38F7  EE46  2020
```

ASCII representation of TI-Compressed Object-Code Format.

```
0.  .T  AS  K       A   ..  B.
.B  ..  C.  .B  ..  8.  .F
```

D.2 Tektronix-Hexadecimal Format

The Tektronix-Hexadecimal format uses sets of message blocks to encode data in ASCII form. The permitted message blocks fall into three categories: data blocks, termination blocks, and abort blocks. Data is encoded as a series of one or more data blocks. Normal termination of the data blocks is indicated by a single termination block. Abnormal termination can be indicated anywhere by means of an abort block.

Figure D-1 illustrates a valid Tektronix-Hexadecimal formatted data file. Each data record starts with a header character which must be a slash (/). The next four characters express the address of the first data byte. The byte count follows this address field and represents the number of bytes in the record. The first checksum field follows the byte count and is calculated from the sum of the address field and the byte count. The data bytes follow the checksum and are represented by a pair of hexadecimal characters. The data bytes are terminated by a second checksum which is calculated from the sum of the data bytes.

The End-of-Record is denoted by a carriage return. The End-of-File or termination record consists of control characters used to signal the end of transmission, a byte count, and a checksum for verification. An abort record starts with two slashes (//) and generates an error message. Tables D-2, D-3, and D-4 summarize the formats for the three Tektronix-Hexadecimal records.

```
| / | AAAA | BC | CS | DB | DB |......| CS | CR |
| 1 |  2   | 3  | 4  | 5  | 5  |      | 6  | 7  |
+-----------------------------------------------+
```

Field	Label	Designation
1	/	Start character (always /)
2	AAAA	Starting address for first data byte
3	BC	Number of data bytes in record
4	CS	First checksum using Fields 2 and 3
5	DB	Data byte
6	CS	Second checksum (sum of data bytes)
7	CR	Carriage return (end-of-record)

FIGURE D-1. TEKTRONIX HEXADECIMAL FORMATTED DATA RECORD

TABLE D-2. SUMMARY OF TEKTRONIX HEXADECIMAL FORMATTED DATA RECORD

Field Name	No. of ASCII Characters	Description
Header	1	Header or start character, always a /
Address	4	Address of the first data byte in the record (in hexadecimal notation)
Byte count	2	The hexadecimal number of data bytes in the record
Checksum 1	2	The eight-bit sum of the four-bit hexadecimal values of the six digits that make up the address and byte count (in hexadecimal notation)
Data byte		One data byte in hexadecimal notation per field
Checksum 2	2	The eight-bit sum, modulo 256, of the four-bit hexadecimal values of the digits that make up the data bytes
Carriage Return		Carriage return (end-of-record)

TABLE D-3. SUMMARY OF TEKTRONIX HEXADECIMAL TERMINATION RECORD

Field Name	No. of ASCII Characters	Description
Header	1	Header or start character, a slash in Termination Record
Address	4	Transfer Address
Byte count	2	The byte count is 00 in Termination Record
Checksum	2	The eight-bit sum of the four-bit hexadecimal values of the six digits that make up the address and byte count (in hexadecimal notation)
Carriage Return		Carriage return (end-of-record)

TABLE D-4. SUMMARY OF TEKTRONIX HEXADECIMAL ABORT RECORD

Field Name	No. of ASCII Characters	Description
Header	1	Header or start character, for the Abort Record this is two slashes (//)
Data Bytes	2	An arbitrary string of meaningless ASCII characters follows the // header
Carriage Return		Carriage return (end-of-record)

The following example illustrates a typical set of data records transmitted in Tektronix Hexadecimal Format.

```
/AAAABCCSDBDBDBDBDBDBDBDBDBDBDBDBCSCR
/AAAABCCSDBDBDBDBDBDBDBDBDBDBDBDBCSCR
/AAAABCCSDBDBDBDBDBDBDBDBDBDBDBDBCSCR
/AAAABCCSDBDBDBDBDBDBDBDBDBDBDBDBCSCR
/AAAABCCSDBDBDBDBDBDBDBDBDBDBDBDBCSCR
/AAAABCCSCR
```

The last record is a termination record.

D.3 Intel Data Record Format

Intel data records are used only in uploads to a host computer (via port D) or to a PROM programmer (via port C).

Intel data records begin with a nine-character prefix and end with a two-character suffix. The prefix consists of a start character (:), the byte count, the address for the first data byte, and the record type (00 = data record, 01 = termination record). The suffix consists of a second checksum field. The End-of-Record is a carriage return.

Figure D-2 illustrates the contents of a typical Intel object data file. Each record begins with a colon (:) followed by a two-character byte count. The four digits that follow the byte count give the memory address of the first data byte. This field is followed by a file-type designation in a one-byte hexadecimal representation. The data follows the file-type designator and each data byte is represented by two hexadecimal digits. The number of data bytes in each record must equal the byte count.

The suffix is a two-digit checksum which is the twos complement of the binary summation of the previous bytes in the record. The final field in the record may contain a line feed, carriage return, or comments.

:	BC	AAAA	TT	DB	DB	......	CS	
1	2	3	4	5	5		6	7

Field	Label	Designation
1	:	Start character (always :)
2	BC	Number of data bytes in record
3	AAAA	Starting address for first data byte
4	TT	Record type (0 = Data, 1 = Terminate)
5	DB	Data byte
6	CS	Checksum = Twos complement of summation of preceding bytes in the record
7	Space	Carriage return, line feed, or comments (end-of-record)

FIGURE D-2. INTEL INTELLEC 8/MDS FORMATTED DATA RECORD

TABLE D-5. SUMMARY OF INTEL INTELLEC 8/MDS FORMATTED DATA RECORD

Field Name	No. of ASCII Characters	Description
Start	1	Start character, always a :
Byte count	2	The hexadecimal number of data bytes in the record
Address	4	Address of the first data byte in the record (in hexadecimal notation)
Record Type	2	A one-byte code to designate the type of record (0 = Data Record, 1 = End-of-File Record)
Data byte	1	One data byte in hexadecimal notation per field
Checksum	2	The negation (twos complement) of the eight-bit sum of the four-bit hexadecimal values of the digits of the preceding data bytes
Space		This space may contain a line feed, carriage return, or comments to indicate the end-of-record code

TABLE D-6. SUMMARY OF INTEL INTELLEC 8/MDS FORMATTED TERMINATION RECORD

Field Name	No. of ASCII Characters	Description
Header	1	Header or start character, always a :
Byte count	2	The byte count is 00 in End-of-File record
Address	4	Address of the first data byte in the record (in hexadecimal notation)
Record Type	2	01 = End-of-File Record

The following example illustrates a typical set of records transmitted in Intel Intellec 8/MDS Format.

```
:BCAAAATTDBDBDBDBDBDBDBDBDBDBCSCR
:BCAAAATTDBDBDBDBDBDBDBDBDBDBCSCR
:BCAAAATTDBDBDBDBDBDBDBDBDBDBCSCR
:BCAAAATTDBDBDBDBDBDBDBDBDBDBCSCR
:BCAAAATT
```

The last record is the End-of-File record.

D.4 Motorola Record Formats

Motorola record formats are used only when uploading to a host computer (via port D) or to a PROM programmer (via port C). The TMS9995 emulator supports two record types: data records and termination records.

Valid Motorola data records start with an eight-character prefix that consists of start characters, byte count, and beginning address location. Each record ends with a two-character suffix consisting of the checksum which is followed by a space.

Each valid data record begins with the start characters S1 which indicates the start of the file. The next data field in the prefix contains the number of bytes in the record. This number includes the number of data, address, and checksum bytes in the record. The byte-count field is followed by the last field in the prefix, which is the address of the first data byte in the record. The prefix is followed by a variable number of data characters, each represented by two hexadecimal digits. The number of data bytes occurring must be three less than the byte count.

The suffix is a two-character checksum which is the hexadecimal representation of the ones complement of the binary summation of the preceding bytes in the record, including the byte count, address, and data bytes. The space that follows the checksum may contain a line-feed character, a carriage return, or comments. Figure D-3 illustrates a typical Motorola formatted data record and Table D-7 summarizes the data fields that make up the Motorola data record.

S1	BC	AAAA	DB	DB	DB	......	CS	
1	2	3	4	4	4		5	6

Field	Label	Designation
1	S1	Start character sequence (S1 for data record)
2	BC	Number of data bytes in record, + 3
3	AAAA	Starting address for first data byte
4	DB	Data byte
5	CS	Checksum (ones complement of the sum of data bytes, the address bytes, and the byte count).
6	Space	May be a carriage return, line feed, or comments (end-of-record)

FIGURE D-3. MOTOROLA EXORCISOR FORMATTED DATA RECORD

TABLE D-7. SUMMARY OF MOTOROLA EXORCISOR FORMATTED DATA RECORD

Field Name	No. of ASCII Characters	Description
Start	1	Start character sequence; S1 for data record, S0 for optional Sign-on Record, S9 for End-of-File Record
Byte count	2	The hexadecimal number of data bytes in the record, + 3
Address	4	Address of the first data byte in the record (in hexadecimal notation)
Data byte	1	One data byte in hexadecimal notation per field
Checksum 2	2	The ones complement of the binary summation of the preceding bytes in the record including the data bytes, the address bytes, and the byte count
Space		This space may contain a line feed, carriage return, or comments to indicate the end-of-record code

TABLE D-8. SUMMARY OF MOTOROLA EXORCISOR FORMATTED TERMINATION RECORD

Field Name	No. of ASCII Characters	Description
Start	1	Start character for End-of-File record is S9
Byte count	2	The byte count is 00 in End-of-File record
Address	4	Address of the first data byte in the record (in hexadecimal notation)
Checksum	2	The checksum is one's complement of byte count, address, and data bytes

The following example illustrates a typical set of records transmitted in Motorola Exorcisor format.

```
S1BCAAAADBDBDBDBDBDBDBDBDBDBDBDBCSCR
S1BCAAAADBDBDBDBDBDBDBDBDBDBDBDBCSCR
S1BCAAAADBDBDBDBDBDBDBDBDBDBDBDBCSCR
S1BCAAAADBDBDBDBDBDBDBDBDBDBDBDBCSCR
S9BCAAAACS
```

The last record is the End-of-File record.

APPENDIX E

INTERRUPT HANDLING

E.1 INTERRUPT HANDLING

This appendix explains some of the details concerning the manner in which the XDS TMS9995 Emulator handles target system interrupts. It covers the following subjects:

* Outstanding interrupts prior to program execution
* Stopping the emulator with Non-Maskable Interrupts (NMI) active

E.2 BEGINNING PROGRAM EXECUTION WITH TARGET INTERRUPTS OUTSTANDING

Any attempt at program execution (RUN, CRUN, SS, etc.) will always execute one instruction before an interrupt is recognized, even if the interrupt is active before execution begins. The significance of this can be explained with the following situation.

Suppose an interrupt is active and program execution begins at an 'IDLE' instruction. Since the IDLE instruction is executed (i.e., the program counter is incremented past the IDLE instruction) before servicing the interrupt, control will return just past the idle instruction when the interrupt service is complete, instead of returning control at the IDLE instruction.

E.3 HALTING EXECUTION WITH NMI ACTIVE

If program execution is halted with NMI active, the NMI signal to the processor is gated OFF. If program execution is then resumed, with NMI still active from the target system, the processor will reexecute the NMI trap. In order to avoid this, the NMI service code should remove NMI, or NMI should be removed before restarting program execution.